Lois Delcambre
Christian Kop
Heinrich C. Mayr
John Mylopoulos
Oscar Pastor (Eds.)

LNCS 3716

Conceptual Modeling – ER 2005

24th International Conference on Conceptual Modeling
Klagenfurt, Austria, October 2005
Proceedings

Springer

Lois Delcambre Christian Kop
Heinrich C. Mayr John Mylopoulos
Oscar Pastor (Eds.)

Conceptual Modeling – ER 2005

24th International Conference on Conceptual Modeling
Klagenfurt, Austria, October 24-28, 2005
Proceedings

Springer

Volume Editors

Lois Delcambre
Portland State University, Computer Science Department
P.O. Box 751 Portland, OR 97207-0751, USA
E-mail: lmd@cs.pdx.edu

Christian Kop
Heinrich C. Mayr
Alpen-Adria Universität Klagenfurt
Institute of Business Informatics and Application Systems
Klagenfurt, Austria
E-mail: {chris,mayr}@ifit.uni-klu.ac.at

John Mylopoulos
University of Toronto, Bahen Center for Information Technology
40 St George Street, Room BA7266
Toronto, Ontario M5S 2E4, Canada
E-mail: jm@cs.toronto.edu

Oscar Pastor
Universidad Politécnica de Valencia
Dept. de Sistemas Informáticos y Computación
Camino de Vera s/n, 46022 Valencia, España
E-mail: opastor@dsic.upv.es

Library of Congress Control Number: 2005934479

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

ISSN	0302-9743
ISBN-10	3-540-29389-2 Springer Berlin Heidelberg New York
ISBN-13	978-3-540-29389-7 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11568322 06/3142 5 4 3 2 1 0

Preface

Conceptual modeling is fundamental to any domain where one must cope with complex real-world situations and systems because it fosters communication between technology experts and those who would benefit from the application of those technologies. Conceptual modeling is the key mechanism for understanding and representing the domains of information system and database engineering but also increasingly for other domains including the new "virtual" e-environments and the information systems that support them. The importance of conceptual modeling in software engineering is evidenced by recent interest in "model-driven architecture" and "extreme non-programming". Conceptual modeling also plays a prominent role in various technical disciplines and in the social sciences.

The Annual International Conference on Conceptual Modeling (referred to as the ER Conference) provides a central forum for presenting and discussing current research and applications in which conceptual modeling is the major emphasis. In keeping with this tradition, ER 2005, the 24th ER Conference, spanned the spectrum of conceptual modeling including research and practice in areas such as theories of concepts and ontologies underlying conceptual modeling, methods and tools for developing and communicating conceptual models, and techniques for transforming conceptual models into effective (information) system implementations. Moreover, new areas of conceptual modeling including Semantic Web services and the interdependencies of conceptual modeling with knowledge-based, logical and linguistic theories and approaches were also addressed.

The Call for Papers attracted 169 research papers from 37 different nations; 31 papers from 22 nations, i.e., 21.9%, were selected for presentation at the conference and publication in these proceedings based on a stringent review process in which each paper was assessed by at least three reviewers. These accepted papers, together with three invited keynote speeches, a demo and poster session, and a concluding panel discussion, were featured in 14 technical conference sessions. ER 2005 also featured five workshops organized in 15 technical sessions and 7 tutorials presented by outstanding experts in their fields. We were enthusiastic about the quality of this year's program in all its particulars.

Many individuals contributed to making ER 2005 a success. First, we thank the authors for their valuable contributions. Second, we thank the members of the Program Committee and the additional reviewers for their detailed reviews and discussion. Special appreciation is due to our last-minute reviewers, i.e., the "Swat Team," who provided additional reviews for papers, nearly around the clock, in parallel with the PC chairs meeting. Similarly, we thank the chairs of the various tracks for their effectiveness. And we offer special thanks to our keynote speakers for their insightful contributions.

We are very grateful to Peter Jelitsch, our student who composed these proceedings and painstakingly adapted nearly every paper to the LNCS layout. Likewise we acknowledge the engagement and enthusiasm of all members of the organizational team, who gave their best to make ER 2005 an unforgettable event. Last but not least we thank our sponsors and supporters, in particular the University of Klagenfurt, the Governor of Carinthia and the Mayor of Klagenfurt, for their financial support.

Klagenfurt, October 2005

Heinrich C. Mayr
Lois Delcambre
John Mylopoulos
Oscar Pastor
Christian Kop

ER 2005 Conference Organization

Honorary Conference Chair

Peter P.S. Chen	Lousiana State University, USA

General Conference Chair

Heinrich C. Mayr	Alpen-Adria University of Klagenfurt, Austria

Scientific Program Co-chairs

Lois M.L. Delcambre	Portland State University, Portland, USA
John Mylopoulos	University of Toronto, Canada
Oscar Pastor López	Universitat Politècnica de València, Spain

Workshop and Tutorial Co-chairs

Jacky Akoka	CEDRIC – CNAM, France/Institut National des Télécommunications, Evry, France
Stephen W. Liddle	Brigham Young University, Provo, USA
Il-Yeol Song	Drexel University, Philadelphia, USA

Panel Chair

Wolfgang Hesse	Philipps-Universität Marburg, Germany

Demos and Posters Chair

Tatjana Welzer	University of Maribor, Slovenia

Industrial Program Chair

Andreas Schabus	Microsoft Austria, Vienna, Austria

ER Steering Committee Liaison Manager

Veda Storey	Georgia State University, USA

Joint Conferences Steering Committee Co-chairs

Ulrich Frank	University of Essen-Duisburg, Germany
Jörg Desel	Catholic University Eichstätt-Ingolstadt, Germany

Organization and Local Arrangements

Markus Adam	Peter Jelitsch	Christine Seger
Stefan Ellersdorfer	Christian Kop	Claudia Steinberger
Günther Fliedl	Heinrich C. Mayr	
Robert Grascher	Alexander Salbrechter	

Program Committee

Jacky Akoka	CEDRIC – CNAM, France/Institut National des Télécommunications, Evry, France
Sonia Bergamaschi	Università di Modena, Italy
Shawn Bowers	University of California, San Diego, USA
Terje Brasethvik	NTNU, Trondheim, Norway
Ruth Breu	University of Innsbruck, Austria
Diego Calvanese	Free University of Bozen – Bolzano, Italy
Cindy Chen	University of Massachusetts, Lowell, USA
Jaelson Brelaz de Castro	Federal University of Pernambuco, Brazil
Shing-Chi Cheung	HKUST, China
Roger Chiang	University of Cincinnati, USA
Stefan Conrad	Heinrich-Heine-Unversität Düsseldorf, Germany
Joao Falcao e Cunha	Universidade do Porto, Portugal
Bogdan Czejdo	Loyola University New Orleans, USA
Karen Davis	University of Cincinnati, USA
Debabrata Dey	University of Washington, USA
Johann Eder	Alpen-Adria University of Klagenfurt, Austria
Ramez Elmasri	University of Texas at Arlington, USA
David W. Embley	Brigham Young University, Provo, USA
Vadim Ermolayev	Zaporozhye State Univ., Ukraine
Ulrich Frank	University of Essen-Duisburg, Germany
Piero Fraternali	Politecnico di Milano, Italy
Antonio L. Furtado	PUC Rio de Janeiro, Brazil
Andreas Geppert	Credit Suisse, Switzerland
Nicola Guarino	CNR, Trento, Italy
Terry Halpin	Northface Univ., Salt Lake City, USA
Sari Hakkarainen	NTNU, Trondheim, Norway
Brian Henderson-Sellers	University of Technology, Sydney, Australia
Shigeichi Hirasawa	Waseda University, Japan

Katsumi Tanaka	Kyoto University, Japan
Ernest Teniente	Universitat Politècnica de Catalunya, Spain
Bernhard Thalheim	Christian-Albrechts-Universität Kiel, Germany
Dimitri Theodoratos	New Jersey Institute of Technology, USA
Juan C. Trujillo	Universidad de Alicante, Spain
Jean Vanderdonckt	Université Catholique de Louvain, Belgium
Michalis Vazirgiannis	Athens University of Economics and Business, Greece
Csaba Veres	NTNU, Trondheim, Norway
Yair Wand	University of British Columbia, Vancouver, Canada
Tengjiao Wang	Peking University, China
Roel Wieringa	University of Twente, The Netherlands
Ge Yu	Northeastern University, China
Shuigeng Zhou	Fudan University, China

External Referees

Reema Al-Kamha
Muhammed Al-Muhammed
Evguenia Altareva
Anastasia Analyti
Danilo Ardagna
Roberta Benassi
Domenico Beneventano
Palash Bera
Ghassan Beydoun
Andreas Boegl
Marco Brambilla
Agne Brilingaite
Giovanni Toffetti Carughi
José María Cavero
Kevin C. Chang
Nam Yoon Choi
Ryan Choi
Sara Comai
Nelly Condori
Valeria de Castro
Cristian Pérez de Laborda
M. de Rougemont
Yihong Ding
Wanchun Dou
Vishal Dwivedi
Magdalini Eirinaki
Maged El-Sayed
Joerg Evermann
Eduardo Fernandez-Medina
Roberta Ferrario
Anders Friis-Christensen
Mathias Goller
Cesar Gonzalez-Perez
Masayuki Goto
Georg Grossmann
Francesco Guerra
Maria Halkidi
Lillian Hella
Wiebe Hordijk
John Horner
Jon Espen Invaldsen
Jürgen Jung
Lutz Kirchner
Christian Koncilia
Saoujanya Lanka
Bo Luo
Andreia Malucelli
Juergen Mangler
Daisuke Matsushita
Raimundas Matulevicius
Enrico Mussi
John Mylopoulos

"Swat Team"

Organized by

Institute of Business Informatics and Application Systems, Alpen-Adria University of Klagenfurt, Austria

Sponsored by

ER Institute
The Governor of Carinthia
The City Mayor of Klagenfurt

In Cooperation with

GI Gesellschaft für Informatik e.V.
Austrian Computer Society

Table of Contents

Temporal and Spatial Modeling

Conceptual Modeling of Structure and Behavior with UML – The Top Level Object-Oriented Framework (TLOOF) Approach

Iris Reinhartz-Berger

University of Haifa, Carmel Mountain, Haifa 31905, Israel
iris@mis.hevra.haifa.ac.il

Abstract. In the last decade UML has emerged as the standard object-oriented conceptual modeling language. Since UML is a combination of previous languages, such as Booch, OMT, Statecharts, etc., the creation of multi-views within UML was unavoidable. These views, which represent different aspects of system structure and behavior, overlap, raising consistency and integration problems. Moreover, the object-oriented nature of UML sets the ground for several behavioral views in UML, each of which is a different alternative for representing behavior. In this paper I suggest a Top-Level Object-Oriented Framework (TLOOF) for UML models. This framework, which serves as the glue of use case, class, and interaction diagrams, enables changing the refinement level of a model without losing the comprehension of the system as a whole and without creating contradictions among the mentioned structural and behavioral views. Furthermore, the suggested framework does not add new classifiers to the UML metamodel, hence, it does not complicate UML.

1 Introduction

Conceptual modeling is fundamental to any domain where one has to cope with complex real world systems. The real world exhibits two separate aspects: structure (objects, nouns, etc.) and behavior (operations, verbs, etc.). Although different aspects, structure and behavior are highly intertwined in the real world: operations get input objects, operations might change structures, sentences include both nouns and verbs, and so on. In spite of the differences between these two aspects in the real world, in the last few decades most of the modeling and programming languages are object-oriented, encapsulating behavior (operations) in structure (objects). The most popular, de-facto standard modeling language is the Unified Modeling Language (UML), which is used for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems [16]. UML 1.x defines ten types of diagrams, which are divided into three categories. Four diagram types represent structure and include the *class*, *object*, *component*, and *deployment diagrams*. Five diagram types, which represent different aspects of dynamic behavior, include *use case*, *sequence*, *activity*, *collaboration* (called communication diagrams in UML 2.0), and *Statechart diagrams*. Finally, *package diagrams* represent ways to organize and manage application modules. The UML 2.0 draft [17] adds three more diagram types: *timing diagrams* for exploring the behaviors of one or more objects throughout a given period of time, *composite structure diagrams* for exploring run-time instances collaborating over communication links, and *interaction overview diagrams* for tailoring interactions and activity diagrams in order to describe the system flow of control. A system modeled by UML consists of several different, but related and even overlapping, diagrams of various types.

The popularity UML gained and the standardization efforts of its creators made UML a common modeling language which is used in the various steps of system development, including during the requirement analysis, design, and testing phases. Many automatic code generators have been developed for generating code from UML models to various (especially object-oriented) programming languages. These usages of UML models require that they will be formal, complete, unambiguous, and consistent, otherwise, their interpretation into code "may lead to misunderstandings and errors that will result in faulty software." [19]

Although UML provides a convenient, standard mechanism for software engineers to represent high-level system designs as well as low-level implementation details [24], several drawbacks prevent UML from being largely used in the industry. The main drawbacks are the fragmentation of UML views and the absence of solid glue between them, which arise syntactic and semantic *consistency problems*. UML syntactic rules relate to well-formedness of expressions, consistency of identifiers with their declarations, etc. Such rules, which are expressed in the UML metamodel, can be checked by diagram editors or CASE tools. Semantic consistency, on the other hand, is concerned with the compatibility of the meaning of the different views. Engels et al pointed two types of semantic consistency: horizontal and vertical [8]. *Horizontal consistency* refers to rules that should be preserved when traveling between different (overlapping) viewpoints of the same system, while *vertical consistency* concerns with rules that should be preserved during the different development stages.

The consistency problems of UML are also associated with *integration problem*s. The different UML views represent a single system. Humans engage in the development process, such as clients, users, designers, and implementers, should comprehend the system as a whole, complete unit. Moreover, automatic tools, for example code generators, should be able to generate a consistent, qualitative implementation from a UML multiple view model.

Several solutions for UML consistency and integration problems have been proposed over the years (e.g., [3], [5], [7], [14]). Most of them suggested using a formal language in addition to UML or running translation, verification, or testing algorithms on existing UML models. In this paper, I suggest a Top-Level Object-Oriented Framework (TLOOF) for creating complete, coherent UML models which capture both system structure and behavior. As a framework, TLOOF consists of a set of assumptions, concepts, and practices that constitutes a way of viewing and constructing UML models. In particular, this approach enables explicit bindings between UML use case, class, and interaction diagrams, thereby supporting incremental development of consistent and integral UML models. A set of consistency rules between these UML views is defined and exemplified.

The structure of the rest of the paper is as follows. Section 2 reviews and discusses the main consistency and integration problems of UML and some of their solutions.

Section 3 presents the proposed approach, exemplifying it on a simple ordering system. Section 4 defines consistency rules between UML views in the TLOOF approach. Finally, Section 5 summarizes and discusses the benefits and shortcomings of the suggested solution and refers to future research plan.

2 Literature Review: Consistency and Integrity of UML Models

2.1 UML Consistency and Integration Problems

The need to model and design complex systems, which involve structural, behavioral, functional, and architectural aspects, introduced the notion of a *view*. Each view presents a different perspective of the system being developed. The actual views and the way in which system aspects are projected onto individual views are method- or language-dependent [10]. Although the usage of multiple views has great benefits in focusing on a specific aspect of the modeled system and in preserving the views in a reasonable size, it also raises consistency and integration problems.

Paige and Ostroff [19] differ between the single model approach, which supports the construction of a single consistent model that can be separated into various views, and the multiple models approach, which is based on independent construction of multiple models of the same system. They propose that "to best support seamless, reversible software development of reliable software, it is preferable to follow the single model principle for a specific subset of development tasks."

Henderson-Sellers [13] reviewed several object-oriented modeling languages that combine several diagram types (each), including Booch, OMT, BON, Fusion, etc., and checked how the different diagram types fit together to create a unified, multi-dimensional model. His conclusion was that at least at the meta-level of description, there is significant convergence and agreement on how the suite of diagrams is connected.

As noted, Engels et al [8] divide UML consistency problems into horizontal and vertical ones. Horizontal consistency problems (also known as inter-model consistency problems) refer to contradictions that might exist due to the fact that the various views model the same system and the information resides at them overlaps. An example of a constraint related to horizontal consistency is: "Each Statechart must correspond to a state dependent class on a class diagram" [10]. The vertical consistency problems refer to inconsistencies or contradictions that exist when applying UML to the different development stages (due to the different abstraction levels of these stages). An example for this type of constraints is: "The information needed for implementing a use case must be described in a class diagram" [10]. While usually the data needed for checking horizontal consistency is explicitly modeled in the UML views, some of the information needed for verifying vertical consistency is implicit or expressed informally.

Another problem that exists due to the use of multiple UML views is misunderstanding of the system as a whole (i.e., *integration problems*). Using their framework for evaluating system analysis and design methods, Tun and Bielkowicz [25] claim that UML views (diagram types) are fragmented and there is little glue between them. Moreover, they assert that without rigorous crosschecking between the views, it would be hard to have confidence that the system would possess essential quality characteristics such as completeness, correctness, and consistency. Two experiments which compared a single-view methodology, Object-Process Methodology (OPM) [6], to multi-view modeling languages, Object Modeling Technique (OMT) and UML ([20] and [22], respectively) showed that the single view of OPM is more effective than the multiple view modeling language in generating a better system specification. Most of the errors in the multiple view models resulted from the need to maintain consistency among the different diagram types and to gather information that is scattered across the views.

The consistency and integration problems of UML are also influenced from the existence of several behavioral views in UML, some of which represent specific scenarios rather than complete behavioral patterns. Uchitel et al [26] proposed an algorithm for synthesizing behavioral models from UML scenarios. Their algorithm translates a scenario specification to a Finite Sequential Processes (FSP) specification, which is then used for building a composite behavior model in the form of a labeled transition system (LTS).

Several studies checked if there is any preference between UML behavioral views. Otero and Dolado [18], for example, compared the semantic comprehension of sequence, collaboration, and state diagrams. The comparison was done in terms of the total time to complete tasks and their scores. They found that the comprehension of behavioral models in object-oriented designs depends on the complexity of the system. However, using sequence diagrams is the most comprehensible way to represent the system behavior. Hahn and Kim [12] conducted an experiment to check the effects of diagrammatic representation on the cognitive integration process of systems analysis and design. The researchers checked the comprehension of process components represented in sequence, collaboration, activity, and activity flow[1] diagrams. The results showed that decomposition of process components (which exists in sequence and collaboration diagrams) had a positive effect on both the analysis and design activities, while layout organization had a positive effect only on the design performance. Ambler [1] suggests using collaboration diagrams to show asynchronous messaging between objects, while sequence diagrams are preferable for synchronous logic.

2.2 Solutions for UML Consistency and Integration Problems

Several solutions have been proposed for UML consistency and integration problems. These solutions can be divided into translation and verification approaches.

Translation approaches translate multi-view models into more formal languages of model checkers. The model checker tool is then deployed to analyze the given model for inconsistencies. Bowman et al [3], for example, use LOTOS in order to present a formal framework for checking consistency among various viewpoints in Open Distributed Processing (ODP). They define consistency between specifications X_1, X_2, …, X_n as the existence of a physical implementation which is a realization of all X_1, X_2, …, X_n. Furthermore, they classify consistency classes, such as binary

[1] Activity flow diagrams are similar to activity diagrams, except that the activities are not arranged within swimlanes.

consistency, complete consistency, balanced consistency, and inter language consistency, and express their characteristics using LOTOS. Rasch and Wehrheim [21] use Object-Z in order to give a precise semantics to UML class and Statechart diagrams and to check for consistencies between these views.

Mens et al [14] suggest restricting to description logic in order to specify and detect inconsistencies between UML models. They claim that the use of description logic is especially relevant since it contains five reasoning tasks that can be directly used to achieve subsumption, instance checking, relation checking, concept consistency, and knowledge base consistency.

Große-Rhode [11] suggests a semantic approach for the integration of views. This approach, which is applied to the structural and behavioral views of UML, is based on transition systems, algebraic specifications, and transformation rules.

Engels et al [8] present a general methodology to deal with consistency problems in UML behavioral views. According to this methodology, relevant aspects of the models are mapped to a semantic domain in which precise consistency tests can be formulated.

Baresi and Pezze [2] suggest transforming fragments of UML models into high-level Petri nets that serve as a formal semantic domain. This way, UML behavioral views can be simulated and analyzed.

Verification approaches present testing or validation algorithms which check inconsistencies and contradictions between various views. Chiorean et al [5] use an OCL-based framework in order to ensure consistency among UML views. All the consistency rules are defined at the metamodel level, supporting their reuse for any specific user model.

Bodeveix et al [4] implemented a tool for checking the coherence between the different UML views. This tool is based on an OCL interpreter and a set of OCL expressions over the UML metamodel. Furthermore, OCL is extended to support temporal constraints over the behavioral views of UML.

Engels et al [7] propose dynamic metamodeling (DMM) as a notation for defining consistency conditions. DMM extends the metamodeling idea by introducing metaoperations for the metamodel classes. These operations encapsulate the dynamic semantics of the classes. A DMM-based testing environment, which consists of a test driver, a test controller, and DMM interpreters, was developed.

Based on a classification of consistency constraints that occur in and between specifications at various stages of the lifecycle, Nentwich et al [15] identify a set of requirements that consistency management mechanisms have to address in order to provide proper support. Examples of these requirements are flexibility in constraint application, a tolerant approach to consistency, support for distributed documents, etc. Using a lightweight framework that leverages standard Internet technologies, the researchers address the consistency problems without requiring tight integration, complex translation of specifications, or bulky tools.

The mentioned translation approaches require definitions of translation rules from UML models to semantic, formal languages and definitions of consistency rules in yet other formal languages. This is usually done in two separate supporting tools: translation generators and model checkers, which together with UML-based CASE tools perform the environment in which the translation approaches exist. Moreover, after detecting inconsistencies a backward process should be applied, translating the locations where inconsistencies were found back to the UML models in order to enable the developers to fix the inconsistencies. The verification approaches require in addition sophisticated environments which include test drivers, interpreters, controllers, etc. Moreover, as noted, some of the consistency rules are not explicitly expressed in UML models, demanding semantic interpretation of the UML models and understanding the intentions of their developers.

While both the translation and verification approaches run one time algorithms for checking UML models after their development processes have been completed, I suggest verifying the legibility of the models during the development process. The suggested approach requires defining a top level diagram which glues the different views of a system under development and represents their relationships explicitly. The developers will be aware of existing inconsistencies at any specific time of the development process, thereby being able to correct the models as early as possible. Detecting inconsistencies in early development phases contributes to shortening the system's delivery time ("time-to-market").

3 The Top-Level Object-Oriented Framework Approach

The Top Level Object-Oriented Framework (TLOOF) approach introduces a TLOOF diagram type which is actually an extension of a use case diagram type. In addition to the actors and use cases which exist in regular use case diagrams, a TLOOF diagram includes collaborations and realization relations, both are already part of the UML vocabulary. *Collaborations* provide a way to group chunks of interaction behavior [9]. In other words, collaborations can be viewed as system processes that might have several possible scenarios, each of which can (or should) be described in a different interaction diagram. *Realizations* specify relationships between specification model elements and model elements that implement them. In particular they link use cases to collaborations. Collaborations are symbolized in UML as dashed ellipses, while realizations are denoted by dashed lines ending with triangles. Fig. 1, for example, is a TLOOF diagram of an ordering system. This system requires that a customer will be able to find a product and order it. During the requirement analysis stage, three specification model elements are established: the actor **Customer** and the use cases **Product Finding** and **Product Ordering**. A more detailed specification could be written, dividing **Product Ordering** into searching, reserving, paying, and supplying, but this type of specification is not needed at the requirement level.

While designing the system, the developers find out that they have to implement three main processes: **Product Searching**, **Product Reserving**, and **Product Paying and Supplying**, each of which is modeled in Fig. 1 as a collaboration. From encapsulation and reuse perspectives, **Product Paying And Supplying** includes **Product Searching**, which realizes **Product Finding** as well as **Product Ordering**. All the three collaborations realize the **Product Ordering** use case. The transition between the abstract level of the system, i.e., the use cases, and the more refined description of the same system (the collaborations) is explicitly specified through realization relations. Moreover, this transition is relatively simple since both use cases and collaborations refer to functional elements.

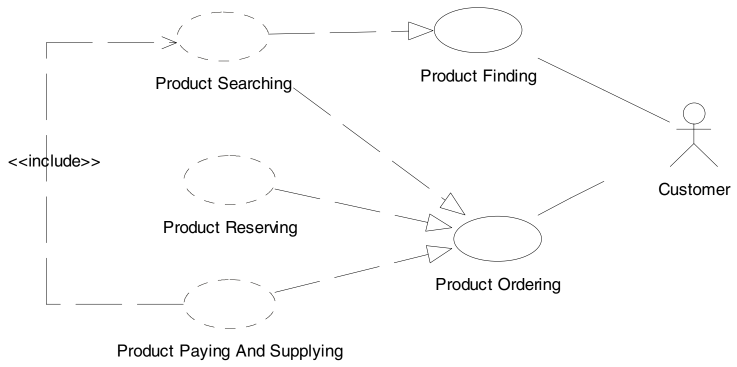


Fig. 1. The TLOOF diagram of an ordering system

Each one of the collaborations that appears in Fig. 1 can be in-zoomed to express its internal structure and behavior. The collaborations are classified into simple and compound collaborations. Compound collaborations are composed of other simple or compound collaborations and, hence, are modeled by composite structure diagrams. A *composite structure diagram* follows activity diagram notations for ordering and activating collaborations. A simple collaboration includes a single (possibly generic) scenario. Simple collaborations are modeled using interaction diagrams, i.e., sequence or collaboration diagrams. A simple or compound collaboration defines which objects can participate in the collaboration, how many objects of the same class can participate in the collaboration, and what their roles are. Using UML stereotypes, there are five possible roles for the associations connecting classes (objects) and collaborations:

1. The <<involved>> stereotype connects a collaboration to a class (or an actor) the objects of which can participate in the collaboration as unchangeable inputs.
2. The <<affected>> stereotype connects a collaboration to a class the objects of which can be affected during the collaboration. These objects exist before and after the collaboration occurrence, but their data or states are changed.
3. The <<created>> stereotype connects a collaboration to a class the objects of which are created during the collaboration.
4. The <<deleted>> stereotype connects a collaboration to a class the objects of which are destroyed during the collaboration.
5. The <<transient>> stereotype connects a collaboration to a class the objects of which are local to the collaboration.

Product Paying and Supplying from Fig. 1, for example, is a compound collaboration which is composed of four simple collaborations, **Product Searching**, **Product Paying**, **Product Supplying**, and **Error Announcing**. First, **Product Searching** is executed, determining if the **Product** was found or not. If the searched **Product** was found, **Product Paying** is executed followed by **Product Supplying**. Otherwise, **Error Announcing** is activated. Either way, the end of **Product Supplying** or **Error Announcing** determines the end of the whole **Product Paying**

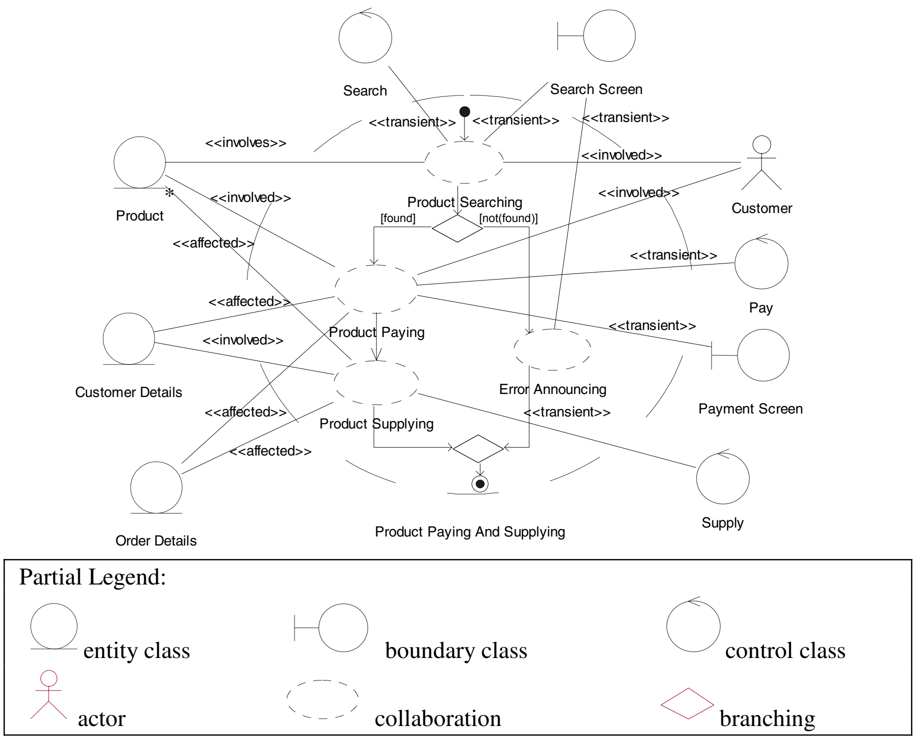


Fig. 2. The composite structure diagram of **Product Paying and Supplying**

And Supplying collaboration. Following the branching and merging notations of UML activity diagrams, Fig. 2 describes the above in a composite structure diagram. In addition, Fig. 2 specifies the object classes needed for the different collaborations. **Product Paying**, for example, uses a boundary object of type **Payment Screen** and a control object of type **Pay** as transient elements. One **Customer** is involved in any **Product Paying** process. During its execution, **Product Paying** also affects an entity object of type **Order Details** and an entity object of type **Customer Details**. Examples for these effects can be changing the order status in **Order Details** and adding the customer's credit card details in **Customer Details**. **Product Paying** can also use any information from the two entity objects, for example the ordered amount (from **Order Details**) which is needed for calculating the final order price. **Product**, on the other hand, is only involved in **Product Paying**, enabling the access to the product price but disabling its change.

Following [12, 18] results, **Product Searching**, which is a simple collaboration, is expressed in Fig. 3 by the sequence diagram[2]. This diagram preserves the "interface" level described by the composite structure diagram in Fig. 2. In other words, in this scenario **Search Screen** and **Search** are transient, while **Customer** and **Product** are only involved (used without being changed).

[2] For simplicity, the operation signatures in the sequence diagram are suppressed, not showing the operation parameters.

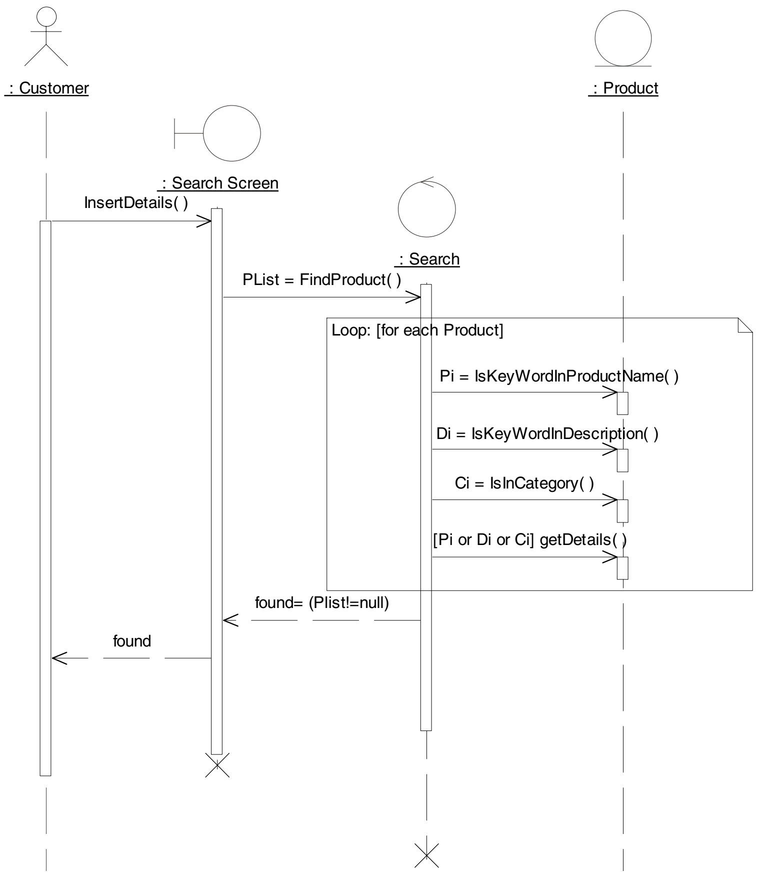


Fig. 3. A sequence diagram describing **Product Searching**

The collaborations, which are described in composite structure diagrams and in interaction diagrams, induce a basic class diagram with its classes (boundary, control, and entity classes), associations, and operations. In the TLOOF approach, the structure of the system is developed to serve the functionality and not the other way, bridging the gap between the requirement analysis phase, which is usually functional- or goal-oriented, and the design phase, which is architectural-oriented. Fig. 4 is the class diagram, or more accurately the robustness diagram [23], induced from the three collaborations of the ordering system applying the following construction rules:

1. All the classes whose objects appear in any interaction diagram appear also in the class diagram.
2. An association between two classes in the class diagram exists if there is a direct message between two objects of these classes in an interaction diagram.
3. All the messages of an interaction diagram are interpreted into operations in the class diagram.

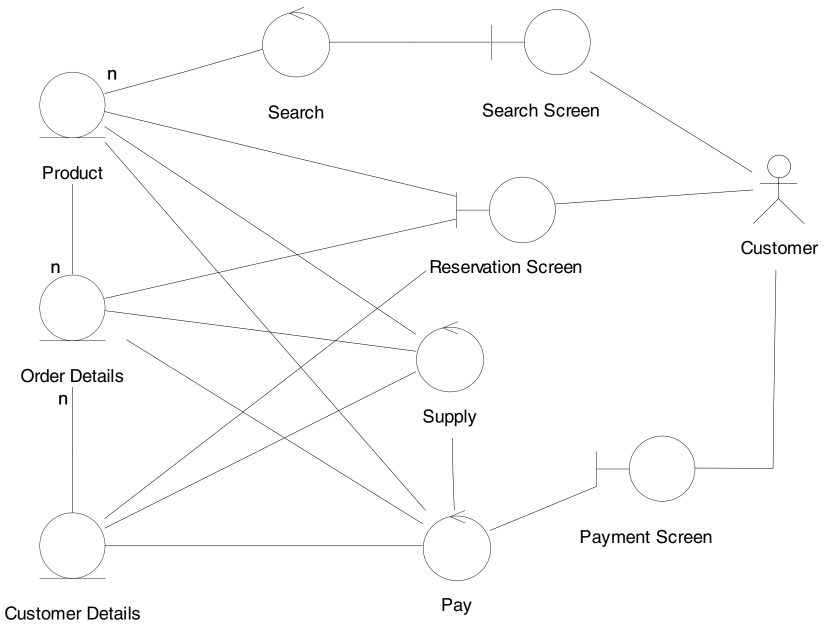


Fig. 4. The induced class (robustness) diagram for the ordering system

These construction rules follow the consistency rules required from class and interaction diagrams in a regular UML model [10]. For clarity purposes, the features (attributes and operations) in Fig. 4 are suppressed. After (automatically) creating the basic class diagram, the developers can improve it by adding attributes, associations, operations, etc.

4 Consistency Rules for the TLOOF Approach

As noted, the TLOOF approach does not increase the vocabulary of UML, while better connecting the different views of the same system. Moreover, the explicit bindings of UML use case, class, and interaction diagrams in the TLOOF approach helps defining consistency rules between these views. This section defines and exemplifies the consistency rules introduced by the TLOOF approach.

4.1 Consistency Rules Within the TLOOF Diagram

As noted, the TLOOF diagram extends the use case diagram and, hence, all the use case diagram rules should be enforced in the TLOOF diagram as well. In particular, each use case in a TLOOF diagram should be connected either directly or indirectly to an actor. Indirect connections exist through inheritance relations, while direct bindings use associations. In addition, the TLOOF approach defines the required realization relations between use cases and collaborations.

The TLOOF realization rule:
Each use case in a TLOOF diagram is realized by at least one collaboration.

Relating to vertical consistency, the TLOOF realization rule enables traceability of the system functional requirements. Each functional requirement, which is expressed by a use case, is realized by at least one collaboration, ensuring that no requirements are lost. Pay attention that the TLOOF realization rule enables a situation in which the implemented system, expressed by the collaborations, will include additional, not requested functionality. In Fig. 1, for example, the **Product Finding** use case (requirement) is realized by one collaboration, **Product Searching**, while **Product Ordering** is realized by three collaborations. In this case, each collaboration realizes a requirement, except of **Product Searching** that realizes two requirements.

4.2 Consistency Rules of Composite Structure Diagrams

A composite structure diagram defines the interface of a collaboration: what are the classes whose objects participate in the collaboration and what are their multiplicities and roles. As noted, the relation between a collaboration and a class can be stereotyped by one of the following: involved, affected, created, deleted, or transient, each of which refers to a different possible effect of the collaboration on the class objects.

The compound collaboration refinement rule:
The declaration (interface) of a compound collaboration includes the declarations of its constituent collaborations. The inclusion order of the collaboration association stereotypes is affected (the most general), created, deleted, involved, and transient (the most particular).

The compound collaboration refinement rule regards to consistency between a composite structure diagram that describes a compound collaboration and the composite structure diagrams that describe its constituents. If we were zooming-out from **Product Paying And Supplying**, shown in Fig. 2, this collaboration would be connected to **Customer** via an involved-stereotyped association, to **Pay**, **Search**, **Supply**, **Payment Screen**, and **Search Screen** via transient-stereotyped associations, and to **Product**, **Order Details**, and **Customer Details** via affected-stereotyped associations.

The actor participation rule:
An actor appears in a composite structure diagram if the actor is connected to a use case which is realized by that collaboration. This connection can be either directly via an association or indirectly by an inheritance relation.

The actor participation rule is derived from the vertical consistency requirement: if an actor is required for a use case, then it will be required for the collaborations that realize (implement) this use case. No contradictions should occur when refining the requirement specification expressed in a use case diagram to a more detailed design specification expressed in composite structure diagrams. In Fig. 2, **Customer** is involved in **Product Searching**, since in Fig. 1 there is an association between **Customer** and **Product Finding**, whose realization is **Product Searching**. Similarly, **Customer** is involved in **Product Paying** due to the fact that **Product Paying** is part of **Product Paying And Supplying** and the latter realizes **Product Ordering**, which is connected to **Customer** in the TLOOF diagram shown in Fig. 1. If **Customer** were not connected directly to **Product Finding** in Fig. 1 but through another use case, say **Product Handling**, from which **Product Finding** inherits, the **Customer** would be still involved in the composite structure diagram of **Product Searching**.

4.3 Consistency Rules Between Composite Structure Diagrams and Interaction Diagrams

Four rules define the consistency required between composite structure diagrams and interaction diagrams. Three of these rules correspond to three of the five stereotypes of composite structure diagrams: created, deleted, and transient[3]. The fourth rule concerns that there will be no additional, redundant objects in the interaction diagrams, i.e., objects whose classes are not declared in the corresponding composite structure diagram.

The created object rule:
Objects of a class which is connected to a collaboration via a created-stereotyped association should be created (without deleting) in at least one related interaction diagram. Furthermore, the number of the created objects in a single interaction diagram should not exceed the corresponding class multiplicity in the collaboration.

The deleted object rule:
Objects of a class which is connected to a collaboration via a deleted-stereotyped association should be deleted (without creating) in at least one related interaction diagram. Furthermore, the number of the deleted objects in a single interaction diagram should not exceed the corresponding class multiplicity in the collaboration.

The transient object rule:
Objects of a class which is connected to a collaboration via a transient-stereotyped association should be created and deleted in at least one related interaction diagram. Furthermore, the number of the transient objects in a single interaction diagram should not exceed the corresponding class multiplicity in the collaboration.

The **Search Screen** and **Search** classes are connected via transient-stereotyped associations to the **Product Searching** collaboration in the composite structure

[3] The two other stereotypes, affected and involved, require naming convention rules and, hence, are not defined as consistency rules.

diagram shown in Fig. 2. In the sequence diagram which describes this collaboration, shown in Fig. 3, one **Search Screen** object and one **Search** object are transient, i.e., created and deleted within the specific scenario. In other words, the effect of the sequence diagram on these objects corresponds (does not violate) the interface declared by the composite structure diagram.

The composite structure integrity rule:
The class of each object that appears in an interaction diagram should appear also in the composite structure diagram of the corresponding collaboration. The class multiplicity in that collaboration is the maximum number of objects that appear in a single interaction diagram of that collaboration.

The composite structure integrity rule ensures that there will be no objects that participate in an interaction diagram, while their classes are not declared in the collaboration interface. Figures 2 and 3 exemplify this rule: the class of each object that appears in Fig. 3 appears also in Fig. 2.

5 Summary and Future Work

The Top-Level Object-Oriented Framework (TLOOF) approach glues commonly used UML views by introducing the TLOOF diagram, which is actually an extension of the use case diagram with realized collaborations. The TLOOF diagram is refined into composite structure diagrams, each of which represents a separate collaboration. A composite structure diagram is refined by other composite structure diagrams in case of compound collaborations or by interaction diagrams in case of simple collaborations. This set of diagrams induces a connected graph with a single root, the TLOOF diagram, whose leaves are interaction diagrams. The connected graph enables smooth transitions from one aspect of the system to another without loosing the legibility and comprehension of the entire system. Seven rules, which can be easily implemented and checked, ensure that the UML models obtained in the TLOOF approach are consistent. This set of rules is complete, since it defines a consistency rule for each element that appears in more than one diagram type. Contrarily to the translation and verification approaches for solving UML consistency and integration problems, the TLOOF approach enforces developing only consistent and integral UML models.

The TLOOF approach makes use of existing notions of UML, such as collaborations, realization relations, and composite structure diagrams. The associations between collaborations and classes are classified using stereotypes, a UML build-in extension mechanism. While not extending the UML vocabulary, the TLOOF approach provides the missing glue for UML views and enables checking model consistency and integrity. Furthermore, the TLOOF approach bridges the gap between the requirement analysis and design stages, enabling requirement traceability. Indeed, checking the comprehension of regular UML models vs. TLOOF models (i.e., UML models that apply the TLOOF approach) on a small group of undergraduate information system students, the TLOOF models were found to be more comprehensive in their description of system behavior and more supportive in requirement traceability.

Further research is planned to deal with overlapping interactions, synchronization points of collaborations, and distribution of collaborations. A series of experiments will verify the comprehension and easiness of developing UML models in the TLOOF approach. A CASE tool for supporting the TLOOF approach is also planned to be developed.

References

1. Ambler, S. W.: How the UML Models Fit Together. Software Development Online: Focus on UML (2000). http://www.sdmagazine.com/documents/s=815/sdm0003z/0003z1.htm
2. Baresi, L., Pezze, M.: On Formalizing UML with High-Level Petri Nets. Concurrent Object-Oriented Programming and Petri Nets (2001) 276-304.
3. Bowman, H., Steen, M., Boiten, E.A., Derrick, J.: A Formal Framework for Viewpoint Consistency. Formal Methods in System Design 21 (2) (2002) 111-166.
4. Bodeveix, J.P., Millan, T., Percebois, C., Le Camus, C., Bazex, P., Feraud, L., Sobek, R.: Extending OCL for Verifying UML Models Consistency. Workshop on Consistency Problems in UML-based Software Development, 5th International Conference on the Unified Modeling Language- the Language and its applications (UML'2002), Dresden, Germany (2002) 75-90.
5. Chiorean, D., Pasca, M., Carcu, A., Botiza, C., Moldovan, S.: Ensuring UML models consistency using the OCL Environment. Workshop on OCL 2.0 - Industry standard or scientific playground?, 6th International Conference on the Unified Modeling Language - the Language and its applications (UML'2003), San Francisco (2003), http://i11www.ira.uka.de/~baar/oclworkshopUml03/papers/06_ensuring_uml_model_consistency.pdf
6. Dori, D.: Object-Process Methodology - A Holistic Systems Paradigm. Springer Verlag, Heidelberg, NY (2002).
7. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Testing the Consistency of Dynamic UML Diagrams. Proc. 6th International Conference on Integrated Design and Process Technology (IDPT 2002), Pasadena CA (2002), http://www.uni-paderborn.de/cs/ag-engels/Papers/2002/EngelsHHS-IDPT02.pdf
8. Engels, G., Kuster, J. M., Groenewegen, L., Heckel, R.: A Methodology for Specifying and Analyzing Consistency of Object-Oriented Behavioral Models. In V. Gruhn (ed.): Proceedings of the 8th European Software Engineering Conference (ESEC) and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9). ACM Press, Vienna Austria (2001) 186-195.
9. Fowler, M., Scott, K.: UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3rd edition, Addison-Wesley (2003).
10. Gomaa, H., Wijesekera, D.: Consistency in Multiple-View UML Models: A Case Study. Workshop on Consistency Problems in UML-based Software Development II, 6th International Conference on the Unified Modeling Language- the Language and its applications (UML'2003), San Francisco (2003) 1-8.
11. Große-Rhode, M.: Integrating Semantics for Object-Oriented System Models. 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete, Greece, Lecture Notes in Computer Science 2076 (2001) 40-60.

12. Hahn, J., Kim, J.: Why Are Some Diagrams Easier to Work With? Effects of Diagrammatic Representation on the Cognitive Integration Process of Systems Analysis and Design. ACM Transactions on Computer-Human Interaction, 6 (3) (1999) 181-213.
13. Henderson-Sellers, B. OO Diagram Connectivity. Journal of Object-Oriented Programming, 11 (7) (1998) 60-68.
14. Mens, T., Van Der Straeten, R., Simmonds, J.: Maintaining Consistency between UML Models Using Description Logic. Workshop on Consistency Problems in UML-based Software Development II, 6th International Conference on the Unified Modeling Language- the Language and its applications (UML'2003), San Francisco (2003) 71-77.
15. Nentwich, C., Emmerich, W., Finkelstein, A., Ellmer, E.: Flexible consistency checking. ACM Transactions on Software Engineering and Methodologies 12 (1) (2003) 28-63.
16. Object Management Group. Unified Modeling Language Specification – version 1.4. ftp://ftp.omg.org/pub/docs/formal/01-09-67.pdf
17. Object Management Group. UML 2.0 Superstructure FTF convenience document. http://www.omg.org/docs/ptc/04-10-02.zip
18. Otero, M.C., Dolado, J.J.: An Initial Experimental Assessment of the Dynamic Modeling in UML. Empirical Software Engineering 7 (2002) 27-47.
19. Paige, R., Ostroff, J.: The Single Model Principle. Journal of Object Technology 1 (5) (2002) 63-81.
20. Peleg, M., Dori, D.: The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. IEEE Transaction on Software Engineering 26 (8) (2000) 742-759.
21. Rasch, H., Wehrheim, H.: Consistency Between UML Classes and Associated State Machines. Workshop on Consistency Problems in UML-based Software Development, 5th International Conference on the Unified Modeling Language- the Language and its applications (UML'2002), Dresden, Germany (2002) 46-60.
22. Reinhartz-Berger, I., Dori, D.: OPM vs. UML – Experimenting Comprehension and Construction of Web Application Models. Empirical Software Engineering (EMSE), 10 (1) (2005) 57-80.
23. Scott, K., Rosenberg, D.: Successful Robustness Analysis. Software Development Online: The lifecycle starts here (2001). http://pyre.third-bit.com/helium/extern/rosenberg04.pdf
24. Tilley, S., Huang, S.: A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding. Proceedings of the 21st annual international conference on Documentation, San Francisco, CA (2003) 184-191.
25. Tun, T., Bielkowicz, P.: A Critical Assessment of UML using an Evaluation Framework. 8th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'03) (2003) 29-37.
26. Uchitel, S., Kramer, J. and Magee, J. Synthesis of Behavioral Models from Scenarios. IEEE Transactions on Software Engineering 29 (2) (2003) 99-115.

How to Manage Uniformly Software Architecture at Different Abstraction Levels

Nassima Sadou, Dalila Tamzalit, and Mourad Oussalah

LINA - CNRS FRE 2729, Faculty of Sciences Nantes University,
2, rue de la Houssiniere BP 92208 44322, Nantes cedex 03, France
{Nassima.sadou, dalila.tamzalit, mourad.oussalah}@univ-nantes.fr

Abstract. We aim to rise software architecture evolution to a higher level of abstraction. We consider the software architecture through three abstraction levels namely, from the most abstract one: the meta level, the architectural one and the application level. According to this, we propose SAEV (Software Architecture EVolution Model). It can describe and manage the evolution at these different levels in a uniform way: as well the evolution of architectures as the evolution of applications. In addition, it can manage the evolution independently of any description or implementation language. For this, software architecture elements (like component, interface, connector and configuration) are considered as first-class entities and SAEV leans on its own concepts and evolution mechanism. SAEV associates to each architectural element its evolution strategy and evolution rules which define its evolution. These rules and strategies must respect all invariants defined on each architectural element to guarantee the coherence of architecture across the evolution.

1 Introduction

Software architecture takes an increasingly importance in the field of software engineering for its re-use, assembly and deployment features. Software architecture offers a high abstraction level for the description of systems, by defining their architectures in terms of components describing the systems functionalities and the connectors which express the interactions among these components [19]. An architecture correctly defined allows the designers to reason about systems properties such as compatibility among components, the performance and reliability at high abstraction level [10,18]

Currently, a number of Architecture Description Languages (ADLs) are defined to aid the components-based systems development, such as C2 [2], ACME [9], Darwin [13]. Most of their efforts focus on the systems specification, development and deployment. Few works are devoted to the evolution of these systems. For the ADLs that approach this problematic of evolution, there proposals are even limited to some techniques such as subtyping, inheritance, composition [15].

We consider the software architecture through three abstraction levels namely, from the most abstract one: the meta level, the architectural one and the application level. we propose SAEV (Software Architecture EVolution Model) to describe and manage the evolution at these different levels in a uniform way: as well the evolution of architectures as the evolution of applications. For this, software architecture elements (like component, interface, connector and configuration) are considered as first-class entities and SAEV leans on its own concepts and evolution mechanism. The evolution of an architectural element is managed through the evolution strategies and evolution rules. The later are based on the ECA rules (Event/Condition/Action) and they describe all the evolutions operations that we can apply on an architectural elements.

The remainder of this paper is organized as follows. Section 2 describes our principal objectives and motivations; section 3 presents the minimal and consensual architectural elements of ADLs ; section 4 describes the proposed evolution model through its concepts and its meta-model as well as its operational mechanism. Section 5 illustrates the evolution model on an example. Section 6 presents the related work, before concluding and presenting our perspectives.

2 Motivations and Main Objectives

Developed systems evolve as well as their architectures. We may need, for instance to add new components, to modify the existing ones or to modify the connections between these components. This evolution must be identified and managed to maintain the architecture coherence of the evolved system. We propose SAEV(Software Architecture EVolution model) as a solution to face the software architecture evolution problem. We are interested more precisely by the structural evolution of architectures. For that, SAEV must:

- Abstract the evolution, from the specific behavior of architectural elements. That allows to:
  - Define a uniform mechanisms for the description and the management of the evolution independently of the architectural elements and their description languages.
  - Support the re-use of these evolution mechanisms in several cases of evolution.
- Be open to the addition of new evolutions, in particular those that are not envisaged initially by the model. It must be for that reflective and adaptive.
- Support static evolution (at the architecture specification time) as well as dynamic one (at the application execution time).

To achieve these objectives, SAEV must take into account all architectural elements proposed by ADLs. We present the main elements in the following section.

3 Main Architectural Elements of ADLs

We present hereafter the main architectural elements commonly supported by the majority of ADLs ([8,9,15,19]). We present first their most accepted definitions in the software architecture community, their Meta model, then we position them according to different abstraction levels.

3.1 Presentation of the Architectural Elements

- Component: represents the computational elements and data stores of a system. It is described by an interface which exports the services that it provides and the services that it requires and one or more implementations. Databases and mathematical functions are examples of component.

- Connector: represents the interaction among components as well as the rules that control this interaction. A connector can describe simple instruction like procedure call or access to a shared variable, but also it can describe a complex interaction like the database access protocols. It is mainly represented by an interface and one or more implementations.

- Interface: the interface is the only visible part of components and connectors. It provides the set of services (provided or required) and interaction points. The interaction points of component are called ports (provided or required port). Those of the connectors are called roles, we distinguish also provided roles from required roles.

-Configuration: represents a connected graph of components and connectors. It describes how they are fastened to each other. The configuration is described also by an interface which provides a set of interaction points (provided ports and required ports) and a set of services. We distinguish also two kinds of links used to fasten configuration's elements (components and connectors).

o *Attachments*: they express which ports of a given component connected to which roles of a connector (a provided port must be attached only to a required role and a required port must be attached only to a provided role).

o *Bindings*: they define links among: a ports of a composed component and those of its subcomponents, a roles of a composite connector and those of its subconnectors or among the ports of a configuration and those of its components.

These elements are represented by the following meta model described using the class diagram of UML[4].

Each architectural element is represented by a class and each element can be connected to other elements by an association of composition. Each element is characterized by an interface and its contents (elements which compose it).

3.2 Architectural Elements Abstraction Levels

Some of the surveyed ADLs such as C2 [2], ACME [9] and Rapide [12] consider components and connectors as first-class entities and distinguish respectively the component-type and the connector-type from their component-instances and connector-instances. This distinction is not valid for other concepts like configuration, interface, etc. For example the configuration is considered only at the moment of the instantiation as a graph of components-instances and connectors-instances. From our point of view, we consider all architectural elements as first-class entities and which can be positioned at three abstraction levels: the Meta level, the Architectural level and the Application one, we illustrate these levels with the figure (Fig2).

1. **Meta Level:** It is the level of definition of all ADLs architectural elements, like *configuration*, *component*, *connector* and *interface.*

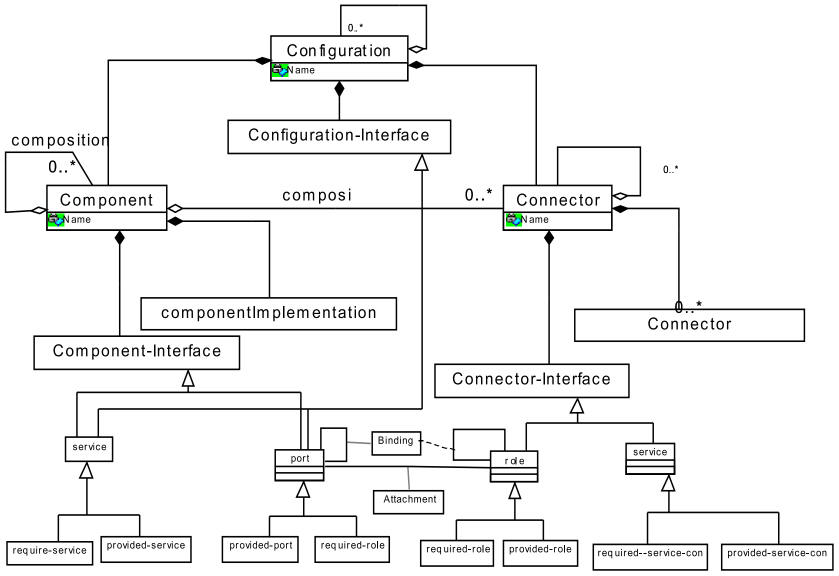


Fig. 1. Architectural Elements Meta model

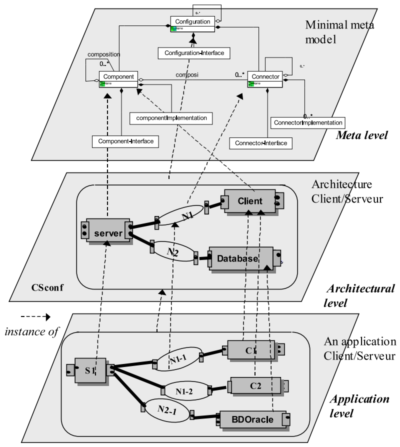


Fig. 2. Architectural Elements Abstraction Level

2. **Architectural Level:** It is the level of the description of any architecture using one or more architectural elements defining in the meta level. The Figure 2, presents a Client/Serveur architecture with a Configuration: *CSConf*; three components : *client*, *server*, *data base* and of two connectors *N1* and *N2*.
3. **Application Level:** It is the level of description of any application in accordance with its architecture. For example, from the preceding architecture client/server, we can build the following application made up of: one instance

of the configuration CSConf: Cf, two instances of the component client: C1, C2 one instance of the component data base: DBoracle, one instance of the component server: S1; two instances of connector N1: N1-1, N1-2 and one instance of connector N2: N2.1.

4 SAEV: Software Architecture EVolution Model

The evolution of software architecture is reflected through the different changes carried out on its elements. These changes can be, for example the addition of a component, the deletion of one of its components, the deletion of a connectors among these components. Each change may cause also impacts that should be managed to maintain the whole architecture in a coherent state. In order to face this requirements and the software architecture evolution needs, the adopted solution must answer at least the following questions:

– What are the elements of an architecture, which can evolve, through its life cycle?
– What are all operations acting on these elements?
– What are the impacts of an operation applied to an element and how to manage this impact?

Basing on these concerns and the previous objectives, SAEV offers a whole of concepts to describe and manage the software architecture evolution.

4.1 SAEV's Meta Model

To describe the evolution of an architecture, we associated with each of its element (architectural element) an evolution strategy. A strategy gathers the whole of evolution rules which describe the operations that can be applied to this architectural element. Thus, each evolution rule must respect the invariants defined on this architectural element.

We detail each concept in the following section, we illustrate each concept only with the meta level elements but the principle remains the same for those of the architectural level.

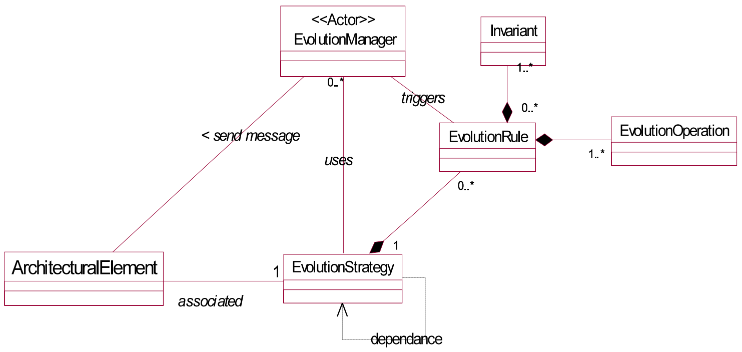


Fig. 3. SAEV's Meta-Model

4.2 SAEV's Concepts

- Architectural Element: It represents any element of a software architecture at each architectural level. In our case, it can be a configuration, a component, a connector, or an interface.

- Invariant: It represents an architectural element constraint which must be respected through its life cycle.Any change in the architecture must maintain the correctness of this invariant. We present hereafter the invariants associated with the main architectural elements: Configuration, component, connector.

Table 1. Architectural Elements Invariants

A.Elements	**Invariants**
Configuration	- a configuration must have an interface, by which it can delegate with other configurations or with its components - a configuration must be composed at least of one component - a connector must connect at least two components - a component can not be related directly to an other component
Component	- a component must be compose at the least of an interface component;
Connector	- a connector must be composed at the least of an interface component;
Interface	- configuration and component interface must be composed at least of one provide port or service

- Evolution Operation: It is an operation that can be applied to a given architectural element and which can cause its evolution. We have identified the following evolution operations: Addition, Deletion, Modification, Substitution. Generally the evolution of software architecture is carried out by the execution of one or more evolution operations on its architectural elements. In the following, we describe examples of evolution operations.

Table 2. Example of Evolution Operations

A.Elements	**Evolution Operations**
Configuration	- Addition, deletion or modification of a provided or required port - Addition, deletion or modification of a provided or required service - Addition or deletion of component or connector - Modification of the name of component or connector
Component	- Addition, deletion or modification of a provided or required port - Addition, deletion or modification of a provided or required service - Addition or deletion of component implementation
Connector	- Addition, deletion or modification of a provided or required role - Addition, deletion or modification of a provided or required service - Addition or deletion of component implementation

Thus it is possible to add, delete and modify the interface and the contents of any architectural element, except the modification of the component or connector implementation which does not raise on the structural evolution.

- Evolution Rule: It describes the execution of an operation on a given architectural element. It expresses the necessary conditions to execute this operation as well as the rules to be triggered if necessary on the other architectural elements, to propagate the rule impacts.

The evolution rules are based on the ECA formalism (Event Condition Action). Thus each evolution rule is made of:

– an event: is the evolution invocation coming from the designer or from another rule. It is intercepted by the evolution manager;
– one or more conditions: that must be satisfied to execute the action part of the evolution rule.
– one or more actions, an action can be an:
  • event, in this case, it will be redirected toward another rule;
  • elementary action to be executed on the architectural element. We note them: Architectural-element-name.Execute.operation-name (parameters);

We present in the following example of evolution rule.

Table 3. Example of Evolution Rules

R1: deletion of Component
Event:delete-comp(Cf: Config,C: comp) **Condition**: $C \in comp(Cf)$, prov-interface(C) connected to prov-interface(Cf) $\exists NC \subset con(Cf)$ and $\forall N \in NC$, N is connected to C and N is not charred **Action**: For $N \in NC$ delete-connector (Cf,N): For $b \in$ bindings(Cf,C) : delete-binding(Cf,C,b) For $I \in$ interface-comp(C) : delete-interface-comp(Cf,C,I) C.Execute-delete-component(Cf)

The rule R1 describes the deletion of the component C belonging to the configuration Cf. This rule triggers firstly the deletion of connectors connected to C, then the deletion of bindings between the component C and configuration Cf, the deletion of the interface of C and finally the deletion of the component C.

The whole of the defined evolution rules is stored in a rules base. The designer will be able to re-use these rules or to create his own evolution rules.

- Evolution strategy: We associate with each architectural element an evolution strategy. An evolution strategy gathers the whole of the evolution rules which describe all the evolution operations (addition, deletion, modification substitution) that can be applied to this architectural element. These rules can be already defined in the evolution rules base as, they can be rules defined by the designer.

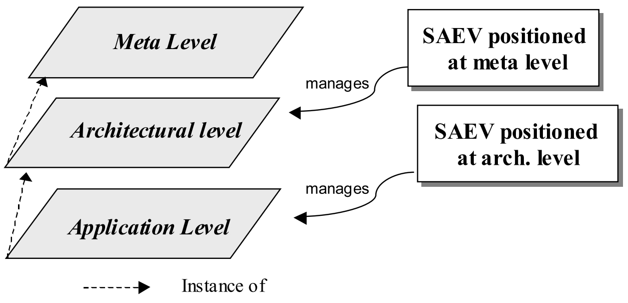


Fig. 4. SAEV and the Abstraction Levels

- Evolution Manager: It is an actor, representing the processing system. Its role is intercepting the events emanating from the designer or the evolution rules towards an architectural element. Then it triggers the execution of the corresponding evolution rules, according to the evolution strategy associated with this architectural element. We detail this process in the following section.

4.3 Evolution Mechanism

The evolution mechanism describes the execution process of the evolution model. It is composed of two steps the interception of the events and the execution of the evolution rules;

Step1: Interception of the event; The evolution manager

– intercepts each event emitted towards the architectural element, either by the designer or by the evolution rules;
– selects the evolution strategy associated with the architectural element on which the event is invoked;
– then selects in this strategy the evolution rules that correspond to the event and that have a satisfied conditions.

Step 2: Execution of the evolution rules The evolution manager triggers the execution of the action part of the selected rule (s). Two cases can arise:

– If the action corresponds to an event, the manager intercepts it also it follows the step1-2.
– If it corresponds to an another action the manager triggers its execution on the architectural element on which the operation is invoked.

5 SAEV and the Abstraction Levels

The proposed evolution model SAEV must be able to describe and manage the evolution of an architecture at the architectural level as well as at the application level. Thus, it can be positioned at the meta level (all its concepts will be defined at the meta level) to manage the evolution of the architectural level and it can

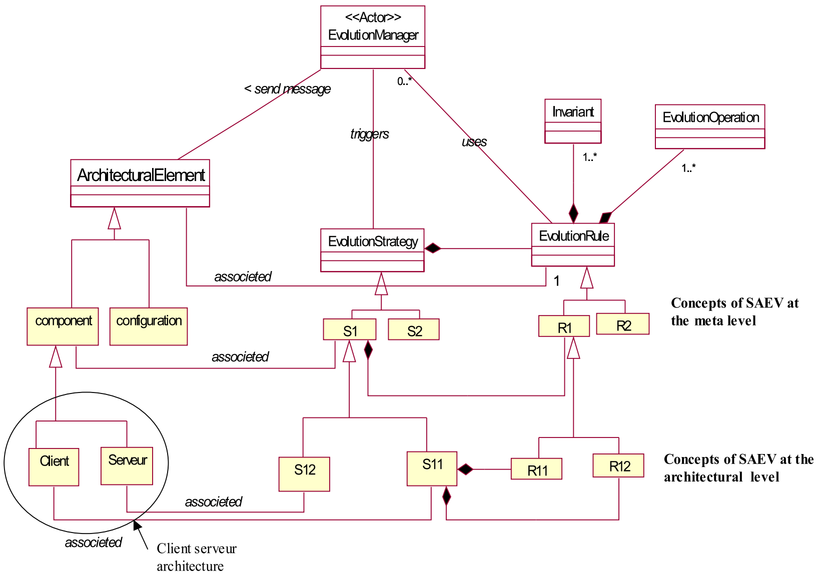


Fig. 5. Description of SAEV at the Different Levels

be positioned at the architectural level (the concepts of SAEV will be defined at the architectural level) to manage the evolution of the application level.

The previous figure illustrates the use of SAEV at the differents abstraction levels. At the Meta level an ArchitecturalElement can represent a: Configuration, Component, Connector or an Interface. We associate to each element its Evolution Operations, Evolution Rules, Evolution Strategy and its invariants. We define them as subclasses of SAEV's classes. In the preceding figure we present an example of strategy:S1 which inherit from the class EvolutionStrategy and which is associated with the architectural element component. S1 is also composed of two rules R1,R2 which inherit from the class EvolutionRule.

At the architectural level an ArchitecturalElement can be any element of the given architecture description. In the previous figure it can be for example the component *Client* or the component *Server*. Also at this level, we associate to each architectural element its Evolution Operations, Evolutions Rules, Evolution Strategies and Invariants).

The concepts of SAEV (Evolutions rules, Invariants, Strategies,) defined at the Meta level are not redefined at the architectural level, and they must be respected at the two levels (architectural and application). For example an evolution rule which describes the deletion of a component *client* at the application level is not redefined at the architectural level, but it extended only the evolution rule which describes the deletion of a Component at the meta level. As the same with all other concepts of SAEV. For these raisons, we have defined the relation of inheritance between the concepts of SAEV defined at the meta level and those defined at the application level.

6 Example

We illustrate the evolution model on example, which describes the evolution of System managing the Organization of a given Company (COS). Its evolution is caused by the reorganization of this company. This example will illustrated the evolution at the architectural level, thus our evolution model SAEV must be positioned at the meta level.

Initially, the configuration of the system COS is made up of the following components: *Registered office (RO)*, *Division(D)*, *Department(DP)*, *Service(S)*; and the connectors among these component *N1)*, *N2*, *N3*, *N4*.

The reorganization of the company causes the creation of new Agencies, instead of divisions,the reorganization of the services distribution. Concretely this evolution will be expressed by the following operations on the configuration COS:

– Deletion of the component Division
– Addition of the component Agency
– The provided services ps1, ps2 offered respectively by the components Register office, and Service will be transferred to the component Agency. The provided service sf3 offered initially by the component Division will be transferred to the component Service.

Evolution steps: We suppose that the preceding evolution rules are memorized in the rule bases defined at the meta level and also the corresponding strategies.

The first event emanating from the designer is the deletion of the component Division (delete-component(COS,Division) emitted toward the configuration COS. The Evolution Manager (E.M):

– intercepts this event and selects the evolution strategy of the *configuration* (S1);
– selects in this strategy the evolution rule which correspond to this event and which have a satisfied conditions, it selects the rule R1 and triggers its execution:

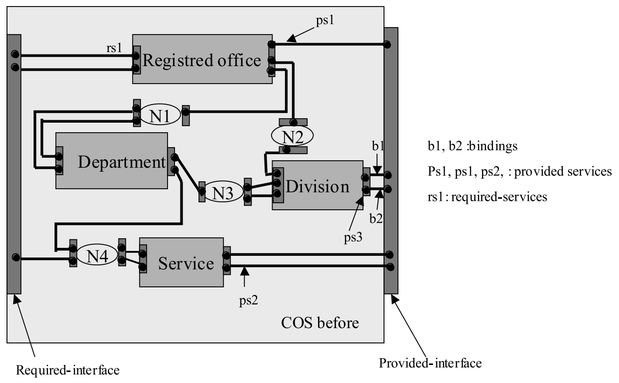


Fig. 6. Configuration COS before Evolution

1. the first action of R1 is an event corresponding to the deletion of connectors (N1, N2) attached to the component Division (delete-connector (COS,N1) and delete-connector(COS,N2);
   (a) The E.M intercepts the first event(delete- connector(COS,N1) and selects the corresponding evolution strategy (the strategy S1);
   (b) selects in this strategy the corresponding rule, the rule: R2

Table 4. Examples of Evolution Rules

R1: Deletion of component	**R2** : deletion of connector

Event:delete-comp(Cf: Config,C: comp)
Condition: $C \in comp(Cf)$, prov-interface(C) connected to prov-interface(Cf)
$NC \subset con(Cf)$ and $\forall N \in NC$, N is connected to C and N is not charred
Action:
For $N \in NC$ delete-connector (Cf,N):
For $b \in bindings(Cf, C)$
delete-binding(Cf,C,b
For $I \in$ interface-comp(C):
delete-interface-comp(Cf,C,I)
C.Execute-delete-component(Cf)
	Event:delete-connect(Cf:Config, N: con)
	Condition: $N \in con(Cf)$ $\exists$ C and C' $\in$ comp(Cf) attached to N
	Action:
	For $t \in attachments(Cf, C, N)$:
	delete-attachment(Cf,C,N,t)
	For $t' \in attachments$: (Cf,C',N)
	delete-attachment(Cf,C',N,t')
	For $I \in$ interface-con (N) :
	delete-interface-cont(Cf,N,I)
	N.Execute-delete-connector(Cf)

R3 : deletion of attachment	**R4** : deletion of interface-connector

Event: delete-attach (Cf:conf,C:comp, N:con,t:att)
Condition: $t \in attach(Cf, C, N)$
Action:
t.Execute-delete-attachment(Cf, C,N)
	Event:delete-inter-con(Cf: Conf, N: con, I: inter-con)
	Condition: $\exists R \subset I$ or $S \subset I$
	Actions: For $r \in R$:
	r.Execute-delete-role(Cf,N)
	For $s \in S$:
	s.Execute-delete-service(Cf,N)

R5 : delete bindings	**R6**: delete interface component
Event : delete-binding(Cf : Conf, C : comp, b :bind) Condition : $b \in$ bindings(Cf,C) Action : b.Execute-delete-binding(Cf, C)	Event : delete-inter-comp(Cf: Conf, C: comp, I : inter-comp) Condition: P: ports $\subset$ I , S:Service $\subset$ I Actions: For p $\in P$ p.Execute-delete-port(Cf,C) For s $\in$ S : s.Execute-delete-service(Cf,C)

Table 5. Examples of Evolution Strategies

Strategies	Operations	Evolution Rules
S1 configuration	Deletion	R1, R2
S2 component	Deletion	R6
S3 connector	Deletion	R4
S4 attachment	Deletion	R3

i. the first action of R2 is delete-attachment (COS,N1,t) using the strategy S4 the E.M triggers the rule R4 which delete all attachments in which the connector N1 takes part;
ii. the second action: delete-interface-connector (COS,N,I) using the strategy S3 the E.M triggers the Rule R6 which delete the interface of the connector N1;
iii. then the last action of R2 "execute-delete-connector(COS,N1) causes the deletion of the connector N1.
with the same steps (1-3) the connector N2 will be deleted also.

2. When all actions of R2 are sequentially executed, the manager executes with the same way the other actions of R1 : The last action of R1 (C.Execute-delete-component(Cf))causes the deletion of the component Division.

Table 6. Comparison of some Architecture Evolution Supports

Crit	**ACME**	**C2**	**SAEV**
CR1	The evolution is described with the architecture ADL. Thus it is integrated in the architecture description	The same with C2	The evolution is defined independently of the architecture description. Thus all its concepts (invariants,rules and strategies)can be reused
CR2	components and connectors are defined with two abstraction levels. (instance and type) Configuration called system is distinguished from the family which describes the whole of system's types	components and connectors are defined with two abstraction levels(type,instance). configuration is defined only at the instance (application)level	it distinguishes three traction levels: meta level,architectural level and application level
CR3	Define the evolution of the components and the connectors types.	define the evolution of the components and connectors at the type and instance level	SAEV can be applied to the three abstraction levels.
CR4	Structural subtyping to evolve the components and connectors	heterogenous subtyping to evolve components and connectors. Offers the operations of addition, deletion upgrading and reconnection of components	Propose a whole and uniform operations to each architectural elements(addition, deletion, modification) as evolution rules which describe these rules
CR5	Allows the static evolution Dynamic ACME is defined to support dynamic evolution	allows static and dynamic evolution	allows static and dynamic evolution

7 Related Works

Few ADLs treat the architecture evolution problem. For those who approach this, the support, they propose are limited to certain techniques such as the inheritance, subtyping, composition, refinement [4] like C2[2] and ACME[8]. C2 is the ADL which most exploited the precedents techniques. Indeed, it proposes several mechanisms of subtyping to evolve the components: interface subtyping, behavior subtyping, implementation subtyping. Several of these mechanisms can be combined together to define a subtype of a component (heterogeneous subtyping). For the connector the language does not propose the same techniques. The C2 connectors don't propose a specific interface and support the communication of a number not predetermined of components.

ACME proposes the same evolution mechanisms for the component and the connector (the connectors of ACME are regarded as first class entities, which structurally do not differ from the components) based on structural subtyping using the clause extend.

To position and compare our evolution model to the evolution supports proposed by others ADLs (C2 and ACME), we have identified 5 criteria, which are:

- CR1:the degree of the abstraction of the evolution.
- CR2: the abstraction level considered for each architectural element.
- CR3: the consideration of the evolution at each abstraction level
- CR4:the evolution operations proposed to evolve each architectural element.
- CR5:the type of evolution considered, static or dynamic.

This comparison is illustrated by the preceding Table (Table 6).

8 Conclusion

We proposed in this article a model for the software architecture evolution, independently of their description languages. We consider the software architecture through three abstraction levels: the meta level, the architectural one and the application level. According to this, we propose SAEV (Software Architecture EVolution Model). It allows the description and the management of the evolution at these different levels in a uniform way: it can positioned at the meta level to manage the evolution of the architectural level, and it can positioned at the architectural level to describe the evolution of the application level. SAEV is based on the common architectural elements for the majority of ADLs: configurations, components, connectors and interfaces. To each element, it associates an evolution strategy. An evolution strategy gathers the whole of evolution rules which describe all the operations that we can apply to this element. The evolution rules must respect the architectural elements invariants to maintain the architecture in a coherent state.

The evolution model SAEV answers the number of the objectives that we have fixed initially (section 2), since the evolution is represented independently of the architectural elements and the same and uniform evolution mechanisms (evolution operations, evolution rules and evolution strategies) are defined to evolve any architectural element.

Until here, we have tested SAEV on examples of architectures specifications that we call static evolution, the implementation of SAEV will then allow us to test and validate our model on cases of dynamic evolution.

References

1. R. Allen : A formal Approach to software Architecture Description, PHD thesis, Carnegie Mellon University Center, Dr. Minneapolis, MN, April 1996.
2. R. Allen, R. Douence, D.Garln : specifying and analyzing Dynamic Software Architectures, In pro-ceeding of the Conference on Fundamental Approaches to Software Engineering, Lisbon, Portugal Mars 1994.
3. L. F. Andrade, J.L. Fiadeiro : architecture based evolution of software systems, LNCS 2804 : 148-181, 2003.
4. G.Booch, J. Rumbaugh, and I. Jacobson: The Unified Modeling Language User Guide, Addison-Wesley Professional, Reading, Massachusetts, 1998.
5. C.Crnkovic, M.Larsson : Challenges of component-based development, in the journal of Systems and Software 201-212, 2002.
6. F. Duclos, J.Estublier and R. Sanlaville : Opened Architecture to software adaptation, software en-gineering, review N 58, September 2001.
7. http://www.eclipse.org/uml2
8. W.B. Frakes, A case study of a reusable component collection in the information retrieval domain, The Journal of Systems and Software, pp 265-270, 2004;
9. D. Garlan, R. Monroe, D. Wile: ACME: Architectural Description Of Components-based systems, Leavens Gary and Sitaraman Murali, Foundations of component-Based Systems, Cambridge Uni-versity Press, 2002,pp. 47-68.
10. D. Garlan, D. Perry: introduction to the special issue on software architecture, IEEE Transactions on Software Engineering, 21(4), April 1995.
11. R. Land: An architectural approach to software evolution and integration, Licentiate thesis, ISBN 91-88834-09-3, Department of Computer Engineering, Mlardalen University, September 19th 2003.
12. D. Luckham, M. Augustin, J. Kenny, J. Vera, D. Bryan, W. Mann: Specification and analysis of System Architectures using Rapide", IEE Transaction on Software Engineering, vol.21, N 4, April 1995,.336-355.
13. J. Magee, N Dulay, S. Eisenbach, J. Kramer : Specifying Distributed Software Architectures, In Proceedings of the fifth European Software Engineering conference, Barcelona, Spain, September 1995.
14. N. Medvidovic, D.S.Rosenblum, and R.N. Taylor: A Language and Environment for Architecture-based Software Development and evolution. In proceeding of the 21st international conference en Software engineering , pp.44-53, may 1999.
15. N.Medvidovic, R. N. Taylor : A Classification and Comparison Framework for Software Architec-ture Description Languages, IEEE Transactions on Software Engineering, Vol. 26, 2000.
16. M.Oussalah, Changes and Versioning in complex objects, International Workshop on Principles of Software Evolution, IWPSE 2001, , Sep. 10 - 11, Vienna University of Technology, Austria.

17. D.E. Perry, A.L. Wolf : Foundations for study of software Architecture, In ACM/SIGSOFT Soft-ware Engineering Notes, volume 17, pages 40-52, October 1992.
18. R. Roshandel, A.V.D. Hoek, M. Miki-Rakic, N. Medvidovic : Mae-A System Model and Environ-ment for Managing Architectural Evolution : ACM Transactions on Software Engineering and Methodology, April 2004.
19. A.Smeda, M. Oussalah, T. Khamaci : A Multi-Paradigm Approach to Describe Complex Software System", WSEAS Transactions on Computers, Issue 4, Volume, 3, October 2003, pp. 936-941.

Schema Integration Based on Uncertain Semantic Mappings

Matteo Magnani[1], Nikos Rizopoulos[2], Peter McBrien[2], and Danilo Montesi[3]

[1] Department of Computer Science, University of Bologna,
Via Mura A.Zamboni 7, 40127 Bologna, Italy
matteo.magnani@cs.unibo.it

[2] Department of Computing, Imperial College London,
180 Queen's Gate, South Kensington Campus, London SW7 2AZ, UK
{nr600, pjm}@doc.ic.ac.uk

[3] Department of Mathematics and Informatics, University of Camerino,
Via Madonna delle Carceri 9, I-62032 Camerino (MC), Italy
danilo.montesi@unicam.it

Abstract. Schema integration is the activity of providing a unified representation of multiple data sources. The core problems in schema integration are: *schema matching*, *i.e.* the identification of correspondences, or *mappings*, between schema objects, and *schema merging*, *i.e.* the creation of a unified schema based on the identified mappings. Existing schema matching approaches attempt to identify a single mapping between each pair of objects, for which they are 100% certain of its correctness. However, this is impossible in general, thus a human expert always has to validate or modify it. In this paper, we propose a new schema integration approach where the uncertainty in the identified mappings that is inherent in the schema matching process is explicitly represented, and that uncertainty propagates to the schema merging process, and finally it is depicted in the resulting integrated schema.

1 Introduction

In this paper we present a new method of schema integration based on uncertain semantic mappings. Schema integration is the activity of providing a unified representation of multiple data sources. The core problems in schema integration are: *schema matching* [1], *i.e.* the identification of correspondences, or *mappings*, between schema objects, and *schema merging* [2], *i.e.* the creation of a unified schema based on the identified mappings. In our approach, we focus on semantic schema integration and on semantic mappings between schema objects. Knowledge about semantic mappings is essential to produce an integrated schema [3]. Early [6,7] and more recent work [4,5,8] has shown that if all semantic mappings are known, then schema merging can be performed semi-automatically.

Unfortunately, it can be very difficult to identify semantic mappings with certainty. Manual schema matching is usually time consuming, and it may be

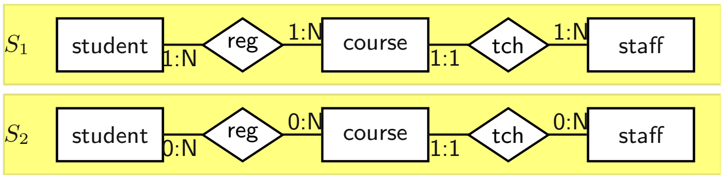


Fig. 1. Schema S_1 and S_2: undergraduate and postgraduate data sources

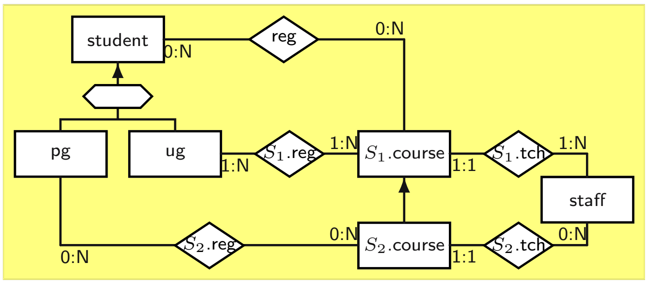


Fig. 2. Schema S_{12}: integration of S_1 and S_2

unfeasible, especially with large databases. Automatic schema matching is inherently uncertain because the semantics of schema objects cannot be fully derived from data and meta-data information. In our novel approach, uncertainty in the identified mappings is represented during the schema matching process, that uncertainty propagates to the schema merging process, and it is depicted in the resulting integrated schema.

As a motivating example, consider the schemas S_1 and S_2 in Figure 1. Schema S_1 models a data source of undergraduate students. Undergraduates are registered (reg) in courses that are taught (tch) by staff members. Schema S_2 models a data source of postgraduate students, which can also optionally register in fourth-year undergraduate courses to refresh their knowledge or familiarize themselves with new subjects. Therefore, S_1.student and S_2.student are disjoint, while S_1.course subsumes S_2.course. Additionally, S_1.staff and S_2.staff are equivalent. The cardinalities of the tch relationship in the two schemas differ, since not all staff members teach fourth-year courses. The aforementioned semantic mappings drive the schema merging process. For example, the disjointness mapping between the student entities triggers schema transformations that rename the entities to make them distinct, *e.g.* into ug and pg, and add a union entity, *e.g.* student, that represents the union set of both undergraduate and postgraduate students. This is illustrated in Figure 2, where the complete integrated schema S_{12} is presented.

However, in general it is impossible to identify fully automatically the correct semantic mappings. Even in the small example above, where the schemas are almost identical, the semantics of the schema objects show subtle differences which make the discovery of the actual semantic mappings very difficult. Most existing techniques [9,10,11] try to identify a single mapping for each pair of objects, which of course could be wrong. For example, an automatic schema matching technique might produce an **equivalence** mapping between the two student entities in S_1 and S_2, based on name comparison.

In this paper, we extend the concept of semantic mapping to include the notion of uncertainty, thus enabling schema matching techniques to show their level of belief on the mappings that they produce. Our goal is the management of this uncertainty. We do not include the implementation details of discovering uncertain mappings, nor the merging technique used to produce the integrated schema, even though we give such examples to illustrate our approach. To gain an intuition of our methodology, assume to have a finite amount of belief that can be distributed to the alternative semantic mappings of two schema objects. When we are certain about a mapping we assign all our belief to it. This is implicitly done by the existing schema matching techniques [1]. A straightforward extension of this concept can be obtained by allowing several alternative mappings to be possible, and distributing our belief to them. For example we might think that the two student entities are either disjoint (if we believe that one entity is undergraduates and the other postgraduates), or equivalent (if both entities represent all the students). This legitimate uncertainty should not prevent the integration of schemas S_1 and S_2. In fact we can think of two possible integrations, one based on disjointness, where one would form a generalisation hierarchy under student, as shown in Figure 2, and the other based on equivalence, where there would be just a single student entity in the final schema. Hence the uncertainty in the mapping between the two student entities propagates to the corresponding alternative integrations. The final integrated schema is created by combining all the produced mappings and it is structurally uncertain.

Our approach, which produces a set of possible mappings for each pair of schema objects, subsumes previous work where a single mapping is specified for each pair. As far as we know, there are two other related approaches that are concerned with uncertainty in schema and data integration. In [12] an approach to integrating XML documents is described, based on probability theory, that deals with uncertainty in data-level schemas. However, we focus on schema integration, and probability theory is just a particular case of the formalism used in our approach to manage uncertainty. In [13], uncertainty is only examined on equivalence mappings, while we provide a much wider set of possible semantic mappings, *e.g.* subsumption and disjointness. Moreover, in [13] only mappings between sets of attributes are considered, while we propose a more general methodology for matching and merging whole schemas.

The paper is organized as follows. In the next section we briefly present an existing schema integration method based on semantic mappings [14,8]. In the following sections, we extend it to deal with uncertainty. In particular, in Section 3 we introduce the theory used to represent uncertainty, and provide the formal definition of *uncertain semantic relationship* (USR), together with illustrative examples. In the same section, we also present a software architecture that can be used to compare schemas and discover USRs. In Section 4 we analyze dependencies between USRs, and describe the process of building an uncertain integrated schema, *i.e.* a set of possible schemas with a belief distribution over them. The main problem tackled in this section is the management of dependencies between USRs. Finally, we draw our concluding remarks. Schemas S_1 and S_2 in Figure 1 will be used as a working example throughout the paper.

2 Schema Integration Based on Semantic Mappings

In this section, we summarize the schema integration approach presented in [14,8], which we then extend in the sections that follow to deal with uncertainty.

2.1 Semantic Relationships

In [14], a mapping between two schema objects is specified by a semantic relationship. We have defined six types of semantic relationships between schema objects based on a set comparison of their *intentional domains*, *i.e.* the set of real-world objects that they represent [14]. We use $Dom_{int}(E)$ to define the intentional domain of an ER entity E. The intentional domain of a binary ER relationship is a subset of the Cartesian domain of the intentional domains of the entities it associates, *e.g.* in schema S_1, $Dom_{int}(\textsf{reg}) \subseteq Dom_{int}(\textsf{student}) \times Dom_{int}(\textsf{course})$. The semantic relationships are:

1. **equivalence ($\overset{S}{=}$):** Schema object ER_1 is equivalent to ER_2, $ER_1 \overset{S}{=} ER_2$, iff $Dom_{int}(ER_1) = Dom_{int}(ER_2)$
2. **subset-subsumption ($\overset{S}{\subset}$):** Schema object ER_1 is a subset of schema object ER_2, $ER_1 \overset{S}{\subset} ER_2$, iff $Dom_{int}(ER_1) \subset Dom_{int}(ER_2)$
3. **superset-subsumption ($\overset{S}{\supset}$):** Schema object ER_1 is a superset of schema object ER_2, $ER_1 \overset{S}{\supset} ER_2$, iff $Dom_{int}(ER_1) \supset Dom_{int}(ER_2)$
4. **intersection ($\overset{S}{\cap}$):** Two schema objects ER_1 and ER_2 are intersecting, $ER_1 \overset{S}{\cap} ER_2$, iff $\neg(ER_1 \overset{S}{\subset} ER_2), \neg(ER_1 \overset{S}{\supset} ER_2), Dom_{int}(ER_1) \cap Dom_{int}(ER_2) \neq \emptyset, \exists ER_3 : Dom_{int}(ER_1) \cap Dom_{int}(ER_2) = Dom_{int}(ER_3)$
5. **disjointness ($\overset{S}{\not\cap}$):** Two schema objects ER_1 and ER_2 are disjoint, $ER_1 \overset{S}{\not\cap} ER_2$, iff $Dom_{int}(ER_1) \cap Dom_{int}(ER_2) = \emptyset, \exists ER_3 : Dom_{int}(ER_1) \cup Dom_{int}(ER_2) \subseteq Dom_{int}(ER_3)$
6. **incompatibility ($\overset{S}{\not\sim}$):** Two schema objects ER_1 and ER_2 are incompatible, $ER_1 \overset{S}{\not\sim} ER_2$, iff $Dom_{int}(ER_1) \cap Dom_{int}(ER_2) = \emptyset$, $\nexists ER_3 : Dom_{int}(ER_1) \cup Dom_{int}(ER_2) \subseteq Dom_{int}(ER_3)$

It is important to notice that object ER_3 in the definition of intersection and disjointness may or may not exist in the schemas. The notation $\exists ER_3$: *condition* means that there is a real-world concept in the domain of the data sources examined, that can be represented by an existing or non-existing schema object ER_3 that satisfies the *condition*. The notation $\nexists ER_3$: *condition* in the definition of incompatibility means that there is no real-world concept that would be represented by a schema object ER_3 to satisfy the specified *condition*. We term **semantically compatible** any two schema objects related by one of the above semantic relationships, except incompatibility.

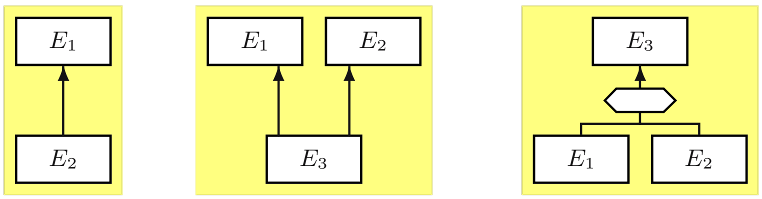
During schema matching, the identification of the above semantic relationships is accomplished by a bidirectional comparison. Our architecture consists of a pool of experts that exploit different types of information to compare schema objects, *e.g.* schema object names, cardinalities, instances, statistical data over the instances, data types, value ranges and lengths. The experts produce similarity degrees which are then aggregated, and with the help of user-defined thresholds the semantic relationships between the schema objects are specified. For example, the comparison of schemas S_1 and S_2 in Figure 1 could produce the following semantic mappings:

$S_1.\mathsf{student} \overset{S}{\not\cap} S_2.\mathsf{student}$	$S_1.\mathsf{course} \overset{S}{\not\sim} S_2.\mathsf{staff}$	$S_1.\mathsf{reg} \overset{S}{\not\cap} S_2.\mathsf{reg}$
$S_1.\mathsf{student} \overset{S}{\not\sim} S_2.\mathsf{course}$	$S_1.\mathsf{staff} \overset{S}{\equiv} S_2.\mathsf{staff}$	$S_1.\mathsf{reg} \overset{S}{\not\sim} S_2.\mathsf{tch}$
$S_1.\mathsf{student} \overset{S}{\not\sim} S_2.\mathsf{staff}$	$S_1.\mathsf{staff} \overset{S}{\not\sim} S_2.\mathsf{student}$	$S_1.\mathsf{tch} \overset{S}{\not\sim} S_2.\mathsf{reg}$
$S_1.\mathsf{course} \overset{S}{\subset} S_2.\mathsf{course}$	$S_1.\mathsf{staff} \overset{S}{\not\sim} S_2.\mathsf{course}$	$S_1.\mathsf{tch} \overset{S}{\not\sim} S_2.\mathsf{tch}$
$S_1.\mathsf{course} \overset{S}{\not\sim} S_2.\mathsf{student}$		

The generation of schema S_{12} in Figure 2 is based on these mappings. However, this ‘definite’ answer misses the fact that we may not be certain that some of the above mappings are correct, and hence alternative integrated schemas exist.

2.2 Schema Merging

In [8], we have defined the merging of schemas based on the semantic mappings specified between their schema objects. Formal rules have been defined that generate both-as-view (BAV) schema transformations [15] and merge two schemas. The application of three such rules on entities E_1 and E_2 is illustrated in Figure 3.



(a) $E_2 \overset{S}{\subset} E_1$ (b) $E_1 \overset{S}{\cap} E_2$ (c) $E_1 \overset{S}{\not\cap} E_2$

Fig. 3. Partial Integrated Schemas: ER Entity Subsumption, Intersection, Disjointness

Figure 3(a) illustrates the *partial integrated schema* that is created when a subsumption relationship is identified between two ER entities, *e.g.* the superset-subsumption relationship identified between entity course in S_1 and course in S_2. We call it a partial integrated schema because it is just a part of the final integrated schema. Figure 3(b) illustrates the partial integrated schema that is created when an intersection relationship is identified between two entities, and Figure 3(c) shows the partial integrated schema created when a disjointness relationship is identified between two entities, *e.g.* the two student entities in S_1 and S_2.

3 Uncertain Semantic Relationships

As already discussed in the introduction, an uncertain semantic mapping is a distribution of beliefs over the set of all possible semantic relationships. To represent beliefs, we have adopted Shafer's belief functions [16]. This choice is justified by the fact that Shafer's belief functions can represent the main kinds of uncertainty present in schema matching (as illustrated in the Examples 1–5 that follow).

The basic concept of Shafer's theory is a function called *basic probability assignment* (BPA), that assigns some probability mass to possible events. The set of all possible elementary events is called *frame of discernment*, and is represented by the letter Θ. In our case, Θ is the set of semantic relationships defined in Section 2, *i.e.* $\{\overset{S}{=}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}$. Possible events correspond to subsets of Θ. For instance, the set $\{\overset{S}{=}, \overset{S}{\cap}\}$ represents the event "The correct semantic relationship is either equivalence or intersection", and $m(\{\overset{S}{=}, \overset{S}{\cap}\})$ is the probability mass supporting exactly this event.

Definition 1 (Basic Probability Assignment (BPA)). *A function* $m : 2^{\Theta} \rightarrow [0,1]$ *is called* basic probability assignment *whenever:*

– $m(\emptyset) = 0$
– $\sum_{A \subseteq \Theta} m(A) = 1$

From a BPA function, we can compute the belief and plausibility of any subset A of Θ.

$$\mathrm{Bel}(A) = \sum_{B \subseteq A} m(B) \tag{1}$$

$$\mathrm{Pl}(A) = \sum_{B \subseteq \Theta, B \cap A \neq \emptyset} m(B) \tag{2}$$

Belief in A is the sum of all probability masses assigned to subsets of A. For example, let A be the set $\{\overset{S}{=}, \overset{S}{\not\sim}\}$. If we assign some probability mass to the set $\{\overset{S}{=}\}$, this increases our belief in all the events containing it. In fact, if we have some evidence supporting the event "The true semantic relationship is equivalence", the same evidence increases also our belief in the event "The true semantic relationship is either equivalence or incompatibility". Plausibility of $A = \{\overset{S}{=}, \overset{S}{\not\sim}\}$ is the sum of all probability masses that are compatible with $\{\overset{S}{=}, \overset{S}{\not\sim}\}$. For example, some probability mass assigned to $\{\overset{S}{=}, \overset{S}{\not\cap}\}$ tells us that A is plausible, without increasing our belief in it, because the right relationship could be disjointness. These definitions can be used to formally define an USR:

Definition 2 (Uncertain Semantic Relationship (USR)). *An* uncertain semantic relationship *between two schema objects A and B is a pair* (Θ, m), *where* $\Theta = \{\overset{S}{=}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}$ *and m is a BPA.*

In the following examples we present the main possible types of USRs, to show that Shafer's theory is expressive enough to represent all USRs that can be found in schema integration.

Example 1 (Certain Relationship). A certain semantic relationship is a special case of USR, where all the probability mass is assigned to a singleton. For example, a BPA $m(\{\overset{S}{=}\}) = 1$ means that we are sure that the true relationship is equivalence.

Example 2 (Probabilistic Relationship). We can use m to assign probabilities to alternative relationships. A BPA $m(\{\overset{S}{\not\cap}\}) = .4, m(\{\overset{S}{\not\sim}\}) = .6$ means that the probability of disjointness is .4, while the probability of incompatibility is .6.

Example 3 (Non-specific Relationship). In many cases, we will only be able to restrict Θ, *i.e.* to exclude some relationships. If we know that two objects are not equivalent, and that the first cannot be a subset of the second, the corresponding BPA will be $m(\{\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}) = 1$.

Example 4 (Partial Ignorance). When we have some information supporting one or more relationships, we should commit part of our belief to them. For instance, a BPA $m(\{\overset{S}{=}\}) = .2, m(\{\Theta\}) = .8$ means that we have some evidence that two objects are equivalent, but we are not sure. Notice that in this case m does not define probabilities. A typical problem with probabilities is the difficulty to justify their precise numerical values. The BPA presented in this example is much more flexible, as it corresponds to a belief $\mathrm{Bel}(\{\overset{S}{=}\}) = .2$ and a plausibility $\mathrm{Pl}(\{\overset{S}{=}\}) = 1$, and thus defines a confidence interval $[.2, 1]$ on the equivalence relationship.

Example 5 (Total Ignorance). As a final example, consider a case in which we have no information about two objects, or we do not want to compare them. This can be very useful to compare parts of schemas, as we show in Section 3.1. We can express our ignorance using the following BPA: $m(\{\Theta\}) = 1$.

3.1 Discovery of USRs

The concept of USR defined above is very intuitive, and is supported by a well known theory at the same time. In this section we present an architecture to discover USRs, and provide an example of schema matching between two entities of S_1 and S_2.

As in the method described in Section 2, the comparison of schema objects is performed by a pool of experts, each one specialized on some features. However, to support the inherent uncertainty of schema matching, experts produce USRs. The mapping between any two schema objects is computed by aggregating the results of all the available experts. Our architecture is illustrated in Figure 4.

The aggregation of USRs is easily achieved by using Dempster's combination rule, that takes two BPAs over the same frame of discernment Θ as input [16]. Using this rule, the combination of experts' beliefs is both based on a sound theory and easy to implement. For every subset A of Θ, the combination of two beliefs (defined by BPAs m_1 and m_2) is defined as:

$$m(A) = \begin{cases} 0 & \text{if } A = \emptyset \\ \frac{\sum_{A_1 \subseteq \Theta, A_2 \subseteq \Theta, A_1 \cap A_2 = A} m_1(A_1) m_2(A_2)}{1 - \sum_{A_1 \subseteq \Theta, A_2 \subseteq \Theta, A_1 \cap A_2 = \emptyset} m_1(A_1) m_2(A_2)} & \text{if } A \neq \emptyset \end{cases} \tag{3}$$

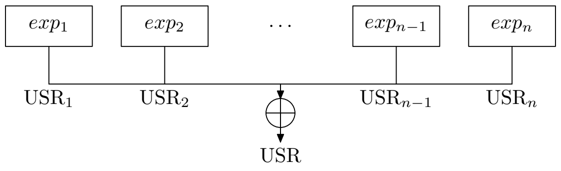


Fig. 4. Architecture proposed to discover USRs

This rule can be used to combine the USRs produced by two experts. The combination of the beliefs of n experts is obtained by iteratively applying it $n-1$ times.

After the application of the rule, it may happen that some semantic relationships are supported by a very small amount of probability mass. In this case, we can decide to dispose of them, and to keep only the relationships supported by a significant amount of probability mass. Thresholds can be used for this purpose. This is useful to improve efficiency, as it reduces the cardinality of the event space, and it allows us not to consider possible semantic relationships that are very unlikely to be the correct ones. However, in this paper we do not investigate how to choose thresholds, as we focus on the theoretical aspects of our method. In general, they can be found experimentally, or set up by the users.

Our architecture has many desirable features: (a) its implementation can be focused on experts, that can be very small independent software agents, (b) it is scalable, as experts can be deleted and added to the pool with no complexity, (c) it is easily parallelizable, as experts can run on different and dedicated hardware, and (d) experts can be software modules, equipped with data analysis tools, or they can be humans, using software interfaces.

The only requirement on experts is to output USRs. Dempster's rule can be used as long as they do not contradict each other. Therefore, human experts can cooperate with software agents to improve the quality of integration of large schemas, thanks to Dempster's combination rule. If a human expert knows or identifies with certainty some relationships, the beliefs of other experts will not be considered, as far as they do not state explicitly that the human USR is wrong. At the same time, we can expect human experts to give their contribution on some parts of the schemas, letting software agents compare the remaining schema objects. This can be done by expressing total ignorance about the objects we do not want to compare. Total ignorance does not influence the combination of beliefs of other experts.

3.2 Examples

To clarify how USRs can be discovered, we present an example involving three experts. However, the definition of experts lies outside the scope of this paper, and we introduce them only to show how our architecture works. This example focuses on the comparison of the two student entities in schemas S_1 and S_2.

The first expert compares the cardinality of two schema objects, *i.e.* the number of instances belonging to them. If cardinalities are equal, subsumption is not possible. If the cardinality of the first object is greater than the other, they cannot be equivalent and the second object cannot subsume the first one. Notice that this expert assumes that all instances belonging to those objects in the real world are stored in the database. It would be easy to improve the expert so that some instances can be missing, using fuzzy comparisons. However, this is not needed in this example. The cardinality of S_1.student is much greater than that of S_2.student, because there are much more undergraduate students than postgraduates. Therefore, the first expert can exclude equivalence and subset-subsumption. The USR produced by this expert is defined by $m_1(\{\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}) = 1$.

The second expert compares object names, using an ontology. The ontology stores information about the six relationships under consideration, when comparing English words. For example, a subsumption relationship between the terms *undergraduate* and *student* corresponds to some confidence on the fact that a schema object whose name is undergraduate is a subset of a schema object called student. Moreover, the expert would also have some (less) confidence about the equality of the two corresponding objects. As it only compares the names of the entities, the ontology-based expert will always assume to be possibly wrong. The second expert, based on the identical names of student entities, might produce the following BPA: $m_2(\{\overset{S}{=}\}) = .7, m_2(\{\overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\cap}\}) = .2, m_2(\{\Theta\}) = .1$.

The third expert compares the instances of two schema objects. For efficiency reasons, it only compares a subset of the instances of S_1.student with all the instances of the S_2.student entity, and *vice versa*. Obviously, the expert cannot compare directly real-world objects, but must compare name, type, and values of the entity identifiers in the ER schemas. This induces uncertainty on the result. In our example, the third expert cannot find matches between the instances of the two student entities, because an undergraduate cannot be a postgraduate and *vice versa*. Therefore, it will support the set of relationships $\{\overset{S}{\cap}, \overset{S}{\not\cap}\}$. However, as already said, the expert cannot be certain of this information. Its USR is defined by: $m_3(\{\overset{S}{\cap}, \overset{S}{\not\cap}\}) = .8, m_3(\{\Theta\}) = .2$.

The combination of m_1, m_2, and m_3 is obtained by applying Dempster's rule, and produces the following USR:

$$m(\{\overset{S}{\cap}, \overset{S}{\not\cap}\}) = 4/5,\ m(\{\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}\}) = 2/15,\ m(\{\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}\}) = 1/15.$$

Table 1. Belief, plausibility of alternative semantic relationships between students

Relationship	Bel	Pl
$\{\overset{S}{=}\}$	0	0
$\{\overset{S}{\subset}\}$	0	0
$\{\overset{S}{\supset}\}$	0	$\frac{1}{5}$
$\{\overset{S}{\cap}\}$	0	1
$\{\overset{S}{\not\cap}\}$	0	1
$\{\overset{S}{\not\sim}\}$	0	$\frac{1}{15}$

This result is what we would expect from the combination of the three USRs. The second expert assigns a large amount of probability mass to the equivalence relationship, but this option is excluded by the first expert. For this reason, equivalence is not considered in the final USR. The high probability mass assigned to disjointness and intersection is justified by the fact that m_1, m_2 and m_3 support these two relationships. In fact, all experts think that disjointness and intersection are plausible, and one of them (the third expert) believes in it.

In Table 3.2 we have indicated belief and plausibility of every alternative relationship. Notice that both $\overset{S}{\cap}$ and $\overset{S}{\not\cap}$ are completely plausible, while $\overset{S}{=}$ and $\overset{S}{\subset}$ are not plausible at all. The choice of further considering $\overset{S}{\supset}$ and $\overset{S}{\not\sim}$ in our analysis depends on the threshold we set up. In our working example, we will not consider them as they are not plausible enough, compared to $\overset{S}{\cap}$ and $\overset{S}{\not\cap}$.

4 Uncertain Integrated Schema

This section presents the schema merging process of our methodology. Based on schema matching and the discovered uncertain semantic relationships, several possible integrated schemas can be created. We explain how the beliefs assigned to the uncertain semantic relationships are propagated to these schemas and a final *uncertain integrated schema* is produced. First, though, the dependencies between the uncertain semantic relationships need to be examined and possible conflicts need to be identified.

4.1 Dependencies Between Semantic Relationships

Consider again the two schemas S_1 and S_2. Similarly to Section 3.2, the uncertain semantic relationships between the two reg ER relationships can be computed. These two ER relationships have identical names but they do not have any instances in common and particularly S_1.reg represents a much larger set of instances. Thus, the three experts described in Section 3.2 will produce the same USRs as the ones produced for students. These are aggregated and the highest probability mass is assigned to disjointness and intersection, $m(\{\overset{S}{\cap}, \overset{S}{\not\cap}\}) = \frac{4}{5}$. Because the rest of the alternatives have very small probability masses we can safely assume that $m(\{\overset{S}{\cap}, \overset{S}{\not\cap}\}) = 1$. The same assumption will also produce $m(\{\overset{S}{\cap}, \overset{S}{\not\cap}\}) = 1$ for the student entities. Finally, suppose that a human expert has specified that the semantic relationship between the course entities is superset-subsumption, $m(\{\overset{S}{\supset}\}) = 1$, *i.e.* S_1.course $\overset{S}{\supset}$ S_2.course.

During schema merging, these produced USRs need to be combined. Table 2 illustrates all their possible combinations. Consider the second row of the table, where S_1.course $\overset{S}{\supset}$ S_2.course, S_1.reg $\overset{S}{\cap}$ S_2.reg and S_1.student $\overset{S}{\not\cap}$ S_2.student. The intersection relationship between the two reg ER relationships specifies that there is at least one common instance between S_1.reg and S_2.reg, *i.e.* there is a common instance of S_1.student and S_2.student that is associated with a common instance of S_1.course and S_2.course. But, according to the second row of the table, the student entities are disjoint and do not have any instances in common. Therefore, the combination of semantic relationships in the second row of the table is invalid.

Table 2. Possible combinations of alternative semantic relationships between course, reg, and student schema objects

S_1.course,S_2.course	S_1.reg,S_2.reg	S_1.student,S_2.student
$\overset{S}{\supset}$	$\overset{S}{\cap}$	$\overset{S}{\cap}$
$\overset{S}{\supset}$	$\overset{S}{\cap}$	$\overset{S}{\not\cap}$
$\overset{S}{\supset}$	$\overset{S}{\not\cap}$	$\overset{S}{\cap}$
$\overset{S}{\supset}$	$\overset{S}{\not\cap}$	$\overset{S}{\not\cap}$

This example manifests the existence of *dependencies* between the semantic relationships of ER relationships and the semantic relationships of the associated ER entities, and *vice versa*. In this paper, we focus just on binary ER relationships. We have exhaustively examined their dependencies and we present in Table 3 all the legal combinations.

The table considers the general case of two ER relationships: ER relationship R_1 that associates ER entities A_1 and B_1 and ER relationship R_2 that associates entities A_2 and B_2 (Figure 5). The first column of the table specifies the semantic relationship between the entities A_1 and A_2, and the second column specifies the semantic relationship between B_1 and B_2. The third column examines the possible semantic relationships between R_1 and R_2.

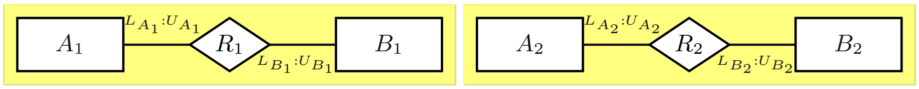


Fig. 5. Two ER relationships: R_1 and R_2

Our previous example, where the intersection relationship between S_1.reg and S_2.reg was invalid, is a case of $A_1 \overset{S}{\subset} A_2$, $B_1 \overset{S}{\not\cap} B_2$ instantiated to S_2.course $\overset{S}{\subset}$ S_1.course, S_2.student $\overset{S}{\not\cap}$ S_1.student. Row nine of Table 3 defines that in this case the legal semantic relationships between R_1 and R_2, instantiated to S_2.reg and S_1.reg, are only incompatibility and disjointness. Thus, the intersection relationship between them is invalid, as previously shown.

In some cases, a semantic relationship between R_1 and R_2 can only be legal when a cardinality condition is satisfied, *e.g.* we can have that $A_1 \overset{S}{=} A_2$, $A_2 \overset{S}{=} B_2$, $R_1 \overset{S}{=} R_2$ if and only if the cardinalities of R_1 and R_2 are identical (first row of Table 3).

Except from dependencies between the semantic relationships of ER relationships and the semantic relationships of their associated ER entities, there are

Table 3. Dependencies between entities and ER relationships

A_1,A_2	B_1,B_2	R_1,R_2
$\overset{\mathrm{S}}{=}$	$\overset{\mathrm{S}}{=}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{=}$: $\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}$, $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{L}_{B_1}{=}\mathrm{L}_{B_2}$, $\mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}$, $\overset{\mathrm{S}}{\subset}$: $\mathrm{L}_{A_1} \leq \mathrm{L}_{A_2}$, $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\neg(\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}, \mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}{=}(1,1))$, $\mathrm{L}_{B_1} \leq \mathrm{L}_{B_2}$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\neg(\mathrm{L}_{B_1}{=}\mathrm{L}_{B_2}, \mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}{=}(1,1))$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{L}_{A_1} \geq \mathrm{L}_{A_2}$, $\mathrm{U}_{A_1} \geq \mathrm{U}_{A_2}$, $\neg(\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}, \mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}{=}(1,1))$, $\mathrm{L}_{B_1} \geq \mathrm{L}_{B_2}$, $\mathrm{U}_{B_1} \geq \mathrm{U}_{B_2}$, $\neg(\mathrm{L}_{B_1}{=}\mathrm{L}_{B_2}, \mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}{=}(1,1))$.
$\overset{\mathrm{S}}{=}$	$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{=}$: $\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}$, $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}$, $\mathrm{L}_{B_1}{=}0$, $\overset{\mathrm{S}}{\subset}$: $\mathrm{L}_{A_1} \leq \mathrm{L}_{A_2}$, $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{L}_{A_1} \geq \mathrm{L}_{A_2}$, $\mathrm{U}_{A_1} \geq \mathrm{U}_{A_2}$, $\mathrm{L}_{B_2}{=}0$, $\mathrm{U}_{B_1} \geq \mathrm{U}_{B_2}$.
$\overset{\mathrm{S}}{=}$	$\overset{\mathrm{S}}{\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{=}$: $\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}$, $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{L}_{B_1}{=}\mathrm{L}_{B_2}{=}0$, $\overset{\mathrm{S}}{\subset}$: $\mathrm{L}_{A_1} \leq \mathrm{L}_{A_2}$, $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\neg(\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}{=}1, \mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}{=}1)$, $\mathrm{L}_{B_1}{=}0$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{L}_{A_1} \geq \mathrm{L}_{A_2}$, $\mathrm{U}_{A_1} \geq \mathrm{U}_{A_2}$, $\neg(\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}{=}1, \mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}{=}1)$, $\mathrm{L}_{B_2}{=}0$, $\mathrm{U}_{B_1} \geq \mathrm{U}_{B_2}$.
$\overset{\mathrm{S}}{=}$	$\overset{\mathrm{S}}{\not\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$: always possible.
$\overset{\mathrm{S}}{=}$	$\overset{\mathrm{S}}{\not\sim}$	$\overset{\mathrm{S}}{\not\sim}$: always possible.
$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{\subset}$: $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{U}_{A_1} \geq \mathrm{U}_{A_2}$, $\mathrm{U}_{B_1} \geq \mathrm{U}_{B_2}$, $\mathrm{L}_{A_2}{=}\mathrm{L}_{B_2}{=}0$, $\overset{\mathrm{S}}{=}$: $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}$, $\mathrm{L}_{A_2}{=}\mathrm{L}_{B_2}{=}0$.
$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\supset}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{\subset}$: $\mathrm{L}_{B_1}{=}0$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{U}_{A_1} \geq \mathrm{U}_{A_2}$, $\mathrm{U}_{B_1} \geq \mathrm{U}_{B_2}$, $\mathrm{L}_{A_2}{=}0$, $\overset{\mathrm{S}}{=}$: $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}$, $\mathrm{L}_{A_2}{=}\mathrm{L}_{B_1}{=}0$.
$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{\subset}$: $\mathrm{L}_{B_1}{=}0$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{U}_{A_1} \geq \mathrm{U}_{A_2}$, $\mathrm{U}_{B_1} \geq \mathrm{U}_{B_2}$, $\mathrm{L}_{A_2}{=}\mathrm{L}_{B_2}{=}0$, $\overset{\mathrm{S}}{=}$: $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}$,$\mathrm{L}_{B_1}{=}\mathrm{L}_{B_2}{=}\mathrm{L}_{A_2}{=}0$.
$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\not\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$: always possible.
$\overset{\mathrm{S}}{\subset}$	$\overset{\mathrm{S}}{\not\sim}$	$\overset{\mathrm{S}}{\not\sim}$: always possible.
$\overset{\mathrm{S}}{\cap}$	$\overset{\mathrm{S}}{\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$, $\overset{\mathrm{S}}{\cap}$: always possible, $\overset{\mathrm{S}}{\subset}$: $\mathrm{L}_{A_1}{=}0$, $\mathrm{U}_{A_1} \leq \mathrm{U}_{A_2}$, $\mathrm{L}_{B_1}{=}0$, $\mathrm{U}_{B_1} \leq \mathrm{U}_{B_2}$, $\overset{\mathrm{S}}{\supset}$: $\mathrm{L}_{A_2}{=}0$, $\mathrm{U}_{A_2} \leq \mathrm{U}_{A_1}$, $\mathrm{L}_{B_2}{=}0$, $\mathrm{U}_{B_2} \leq \mathrm{U}_{B_1}$, $\overset{\mathrm{S}}{=}$: $\mathrm{L}_{A_1}{=}\mathrm{L}_{A_2}{=}0$,$\mathrm{L}_{B_1}{=}\mathrm{L}_{B_2}{=}0$, $\mathrm{U}_{A_1}{=}\mathrm{U}_{A_2}$, $\mathrm{U}_{B_1}{=}\mathrm{U}_{B_2}$.
$\overset{\mathrm{S}}{\cap}$	$\overset{\mathrm{S}}{\not\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$: always possible.
$\overset{\mathrm{S}}{\cap}$	$\overset{\mathrm{S}}{\not\sim}$	$\overset{\mathrm{S}}{\not\sim}$: always possible.
$\overset{\mathrm{S}}{\not\cap}$	$\overset{\mathrm{S}}{\not\cap}$	$\overset{\mathrm{S}}{\not\sim}$, $\overset{\mathrm{S}}{\not\cap}$: always possible.
$\overset{\mathrm{S}}{\not\cap}$	$\overset{\mathrm{S}}{\not\sim}$	$\overset{\mathrm{S}}{\not\sim}$: always possible.
$\overset{\mathrm{S}}{\not\sim}$	$\overset{\mathrm{S}}{\not\sim}$	$\overset{\mathrm{S}}{\not\sim}$: always possible.

Table 4. Dependencies between schema objects of the same kind

A,B	B,C	A,C
$\overset{S}{\equiv}$	$\overset{S}{\subset}$	$\overset{S}{\subset}$
$\overset{S}{\equiv}$	$\overset{S}{\cap}$	$\overset{S}{\cap}$
$\overset{S}{\equiv}$	$\overset{S}{\not\sim}$	$\overset{S}{\not\sim}$
$\overset{S}{\subset}$	$\overset{S}{\supset}$	$\overset{S}{\equiv}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\subset}$	$\overset{S}{\not\cap}$	$\overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\supset}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$
$\overset{S}{\supset}$	$\overset{S}{\not\cap}$	$\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\cap}$	$\overset{S}{\cap}$	$\overset{S}{\equiv}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\cap}$	$\overset{S}{\not\sim}$	$\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\not\cap}$	$\overset{S}{\not\sim}$	$\overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$

A,B	B,C	A,C
$\overset{S}{\equiv}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$
$\overset{S}{\equiv}$	$\overset{S}{\not\cap}$	$\overset{S}{\not\cap}$
$\overset{S}{\subset}$	$\overset{S}{\subset}$	$\overset{S}{\subset}$
$\overset{S}{\subset}$	$\overset{S}{\cap}$	$\overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\subset}$	$\overset{S}{\not\sim}$	$\overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\supset}$	$\overset{S}{\cap}$	$\overset{S}{\supset}, \overset{S}{\cap}$
$\overset{S}{\supset}$	$\overset{S}{\not\sim}$	$\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\cap}$	$\overset{S}{\not\cap}$	$\overset{S}{\cap}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\not\cap}$	$\overset{S}{\not\cap}$	$\overset{S}{\equiv}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$
$\overset{S}{\not\sim}$	$\overset{S}{\not\sim}$	$\overset{S}{\equiv}, \overset{S}{\cap}, \overset{S}{\subset}, \overset{S}{\supset}, \overset{S}{\not\cap}, \overset{S}{\not\sim}$

also dependencies between the semantic relationships of the same type of constructs. Consider the following example. The ER relationship S_1.tch subsumes S_2.tch but we might be uncertain about the semantic relationship between S_1.reg and S_2.tch since both of them associate person identifiers with course identifiers. S_1.reg has a much larger set of instances than S_2.tch, and therefore equivalence and subset-subsumption relationships are excluded. Thus, from a comparison of S_1.reg and S_2.tch a pool of experts could decide to support the set $\{\overset{S}{\supset}, \overset{S}{\cap}, \overset{S}{\not\sim}\}$ of possible semantic relationships. However, since S_1.reg and S_1.tch are incompatible, based on the structure of S_1, and S_1.tch subsumes S_2.tch, the intersection and superset-subsumption relationships between S_1.reg and S_2.tch are also excluded. Therefore, S_1.reg and S_2.tch must be incompatible.

This restriction of relationships is generalised in Table 4, where all legal combinations of semantic relationships between three objects A, B and C of the same type of construct are defined. Objects B and C belong to the same schema thus their semantic relationship can be derived from the schema structure. Semantic relationships between A,B and A,C are discovered during schema matching. In our example of S_1.reg and S_2.tch, A is instantiated to S_2.tch and B,C to S_1.tch and S_1.reg, respectively. If the semantic relationships S_2.tch $\overset{S}{\subset}$ S_1.tch and S_1.tch $\overset{S}{\not\sim}$ S_1.reg are certain, then based on Table 4 S_1.reg and S_2.tch can only be disjoint or incompatible.

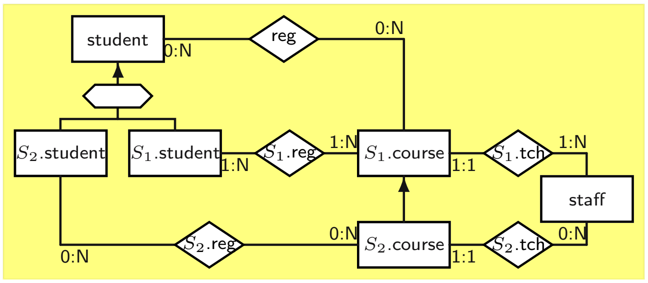
4.2 Schema Merging

In the previous sections we compared student and reg schema objects, obtaining a set of possible semantic relationships between them, with BPAs representing our belief distribution. In particular, both student and reg schema objects could be either disjoint or intersecting. This is shown in Table 2.

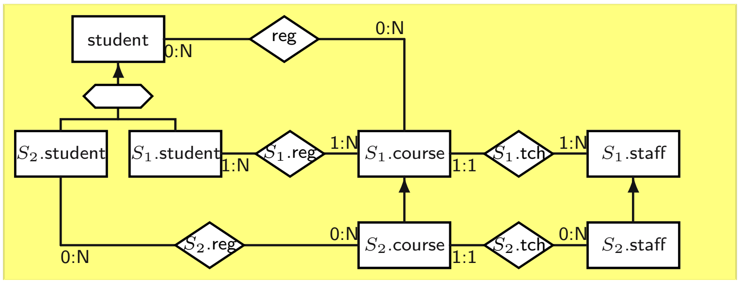
Now assume that S_1.course $\overset{S}{\supset}$ S_2.course and S_1.tch $\overset{S}{\supset}$ S_2.tch relationships are certain, while the relationship between S_1.staff and S_2.staff could be $\overset{S}{\equiv}$, with a probability of .7, or $\overset{S}{\supset}$, with a probability of .3. We can build a complete table (Table 5), that is an extension of Table 2, representing all possible combinations of semantic relationships between all pairs of schema objects. In Table 5 we have concentrated only on compatible objects. Each row of this final table corresponds to a possible integrated schema, where each semantic relationship defines a partial integrated schema, like those represented in Figure 3. For example, in the possible integrated schema (a) of Table 5 staff entities are equivalent, while in the possible integrated schema (b) S_1.staff subsumes S_2.staff. Based on this table we can create the corresponding schemas. The schemas corresponding to rows (a) and (b) of Table 5 are illustrated in Figure 6.

Table 5. Possible combinations of semantic relationships in the integrated schema

#	S_1.stud.,S_2.stud.	S_1.reg,S_2.reg	S_1.course,S_2.course	S_1.staff,S_2.staff	S_1.tch,S_2.tch
(a)	$\overset{S}{\not\cap}$	$\overset{S}{\not\cap}$	$\overset{S}{\supset}$	$\overset{S}{\equiv}$	$\overset{S}{\supset}$
(b)	$\overset{S}{\not\cap}$	$\overset{S}{\not\cap}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$
(c)	$\overset{S}{\cap}$	$\overset{S}{\cap}$	$\overset{S}{\supset}$	$\overset{S}{\equiv}$	$\overset{S}{\supset}$
(d)	$\overset{S}{\cap}$	$\overset{S}{\cap}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$
(e)	$\overset{S}{\cap}$	$\overset{S}{\not\cap}$	$\overset{S}{\supset}$	$\overset{S}{\equiv}$	$\overset{S}{\supset}$
(f)	$\overset{S}{\cap}$	$\overset{S}{\not\cap}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$	$\overset{S}{\supset}$



(a)



(b)

Fig. 6. Two of the final alternative integrated schemas generated by our approach

The BPA obtained as a combination of all the aforementioned USRs is defined by $m\{(a), (c), (e)\} = .7$, and $m\{(b), (d), (f)\} = .3$. The corresponding beliefs and plausibilities can be easily computed using (1) and (2). The meaning of this BPA reflects the uncertainty on the partial integrated schemas. The set $\{(a), (b), (c), (d), (e), (f)\}$, together with its BPA, is called an *uncertain integrated schema*, and is the final product of our schema integration approach on our working example.

From the uncertain integrated schema we can reconstruct all the previously produced USRs. For example, we previously assigned a probability mass of 1 to the set of relationships $\{\overset{S}{=}, \overset{S}{\cap}\}$ between the two student entities. This value can be obtained from the uncertain integrated schema by adding together all the probability masses assigned to combinations of possible integrated schemas where $S_1.\mathsf{student} \overset{S}{\not\cap} S_2.\mathsf{student}$ or $S_1.\mathsf{student} \overset{S}{\cap} S_2.\mathsf{student}$. This corresponds to all the rows of Table 5, *i.e.* all possible schemas. Similarly, if we sum all masses assigned to possible combinations of schemas where $S_1.\mathsf{staff} \overset{S}{=} S_2.\mathsf{staff}$, we obtain .7, while for $S_1.\mathsf{staff} \overset{S}{\supset} S_2.\mathsf{staff}$ we obtain .3.

5 Conclusion and Future Work

In this paper we have presented a new method of schema integration. Differently from other existing methods, our approach manages the inherent uncertainty in (semi-)automatic schema matching, and supports six kinds of semantic relationships between schema objects. These features are essential to cope with real schema integration tasks, where many semantic relationships are possible, and it is very unlikely to know all of them with certainty.

An analysis of the computational complexity of our method is outside the scope of this paper. However, it is easy to identify two main possible causes of inefficiency related to the management of uncertainty. The first is the combination of the USRs produced by the experts. In fact, the complexity of *exact methods* for performing Dempster's combination rule is exponential on the size of the frame of discernment, because it must consider all its subsets in the worst case. However, the frame of discernment in our method contains only six elements – our semantic relationships. Therefore, the complexity of the combination is bounded by a small constant. The second issue is the number of possible integrated schemas generated by the method, that can be exponential on the number of schema objects. However, in practice the output of our method will not be the set of all possible integrated schemas, but only the most probable ones. The number of schemas returned by the method can be decided in advance. Finally, an appropriate use of thresholds can further reduce the number of schemas, without losing significant information.

While the theory underlying our method has been presented in this paper, we still need to experimentally verify its efficiency and effectiveness. In the future, we are going to implement it as an extension of an existing schema integration software [14].

References

1. Rahm, E., Bernstein, P.: A survey of approaches to automatic schema matching. VLDB Journal **10** (2001) 334–350
2. Bernstein, P.: Applying model management to classical meta data problems. In: Proc. CIDR (2003) 209–220
3. Batini, C., Lenzerini, M., Navathe, S.: A comparative analysis of methodologies for database schema integration. ACM Computing Surveys **18** (1986) 323–364
4. Bernstein, P.A., Pottinger, R.A.: Merging models based on given correspondences. In: Proc. 29th VLDB Conference, Berlin (2003)
5. Melnik, S., Rahm, E., Bernstein, P.A.: Rondo: a programming platform for generic model management. In: Proc. SIGMOD, ACM Press (2003) 193–204
6. Hayne S., Ram S.: Multi-User View Integration System (MUVIS): An Expert System for View Integration. In: ICDE (1990) 402–409
7. Spaccapietra, S., Parent, C.: View Integration: A Step Forward in Solving Structural Conflicts. IEEE TKDE, 6(2), (1994) 258–274
8. Rizopoulos, N., McBrien, P.: A general approach to the generation of conceptual model transformations. In: Proc. CAiSE. LNCS, Springer-Verlag (2005)
9. Madhavan, J., Bernstein, P., Rahm, E.: Generic schema matching with Cupid. In: Proc. 27th VLDB Conference. (2001) 49–58
10. Doan, A., Madhavan, J., Domingos, P., Halevy, A.: Learning to map ontologies on the Semantic Web. In: Proc. World-Wide Web Conference. (2002) 662–673
11. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In: ICDE. (2002) 117–128
12. van Keulen, M., de Keijzer, A., Alink, W.: A probabilistic XML approach to data integration. In: ICDE. (2005)
13. Gal, A., Anaby-Tavor, A., Trombetta, A., Montesi, D.: A framework for modeling and evaluating automatic semantic reconciliation. VLDB Journal **14** (2005) 50–67
14. Rizopoulos, N.: Automatic discovery of semantic relationships between schema elements. In: ICEIS (1). (2004) 3–8
15. McBrien, P., Poulovassilis, A.: Data integration by bi-directional schema transformation rules. In: Proc. ICDE, IEEE (2003) 227–238
16. Shafer, G.: A mathematical theory of evidence. Princeton University Press (1976)

Combining Intention-Oriented and State-Based Process Modeling

Pnina Soffer[1] and Colette Rolland[2]

[1] MIS Department, Faculty of Social Science, Haifa University, Carmel Mountain, Haifa 35901, Israel
spnina@is.haifa.ac.il
[2] Universite Paris1 Panthéon Sorbonne CRI, 90 rue de Tolbiac, 75013 Paris, France
Colette.rolland@univ-paris1.fr

Abstract. Business process modeling and design has gained importance in recent years. Consequently, a large number of modeling languages have emerged. Many of them lack formality, whereas some others support the verification of the designed process. Most of existing modeling languages adopt an operational view focusing on how the process is performed. By contrast, others follow the human intention of achieving a goal as the force that drives the process, and concentrate on what the process must do, i.e. on its rationale. The aim of this paper is to combine intention-oriented modeling with formal state-based modeling and achieve their synergy, benefiting from the advantages of both. We use the Map formalism as an example of the former and the Generic Process Model (GPM) as an example of the latter. The paper proposes a procedure for converting a Map into GPM concepts, illustrates it with the SAP Material Management Module and shows the benefits resulting from it.

1 Introduction

Conceptual modeling is aimed at representing the real world for purposes such as understanding, communicating, and reasoning in the process of information systems analysis and design. An important area that emerged in recent years is business process modeling, whose main focus is capturing behavioral aspects of the world, but it also relates to other aspects. Various types of process modeling languages and formalisms have emerged, supporting a variety of purposes. The existing formalisms can be roughly classified according to their orientation to activity-sequence oriented languages (e.g., UML Activity Diagram), agent-oriented languages (e.g., Role-Activity Diagram [7]), state-based languages (e.g., UML statecharts), and intention-oriented languages (e.g., Map [9]). Many of these languages lack formality, and serve as a graphical tool assisting in the creative task of process and IS design. The lack of formality makes the analysis and verification of the designed processes a difficult task.

The concept of goal is central in business process modeling and design. It is included in many definitions of business processes (e.g., "a business process is a set of partially ordered activities aimed at reaching a goal" [6]). However, most process modeling languages do not employ a goal construct as an integral part of the model. This is sometimes justified by viewing these models as an "internal" view of a process, focusing on *how* the process is performed and externalizing *what* the process is intended to accomplish in the goal [5].

In contrast, intention-oriented process modeling focuses on *what* the process is intended to achieve, thus providing the rationale of the process, i.e. *why* the process is performed. Intention-oriented process modeling follows the human intention of achieving a goal as a force that drives the process. As a consequence, goals to be accomplished are explicitly represented in the process model together with the different alternative ways for achieving them, thus facilitating the selection of the appropriate alternative for achieving the goal.

Process goals are also present in some state-based modeling formalisms (e.g., [2]). However, as opposed to intention oriented goals, in state-based modeling a goal stands for a state or a set of states on which the process terminates. State-based modeling captures a process as a flow of states, leading to the goal state. This representation of a process takes a structural view rather than an intentional view, and can be formal enough to provide a basis for analyzing the properties of a process model and its validity [13].

The main difference between the goal concept in intention-oriented modeling and state-based modeling is that in the former, while aiming at representing the human intention, goals are not formally defined and may bear a rather vague meaning. In the latter, in contrast, goals are formally defined, but are not directly related to the human intention.

The aim of this paper is to combine intention-oriented modeling with state-based modeling and achieve their synergy, benefiting from the advantages of both. The intention-oriented modeling notation we use is the Map formalism [9, 11], and the state-based modeling notation is the Generic Process Model (GPM) [12, 13]. The Map notation is intuitive and easy to apply and understand. It is particularly suitable for representing unstructured processes, whose sequence of activities may vary in different situations, or processes including variability (e.g. product lines, ERP or adaptable processes), whose sequence of activities is selected at run time depending on the situation at hand. However, Maps are not formally defined, hence there is no structured procedure for analyzing a Map for deficiencies and invalidity. Furthermore, while the map concepts are intuitively understood, there is no precise definition to their semantics. We suggest a procedure for converting a Map to the state-based concepts of GPM, and use the formality and precision gained in order to achieve a better understanding of the concepts underlying the map. In particular, this understanding provides insights to the essence of acting on an intention. The result is an intention-oriented model which is formally defined and can be analyzed for completeness and validity.

The remainder of the paper is organized as follows: Section 2 provides an overview of both modeling formalisms, Map and GPM; Section 3 interprets Map concepts in GPM terms; Section 4 presents a procedure for transforming Map representation into GPM model, illustrates it by an example, and demonstrates how a model can be analyzed and improved by applying this transformation; conclusions are given in Section 5.

2 The Map and GPM Formalisms

This section provides an overview of both modeling formalisms, the Map and GPM.

2.1 An Overview of Map

In this section we introduce the concept of a map and illustrate it with the *Material Management map (MM map)*, based on the information provided in [1] regarding the SAP R/3 Materials Management module.

The *Map* representation system allows to represent a process model expressed in intentional terms. It provides a representation mechanism based on a non-deterministic ordering of *intentions* and *strategies*.

A *map* is a labeled directed graph (Figure 1) with *intentions* as nodes and *strategies* as edges. An edge enters a node if its strategy can be used to achieve the intention of the node. Since there can be multiple edges entering a node, the map is capable of representing many strategies that can be used for achieving an intention.

An *intention* represents a goal that can be achieved by the performance of a process. For example, the MM map in Figure 1 has *Purchase Material* and *Monitor Stock* as intentions. Furthermore, each map has two special intentions, *Start* and *Stop*, to respectively start and end the process.

A *strategy* is an approach, a manner to achieve an intention. In Figure 1 *Manual strategy* is a manner to manually generate an order to *Purchase Material.*

A *section* is a key element of a map. It is a triplet as for instance <*Start, Purchase Material, Planning Strategy*> which couples a source intention (*Start*) to a target intention (*Purchase Material*) through a strategy (*Planning strategy*) and represents a way to achieve the target intention *Purchase Material* from the source intention *Start* following the *Planning Strategy*. Each section of the map captures a specific manner in which the process associated with this goal can be performed. A section may be recursive when its source and target intentions are the same. In Figure 1, the section <*Purchase Material, Purchase Material, reminder strategy*> is recursive.

Sections of a map are *connected* to one another. This occurs:

(a) When a given goal can be achieved using different strategies. This is represented in the map by several sections between a pair of intentions. The topology corresponding to the case where several strategies can be selected is called a *multi-thread*. In Figure 1 the multi-thread between *Start* and *Purchase Material* represents the two ways in which the *Purchase Material* intention can be achieved (manually and by planning). When the strategies are mutually exclusive, sections are said to constitute a *bundle*, specified by a dotted line and refined in a separate map. In Figure 1, *Planning strateg*y is a bundle composed of two exclusive strategies to achieve the Purchase Material intention, namely *By reorder point planning* and *by forecast based planning*, as shown in Figure 2. Only one of these can be selected each time the *Planning strategy* is taken.

(b) When an intention can be achieved by several combinations of strategies. This is represented in the map by a pair of intentions connected by several sequences of sections. Such a topology is called a *multi-path*. In Figure 1 there are five paths leading from *Start* to *Monitor Stock*.

In general, a map is a multi-path from *Start* to *Stop* and contains bundles and multi-threads. Figure 1 contains several paths from *Start* to *Stop* to handle the "normal cases" and complete the process (i.e. to achieve the *Stop*) through the

Purchase Material and the *Monitor Stock* intentions. This map also allows exceptional cases as, for instance, with the path that directly allows to *Monitor Stock* following the *By Bill for Expenses* strategy.

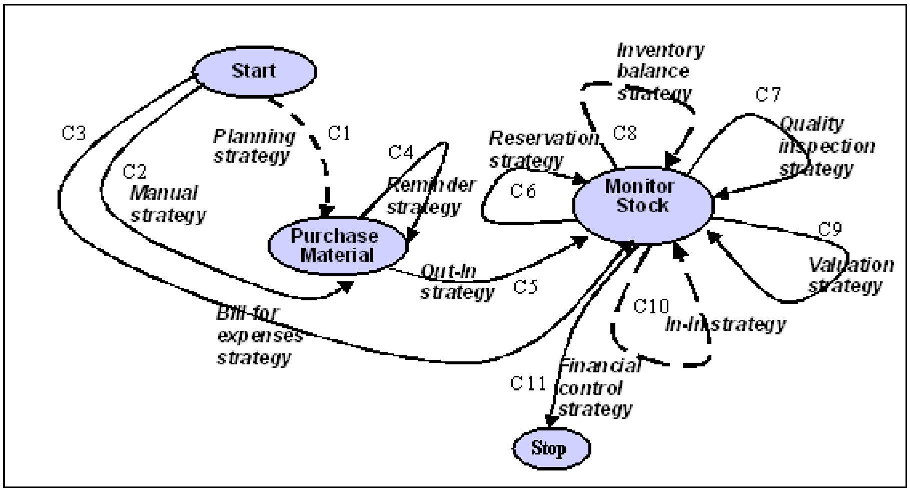


Fig. 1. The material management map

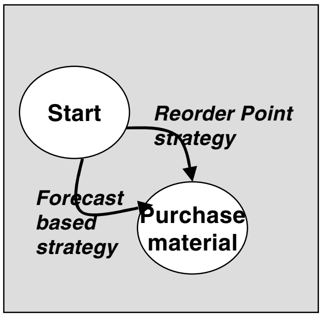


Fig. 2. The Planning strategy bundle

Finally, a *section* of a map can be *refined as another map*. This happens when it is possible to view the section as having its own intentions and associated strategies. The entire refined map then represents the section.

As a consequence of its intentional orientation, a map does not represent a flow of tasks. Rather, it presents a non deterministic ordering of intention / strategy selections to accomplish the main process intention. Besides, given the multi-path and multi-thread topologies, a map is able to present a global perspective of the diverse ways of achievement the process intention. The map of Figure 1 for example, shows 25 paths from *Start* to *Stop*, 5 following the *Bill for Expenses strategy*, 10 following the *Planning Strategy*, and 10 following the *Manual strategy.* All these paths allow to achieve the main process intention namely*, Satisfy Material Need Efficiently*.

The Map was selected as the intention-oriented model to be formalized in this paper because, unlike other intention-oriented models (e.g., i* [15]), it captures the flow of a process and establishes a direct relation between a goal and the actions that can be taken in order to achieve it. As well, maps employ a small set of constructs (as compared to i*) and therefore allow us to concentrate on the concept of intention and the way it drives a process.

2.2 An Overview of GPM

GPM is based on Bunge's ontology [3, 4], as adapted for information systems modeling (e.g., [8, 14]), for conceptual modeling, and for modeling business process concepts.

According to the ontological framework, the world is made of *things* that possess *properties*. Properties can be *intrinsic* (e.g. height) to things or *mutual* to several things (e.g. a person works for a company). Things can compose to form a *composite* thing that has *emergent* properties, namely, properties not possessed by the individuals composing it. Properties (intrinsic or mutual) are perceived by humans in terms of *attributes*, which can be represented as functions on time. The *state* of a thing is the set of values of all its attribute functions (also termed state variables). When properties of things change, these changes are manifested as state changes or *events*. State changes can happen either due to internal transformations in things (self action of a thing) or due to *interactions* among things. Not all states are possible, and not all state changes can occur. The rules governing possible states and state changes are termed *state laws* and *transition laws*, respectively. States can be classified as being *stable* or *unstable*, where an unstable state is a state that must change by law, and a stable state is a state that can only change as a result of an action of something external to the thing or the domain.

A *domain* is a part of the world of which we wish to model changes, and represents the scope of our control. A domain is a set of things and their interactions, and is represented by a set of state variables, which stand for the intrinsic and mutual properties of these things, including emergent properties of the domain itself. A *sub-domain* is a part of the domain, represented by a subset of the domain state variables. A sub-domain may be in a stable state while the entire domain is in an unstable state, meaning that a different part of the domain is currently subject to changes.

A *process* is a sequence of unstable states, transforming by law until a stable state is reached. A process is defined over a domain, which sets the boundaries of what is in a stable or an unstable state. Events that occur outside the domain are *external events* and they can activate the domain when it is in a stable state.

A process model in GPM is a quadruple <S, L, I, G>, where S is a set of states representing the domain of the process; L is the law, specified as mapping between subsets of states; I is a subset of unstable states, which are the initial states of the process after a triggering external event has occurred; G is a subset of stable states, which are the goal of the process. Subsets of states are specified by conditions defined over criterion functions in the state variables of the domain. Hence, a process starts when a certain condition on the state of the domain holds, and ends when its goal is reached, i.e., when another condition specified on the state of the domain holds. For example, the initial set of a production process can be specified as {s| Production

Order Status = "Released" AND Materials = "Available" AND Resources = "Available"}, which is a set of unstable states (that became unstable by the release of the production order). The goal of this production process can be specified as {s| Production Order Status = "Completed" AND Quality = "Approved"}, which is a set of stable states. The states in the goal set may differ from each other in the values of state variables such as production time and cost. Nevertheless, they all meet the condition specified. The criterion function defines the set of state variables that are relevant for determining that the process has reached its goal.

3 Interpreting Map in GPM Terms

In this section we interpret the concepts of Map in GPM terms and establish a set of concepts which is common to both formalisms.

3.1 Basic Concepts

A Map is specified as a set of intentions or goals to be achieved, and strategies for achieving them. The goals can be interpreted as sets of desired states, on which the strategies terminate. However, taking an action (strategy) aimed at reaching a goal does not necessarily end in attaining it. Some goals may require a number of actions to be performed before being achieved. In other cases an action may or may not achieve the goal, and, based on the result accomplished, that action may be repeated or another action may be taken. Still, the Map notation specifies every strategy as leading to an intention, even if it is not able to immediately achieve the goal. In fact, the intention a strategy leads to specifies *why* this strategy is taken rather than the goal it actually achieves. Hence, transforming this into state-based concepts, we may view an intention to which a strategy leads as having a "core", which is the goal to be attained, and a broader set of states in which some action towards attaining the goal has already been performed. Taking an action (strategy) aimed at reaching the goal, does not necessarily end in attaining the goal, but should reach a state which is closer to that goal than the one prior to the action. All the strategies that lead to a given intention must at least end in a state where some action towards attaining the goal has been done.

Hence, defining an intention in state-based terms should include two parts: a basic subset of states indicating that some action has been performed, and the goal set.

Formally expressed:

An *Intention* I is specified as $<B_I, G_I>$, where B_I is the intention *Basic* subset of states and G_I is the intention *Goal.*

Being "in" an intention I means being in a state $s \in B_I$. The Goal set is, naturally, a subset of the basic set of the intention, $G_I \subseteq B_I$. Based on the GPM notation, all the subsets of states are specified in terms of conditions over criterion functions.

For example, B_I of the intention *Purchase Material* in Figure 1 is the set of states where a purchase order was issued, and G_I is the set of states where the goods arrived from the supplier.

A *strategy* is an action by which an intention can be achieved.

A *section* in a Map is comprised of a source intention, a target intention, and a strategy leading from the source to the target. In GPM terms, a section is a mapping between subsets of states, hence it specifies the law. When going from one intention to another, a strategy may be selected based on (a) preferences and success expectations, or (b) the current situation. The latter case means that a strategy does not necessarily start on any state $s \in B_I$. Rather, it can start on a subset of the states in the Basic subset of its source intention. Similarly, it leads to a state belonging to a subset of its target intention Basic set, which is a result of the specific actions of the strategy. In Figure 1 the *Bill for Expenses* strategy will start on a subset of "emergency" states, and end on a subset of states where goods arrive and expenses are billed for. Two or more strategies that share a common initial subset of states are a bundle. It is therefore clear that a section is not only defined by the Basic set of its source and target intentions, but by more specific subsets of initial and final states. Formally expressed:

A section S from intention I to intention J is a mapping between an initial subset of states I_S and a final subset of states F_S, such that:

(a) $I_S \subseteq B_I$
(b) $F_S \subseteq B_J$

Going from intention I to intention J means trying to achieve the goal of J. Even if this goal is not achieved, the strategy taken should result in a state which is "closer" to that goal than the one prior to the action taken. We would like to be able to state that $|G_J - F_S| < |G_J - I_S|$. This is obvious when the section includes two different intentions, leading from intention I to J, since the desired goal $G_J \subseteq B_J$, on which the section ends.

However, for recursive sections, whose source and target intentions are the same one, the notion of distance between subsets of states should be examined.

Assume the goal criterion function relates to a single state variable (i.e., it has a single dimension). We shall also assume that this state variable exists in a domain of values where the operators >, <, = hold. This means that there is some kind of ordinality in the values that may be attained by state variables. This ordinality may be numerical, preference-based, or a result of procedural sequence. Then, moving along this dimension, the distance from the goal changes, and one can clearly identify that a strategy ends on a state which is closer to the goal than the state before. However, a criterion function may relate to a number of state variables (i.e., be multi-dimensional). Furthermore, these dimensions may have trade-off relations among them. Computing a precise distance from the goal in such situations may involve weighing techniques, thus the computed distance would depend on the weight assigned to each dimension. Nevertheless, changes in the distance with respect to each dimension separately can still be straightforward and easily perceived.

As an example, assume an intention of improving a production process. The improvement may be in terms of cost, time, and quality, which have trade-off relations among them. The improvement intention can be achieved through different strategies: cost-reduction strategy, quality-improvement strategy, time-reduction strategy, and so on. However, the cost-reduction strategy may inversely affect the product quality, the quality-improvement strategy may increase the cost, the time-reduction strategy may decrease the cost but damage the quality, etc. Yet, they are all valid strategies.

For any practical purpose, we may assume the process designer can clearly identify and evaluate whether a strategy “contributes” to achieving a goal along each dimension, and whether a final subset of states is “closer” to a goal set. For such purposes, a move along at least one dimension will be sufficient for determining that such contribution is made, and we can expect different strategies to contribute along different dimensions. Furthermore, a Map is a model that specifies possible and alternative paths (combinations of strategies), thus facilitating the selection of an appropriate strategy at run time, when the process is executed. Such selection may take into account trade-offs among dimensions and assign appropriate weights in the course of the decision making. However, this decision and its factors are situation-dependent, and not a part of the process modeling and design phase.

In summary, the computation of an absolute distance between subsets of states is situation dependent. The strategies in a map should clearly reduce the distance from the goal set along one dimension, thus establish a possible path to be taken, where the actual decision whether to take this path can be made in run time.

3.2 Section Classification

To gain more understanding about the nature of processes modeled by Maps, we shall now elaborate on types of sections, and differentiate them to classes based on their behavior. A section may either be recursive or non-recursive. Its initial subset of states (I_S) is a subset of the basic set of its source intention, and its final subset of states (F_S) is a subset of the basic set of its target intention. As discussed above, the final subset of states should be defined so that the section ends in a state whose distance from the goal is smaller than the initial distance.

Figure 3 outlines possible cases of sections, their initial and final subsets with respect to their source and target intentions.

Cases 1 and 2 are cases where the strategy leads to a state closer to the goal than the initial state, but not to the goal itself. In case 1 the strategy leads from intention I to intention J. Its initial set is a subset of the goal of I, as this goal must have been achieved before proceeding to the next intention. Reaching intention J means reaching a state where some action towards attaining the goal has been performed, but has not actually achieved the goal. It might be that the goal requires a number of actions to be performed, in which case a recursive strategy can be taken, or that the goal achievement depends on an external event yet to occur. For example, one may have an intention of calling a meeting to discuss a certain issue. A strategy of preparing a presentation and materials for discussion will bring to a state which is closer to the goal, but the goal itself will not be achieved until the other participants will arrive.

Case 2 is of a recursive strategy, going between two subsets of the same intention, getting closer to the goal, but not reaching it. A recursive strategy can be taken only after some action towards the goal has already been performed, so the current (initial) state belongs to the basic set of the intention. For example, after preparing the presentation and material for the meeting discussed earlier, a recursive strategy of this type would be to send reminders to all the participants. This strategy is a move in the direction of the goal (the meeting), but cannot reach the goal itself.

Cases 3 and 4 are cases where the strategy may lead to the goal and may not, as its final set of states partly overlaps the goal set. These are cases where only after an

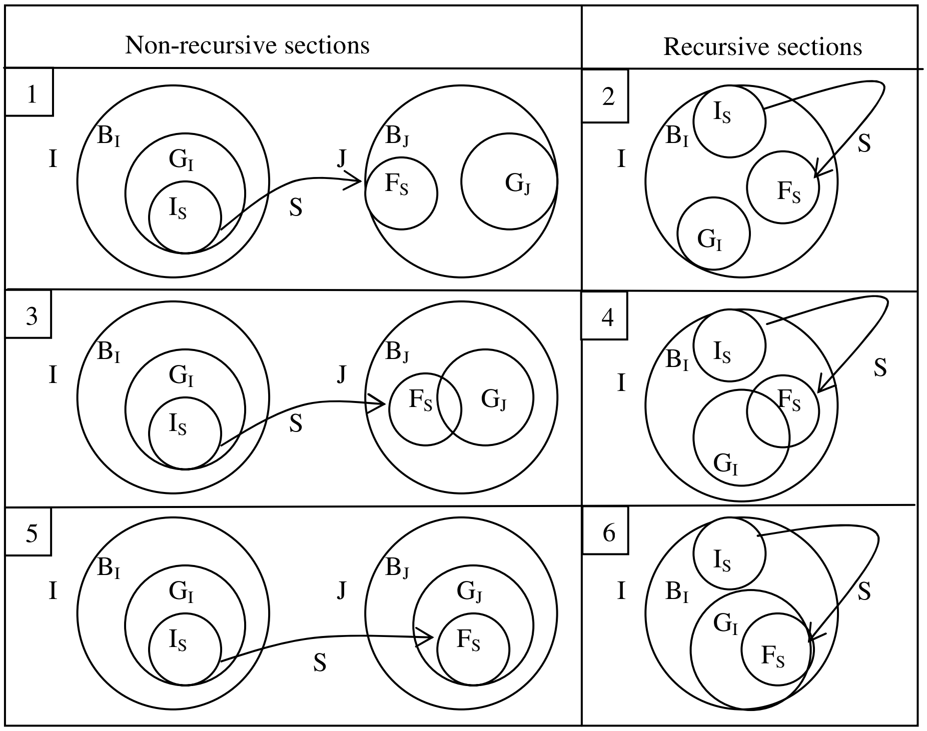


Fig. 3. Possible cases of Initial and Final subsets of sections

action was taken one can see if the goal has been achieved or if another (recursive) strategy should be taken. For example, when a mechanic tries to fix a car, he cannot be certain that the action he is taking will solve the problem indeed. If the problem is not solved, then he uses the new information he gained to reassess the situation and decide what his next strategy will be. In other words, the strategies that belong to cases 3 and 4 have a potential of reaching the goal, but are not certain to do so. If the goal is not reached by them then a recursive strategy is still needed.

Cases 5 and 6 are cases where the strategy leads to the goal directly and with certainty. For example, producing an item is a strategy that leads to the goal of having the item available.

It may seem as if there is a seventh case, which is unique to intentions of maintaining a certain state that has already been reached. In such cases some recursive strategies are aimed at verifying that the desired state is not violated. As an example, consider the intention of keeping one's body in a good health, which has a recursive strategy of periodical physical examinations. It may seem that the initial state of such strategies is *in* the goal set. However, it is not certain to be so. Uncertainty makes the initial state of these strategies to be outside the goal set, and verification brings the final state back to being in the goal.

The above analysis leads to the following general results:

Result 1: Let S be a non-recursive section (I, J, s) then $I_S \subseteq G_I$.

This result is based on the assumption that a new intention will be sought only once the former one has been achieved. This is true in most cases. However, there are cases where an intention is temporal, and ceases to exist at a moment in time even if it is not achieved. For example, the intention of saving a drowning person will cease to exist when it is clear that the person cannot be saved anymore. Then the strategy leading to whatever the next intention is, will not start at the goal set of the current intention.

Result 2: Let S be a recursive section (I, I, s) then $I_S \cap G_I = \emptyset$.

This result is straightforward, since a recursive strategy is not needed and will not be performed if we are already at the goal set of the intention.

4 Representation Transformation and Process Analysis

In this section we propose a procedure for transforming a Map to a GPM model, demonstrate it using the material management example, and discuss the insights that can be gained by this transformation.

4.1 The Transformation Procedure

Based on the concepts discussed and defined in Section 3, transforming a map to a GPM model would include the following steps:

1. Define each intention:
   (a) The intention Goal G_I, in terms of conditions over criterion functions.
   (b) The intention Basic Set B_I, in terms of conditions over criterion functions, specifying states where some action towards the goal has been performed. $G_I \subseteq B_I$.
   (c) The Start and Stop intentions can be defined as sets of states without distinction between a Basic Set and a Goal set.
2. Define sections as the law: a mappings between subsets of states (I_S and F_S), applying the following:
   (a) If the section is between two different intentions I and J, then:
      i. The initial set $I_S \subseteq G_I$, where the conditions additional to the Goal condition specify the situation in which a certain strategy is to be taken.
      ii. The final set $F_S \subseteq B_J$, where the conditions additional to the Basic Set condition specify the situation after the actions that were performed as part of the specific strategy.
   (b) If the section includes a recursive strategy from an intention I to itself, then:

   i. The initial set satisfies $I_S \subseteq B_I$ and $I_S \cap G_I = \varnothing$.
   ii. The final set F_S should be closer to G_I at least along one dimension of the G_I criterion function.
  (c) Bundles are addressed as sets of sections, whose I_S is mutual.
3. Repeat for each section refinement, placing I_S as the Start intention of the refined map and F_S as the Stop intention of the refined map.

To demonstrate the procedure, we shall use the material management example, and transform the map shown in Figure 1 to GPM representation. For simplicity, we shall not elaborate on the details of the bundles included in the map.

Step 1: Intention Definition
The states defining the intentions are specified and explained in Table 1.

The *Start* and *Stop* intentions specify states where nothing has been done and where the specific material handling is finished, respectively. Notice that the Goal conditions form subsets of the sets specified by the Basic conditions for the intentions of *Purchase Material* and *Monitor Stock*.

Table 1. Intention definition

Intention	**Basic condition**	**Goal condition**	**Explanation**
Start	(*Purchase Requisition*: not existing) AND (*Purchase Order*: not existing) AND (*Bill for Expenses*: not existing).		Start is when nothing has been done for purchasing
Purchase Material	*Purchase Order Status* ≤ “delivered”	*Sourced Goods* = “arrived”	Basic condition is that a purchase order exists. The goal is the arrival of material.
Monitor Stock	*Sourced Goods* = ”in stock”	*Stock Data*(attributes) = *Stocked Goods*(properties)	Basic condition is the existence of material in stock. Monitoring is keeping the data an accurate representation of reality.
Stop	(*Invoice* = “verified”) AND [(*Payment* = “authorized”) XOR (*Payment* = “blocked”)]		Stop is when payment of purchased goods is passed to the finances module.

Step 2: Define Sections as the Law
The states defining the sections are specified and explained in Table 2. The Initial and final conditions of each section include the basic or goal conditions of the relevant intentions (depending on the section type), and additional conditions. The sections are marked according to their mark in Figure 1. The explanations provide some additional assumptions made about the initial and final states of the sections.

Table 2. Section definition

Section	Initial condition	Final condition	Explanation
C1	(*Purchase Requisition*: not existing) AND (Purchase_ Order: not existing) AND (*Bill for Expenses*: not existing) AND (*Material Planning Type* = "automatic")	(*Purchase Order Status* < "delivered") AND (*Requisition Status* = "converted to order")	The planning strategy applies to materials whose planning type is defined as automatic. Requisitions are generated and converted to orders
C2	(*Purchase Requisition*: not existing) AND (*Purchase Order*: not existing) AND (*Bill for Expenses*: not existing) AND (*Material Planning Type* = "manual")	(*Purchase Order Status* < "delivered") AND (*Requisition Status* = "approved")	The manual strategy applies to materials whose planning type is defined as manual. Requisitions are generated and approved.
C3	(*Purchase Requisition*: not existing) AND (Purchase_ Order: not existing) AND (*Bill for Expenses*: not existing) AND (*Required Date – Current Date* < XX)	(*Sourced Goods* = "in stock") AND (*Expenses Billing* = "registered")	The bill for expense strategy applies when the material is needed suddenly and urgently, for a date which is less than XX days from the current
C4	(*Purchase Order Status* < "delivered") AND (*Delivery Date* < *Current Date*)	(*Purchase Order Status* < "delivered") AND (*Order History*: Reminder registered)	The reminder strategy is taken when the delivery date passed. It sends a reminder and registers it.
C5	(*Purchase Order Status* < "delivered") AND (*Sourced Goods* = "arrived")	(*Sourced Goods* = "in stock") AND (*Purchase Order status* = "delivered")	The out-in strategy is taken when goods arrive and puts them in stock.
C6	(*Sourced Goods* = "in stock") AND (*Reservation Request*)	(*Sourced Goods* = "in stock") AND (*Stock Status* = "reserved")	Stock status records a reservation request for the attributes to match reality
C7	(*Sourced Goods* = "in stock") AND (*Quality* = "not verified")	(*Sourced Goods* = "in stock") AND (*Quality* = "approved")	The quality attribute reflects uncertainty until quality is verified and approved
C8	(*Sourced Goods* = "in stock") AND (*Balance checking* = "needed")	(*Stock Data*(attributes) = *Stocked Goods*(properties)) AND (*Stock Documents* = "generated")	The balance strategy serves for verifying and documenting that the data attributes match reality
C9	(*Sourced Goods* = "in stock") AND (*Material Value* = "not recorded")	(*Sourced Goods* = "in stock") AND (*Material Value* = "recorded")	Recording the material value sets the value attribute to match reality
C10	(*Sourced Goods* = "in stock") AND (*Movement Request*)	(*Sourced Goods* = "in stock") AND (*Inventory Transaction* = "recorded")	Recording a physical movement so location data matches reality
C11	(*Stock Data*(attributes) = *Stocked Goods*(properties)) AND (*Invoice* = "arrived")	(*Invoice* = "verified") AND [(*Payment* = "authorized") XOR (*Payment* = "blocked")]	Verifying the invoice and authorizing/blocking its payment

Note, that most of the recursive sections of the *Monitor Stock* intention are not assumed to lead with certainty to the goal (which is a full match between the stock data as registered and its real properties). This is because each strategy contributes to the accurate representation with respect to a specific issue only (e.g., quality, location, reservation, etc.). These strategies may get to the goal along one of its dimensions, while the other dimensions are not involved. The only strategy that leads to the goal with certainty is the *Inventory Balance* strategy, where the accuracy of the data is verified completely.

The example used here does not include section refinements, hence, step 3 of the procedure shall not be demonstrated. A refinement of the section <*Purchase Material, Monitor Stock, Out-In strategy*> is presented in [10]. Applying the procedure to this refinement, the *Start* intention would be specified by the condition (*Purchase Order Status* < "delivered") AND (*Sourced Goods* = "arrived") and the *Stop* intention would be specified by (*Sourced Goods* = "in stock") AND (*Purchase Order Status* = "delivered").

4.2 Process Analysis

Transforming a Map representation to GPM representation enables identification of anomalies and deficiencies in the represented process. In particular, we may find indications for incompleteness of the specification and discontinuity of the process.

Deficiency of specification may be indicated by one of the following three cases:

(1) An intention in which the Basic Set includes states where no outgoing strategy is defined. Formally expressed: $B_I - \bigcup_{I_S \subseteq B_I} I_S \neq \phi$.

(2) An intention for which no strategy leads to the Goal set (i.e., all its ingoing strategies belong to cases 1 and 2 according to the classification of Figure 3). Formally: $F_s \cap G_I = \varnothing$ for every S leading to I.

(3) A section (except sections including the *Start* intention) for which no strategy leads to the initial set. Formally: $F_{S'} \cap I_S = \varnothing$ for every $S' \neq S$.

Case (1) is definitely a case of incompleteness, since it indicates the existence of states for which the law does not specify how to proceed. Assume the section <*Start*, *Purchase Material*, *Bill for expenses* > is not in figure1. Thus, due to the (*Bill for Expenses* : not existing) part of the basic condition of Start, this rule would help discovering the map is lacking a strategy.

Cases (2) and (3) may either indicate incompleteness or discontinuity.

In case (2): if the intention goal can only be achieved as a result of an external event then the process is non-continuous. Otherwise, a strategy for achieving the goal needs to be defined for the law to be complete.

In the material management example, the *Purchase Material* intention does not have any strategy that leads to its goal. The goal, which is the arrival of goods, can only be obtained by an external event. As suggested in [10, 13], in such cases a reminder strategy is needed in order to make sure that the process will not be waiting for that external event indefinitely. It is also possible to specify an exception handling strategy, to be taken if the external event does not occur after the reminder strategy has been taken. The exception handling strategy will cancel the purchase order and terminate the process.

In case (3): if the initial set of a section can only be achieved as a result of an external event then the process is non-continuous. Otherwise, a strategy for reaching a state where the next step can begin needs to be defined for the law to be complete.

In the material management example the *Financial Control* strategy can only start when an invoice has arrived (external event). A reminder strategy is needed in order to make sure that this happens. However, there is no explicit intention to which such strategy should lead. This may indicate that payment for goods is an intention by itself, which should be separated from the *Stop* intention.

The result of the analysis is presented in Figure 4.

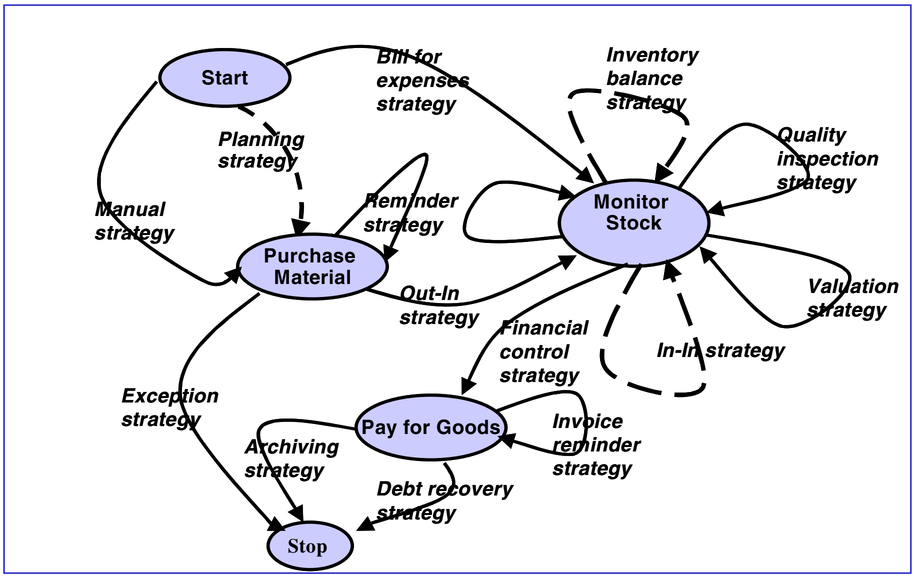


Fig. 4. Modified material management map

The new material management map includes an *Exception* strategy, leading from the *Purchase Material* intention to the *Stop* intention to be taken in cases of failure to receive the goods from the supplier. This strategy includes canceling the purchase order, so the process can start again.

As well, a new intention of *Pay for Goods* is added to the map. The *Financial Control* strategy leads to this intention, and it also has a recursive strategy of *Invoice Reminder*, aimed at assuring that the invoice arrives from the supplier and the process is not held waiting for it.

An *Archiving* strategy leads to the *Stop* intention. The *Archiving* strategy includes archiving the process data (purchase order, payment details, etc.) once the process is completed. We need to add as well a *Debt recovery* strategy to handle the case where the external event does not occur after the reminder strategy has been taken.

5 Conclusion

By combining the map and GPM we arrived at a formalism which precisely defines how human intentions drive a process. The result is an intention driven approach to process modeling that supports the analysis and verification of a designed map.

Modeling a business process in map terms provides an intentional view of what the process aims to achieve and the different ways to do it. Intentions in the map express goals to be attained and hide the details of how to implement them. Strategies are made explicit thus showing the different ways of achieving a goal. Finally, the map, as a multiple assembly of goals with multiple ways of achieving them, represents multiple variations in a business process.

Transforming a business process representation in map terms into GPM concepts provides a state-based formalisation of a map which is conducive to analysis and verification. We showed how reasoning on intention and section sets of states can help identifying incompleteness in the map specification. The basis of this reasoning is the clear distinction between the ultimate goal of an intention and its basic set, where some actions are already done, but the goal is not reached yet. This distinction led to the section classification presented in the paper, which corresponds to the validity analysis guidelines of GPM. Since the reasoning is semantic rather than technical, it is applicable to process models at a variety of scales.

However, it is clear that process analysis needs to be guided and we expect to lay down guidelines to help detecting anomalies and deficiencies in the represented process. It seems also, from the experiments that we conducted, that generic patterns for corrective actions in given deficiency situations could be designed. This will form the topic of future work.

References

1. ASAP World Consultancy and J. Blain et al, *Using SAP R/3,* Prentice Hall of India, 1999.
2. Bider, I., Johannesson, P., Perjons, E. (2002), "Goal-Oriented Patterns for Business Processes", Position paper for Workshop on Goal-Oriented Business Process Modeling (GBPM'02).
3. Bunge. M., *Treatise on Basic Philosophy: Vol. 3, Ontology I: The Furniture of the World.* Reidel, Boston, 1977.
4. Bunge. M., *Treatise on Basic Philosophy: Vol. 4, Ontology II: A World of Systems*, Reidel, Boston, 1979.
5. Dietz, J.L.G., Basic Notions Regarding Business Processes and Supporting Information Systems, Proceedings of BPMDS'04, *CAiSE'04 Workshops Proceedings*, Latvia, Riga, Vol. 2, pp. 160-168, 2004
6. Hammer, M. and Champy, J. (1994), *Reengineering the Corporation – A manifesto for Business Revolution*, Nicholas Brealey Publishing, London.
7. Ould, M. A., *Business Processes: Modeling and Analysis for Reengineering and Improvement*, John Wiley & Sons, 1995.
8. Paulson D. and Wand, Y., 1992, "An Automated Approach to Information Systems Decomposition," *IEEE Transactions on Software Engineering*, 18 (3), pp. 174-189.
9. C.Rolland, N. Prakash, A. Benjamen*, A Multi-Model View of Process Modelling,* Requirements Engineering Journal, 4(4) pp 169-187, 1999

10. Rolland C., Prakash N., *Bridging the gap between Organizational needs and ERP functionality*. Requirements Engineering Journal 5, 2000.
11. C. Salinesi, C. Rolland. *Fitting Business Models to Software Functionality: Exploring the Fitness Relationship*. 15th Conference on Advanced Information Systems Engineering, (CAISE'03), Springer-Verlag (pub), 2003.
12. Soffer, P., and Wand, Y., 2003, "On the Notion of Soft Goals in Business Process Modeling", *Business Process Management Journal* (to appear).
13. Soffer P. and Wand Y., 2004, Goal-driven Analysis of Process Model Validity, *Advanced Information Systems Engineering (CAiSE'04)* (LNCS 3084), p. 521-535
14. Wand, Y. and. Weber, R (1990), "An Ontological Model of an Information System", IEEE Transactions on Software Engineering, Vol. 16, No. 11, pp. 1282-1292.
15. Yu, E., and Mylopoulos, J. (1996), "Using Goals, Rules, and Methods to Support Reasoning in Business Process Reengineering", International Journal of Intelligent Systems in Accounting, Finance and Management, Vol. 5, pp. 1-13.

Pattern-Based Analysis of the Control-Flow Perspective of UML Activity Diagrams*

Petia Wohed[1], Wil M.P. van der Aalst[2,3], Marlon Dumas[3], Arthur H.M. ter Hofstede[3], and Nick Russell[3]

[1] Centre de Recherche en Automatique de Nancy, Université Henri Poincaré - Nancy 1/CNRS, BP239, 54506 Vandoeuvre les Nancy, France
petia.wohed@cran.uhp-nancy.fr

[2] Department of Technology Management, Eindhoven University of Technology, GPO Box 513, NL5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

[3] Faculty of Information Technology, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
{m.dumas, a.terhofstede, n.russell}@qut.edu.au

Abstract. The Unified Modelling Language (UML) is a well-known family of notations for software modelling. Recently, a new version of UML has been released. In this paper we examine the Activity Diagrams notation of this latest version of UML in terms of a collection of patterns developed for assessing control-flow capabilities of languages used in the area of process-aware information systems. The purpose of this analysis is to assess relative strengths and weaknesses of control-flow specification in Activity Diagrams and to identify ways of addressing potential deficiencies. In addition, the pattern-based analysis will yield typical solutions to practical process modelling problems and expose some of the ambiguities in the current UML 2.0 specification [9].

Keywords: UML, Activity Diagrams, Workflow Patterns, YAWL.

1 Introduction

The Unified Modelling Language (UML), frequently referred to as a de facto standard for software modelling, has recently undergone a significant upgrade to a new major version, namely UML 2.0[1]. Being a multi-purpose language, UML offers a spectrum of notations for capturing different aspects of software structure and behaviour. One of these notations, namely Activity Diagram (AD), is intended for modelling computational and business/organisational processes.

If the UML AD notation is to be adopted as a standard for business process modelling, it should compare favourably with other notations in this space. In order to facilitate such a comparison a comprehensive analysis on UML AD has been performed

* This work is funded in part by Interop NoE, IST-508011, and by the Australian Research Council under the Discovery Grant "Expressiveness Comparison and Interchange Facilitation Between Business Process Execution Languages".

[1] http://www.uml.org

and the results of it are reported here. The goal of this analysis has been to evaluate the capabilities and limitations of UML AD.

Evaluating and comparing modelling notations, particularly in the area of business processes, is a delicate endeavour. Empirical evaluations in terms of case studies may lead to valuable insights, but the conclusions are difficult to generalise due to their restricted scope. Theoretical evaluations on the other hand rely heavily on the evaluation framework they utilize. For instance, evaluations in terms of ontologies, such as the Bunge Wand and Weber (BWW) ontology [6,10], lead to coarse-grained results since these ontologies are composed of highly general concepts whose pertinence and manifestation in the context of business processes have not yet been studied.

In order to provide a more fine-grained analysis we have chosen a specialised evaluation framework. It is constituted by the set of workflow patterns defined on `www.workflowpatterns.com`. While originally developed as an instrument to evaluate languages supported by workflow systems, these patterns have also successfully been used to evaluate languages for process-aware information systems development [5,16,15,8,18]. Initially restricted to the control-flow perspective (i.e. the ordering of activities in a process) [3] these patterns have recently been extended in accordance with Jablonski and Bussler's classification [7] to accommodate the data perspective (which deals with data transfer between activities) [12] and the resource perspective (dealing with the resource allocation for the execution of the activities within a process) [11]. Moreover, based on the workflow patterns framework, a workflow definition language called YAWL (Yet Another Workflow Language) has been designed [2] and implemented [1]. YAWL provides a reference formalisation for the control-flow patterns.

Hence, we motivate our choice for using the workflow patterns framework by arguing that it is 1) well tested, 2) provides a sufficient level of granularity for a deep analysis and 3) it is the most complete and powerful framework existing for evaluating the capabilities of a process modelling language. Moreover, using a framework which has been applied numerous times, will facilitate comparison between the analysed languages.

Accordingly, this paper reports the results of an evaluation of UML 2.0 AD in terms of the workflow patterns. Due to space limitations it presents the results from the evaluation of the control-flow perspective only[2]. The contributions of the paper are:

- The identification of some limitations in UML AD and recommendations for addressing these with minimal disruption to the current design of the language.
- Discussions on how to capture the patterns in UML AD which provide elements of reusable knowledge for process designers that encounter these patterns.
- An analysis of UML AD, e.g., pointing out ambiguities in the behaviour part of the current version of the specification [9].

An evaluation of UML AD version 1.4 in terms of these patterns has been previously reported [5]. However, while in UML 1.4, activity diagrams are based on statecharts, in UML 2.0 they have a semantics defined in terms of token flow inspired by (though not fully based on) Petri nets. Thus, the evaluation of UML 2.0 leads to different results

[2] For an evaluation of the data perspective see [17].

than the one for UML 1.4. Furthermore, an attempt at evaluating UML 2.0 AD using the workflow patterns has been conducted by White [15]. However, some of the results reported by White may be questioned as explained in the remainder of this paper.

The paper is organised as follows. Section 2 briefly introduces the UML 2.0 AD notation. Section 3 reports on the evaluation of UML AD in terms of the control-flow patterns. Finally, Section 4 summarises the results and concludes the paper.

2 Overview of UML 2.0 AD

In UML AD the fundamental unit of behaviour specification is the *Action*. "An action takes a set of inputs and converts them to a set of outputs, though either or both sets may be empty." [9], p. 229[3]. Actions may also modify the state of the system. The language provides a very detailed action taxonomy, where more than 40 different action types are specified. However, a deep discussion of them is outside the scope of this paper and in Figure 1a we only present the action types that we have found to be relevant to our evaluation. These are *Action*, *Accept Event*, *Send Signal*, and *Call Behavior Action*.

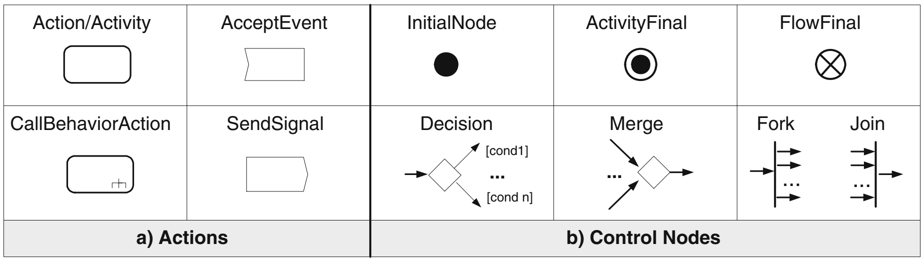


Fig. 1. UML 2.0 AD, Main Symbols

Furthermore, to present the overall behaviour of a system, the concept of *Activity* is used. Activities are composed of actions and/or other activities and they define dependencies between their elements. Graphically, they are composed of nodes and edges. The edges, used for connecting the nodes, define the sequential order between these. Nodes represent either *Actions*, *Activities*, *Data Objects*, or *control nodes*. The various types of control nodes are shown in Figure 1b.

3 Workflow Control-Flow Patterns in UML 2.0 AD

In this section, an analysis of UML AD version 2.0 is provided in terms of the control-flow patterns as defined in [3]. In this analysis YAWL (Yet Another Workflow Language) [2] is used as a reference realisation of the patterns (where appropriate). As YAWL is a formally defined language, its solutions for the patterns leave no room for ambiguity. Due to space restrictions the patterns themselves will not be discussed in detail here; for this the reader is referred to [3].

[3] In the remainder of this paper page numbers without reference refer to [9].

3.1 Basic Control-Flow Patterns, Multiple Choice and Multiple Merge

The first seven control-flow patterns, namely Sequence, Parallel Split, Synchronisation, Exclusive Choice, Simple Merge, Multiple Choice, and Multiple Merge are directly supported in UML AD. In fact, the first five of these patterns are supported by basically all process modelling and description languages and they correspond to control-flow constructs defined by the Workflow Management Coalition [14].

Figure 2 shows the solutions to the first seven patterns in UML AD and, as a point of comparison, in YAWL. The following paragraphs briefly discuss these solutions. Descriptions of the patterns are not included as they are relatively straightforward and they can be found in [3].

There are two ways of representing **Sequence** in YAWL (see Figure 2a). Two tasks can be connected directly, or they can have a condition (which corresponds to the concept of "place" in Petri nets) in between. Where two tasks are connected directly the condition in between exists in a formal sense, it is just not shown graphically. In UML, this basic pattern is solved in a very similar manner (see Figure 2b), i.e. through a control-flow arrow (p. 382), though UML does not explicitly support the notion of state hence there is no equivalent concept to the YAWL condition.

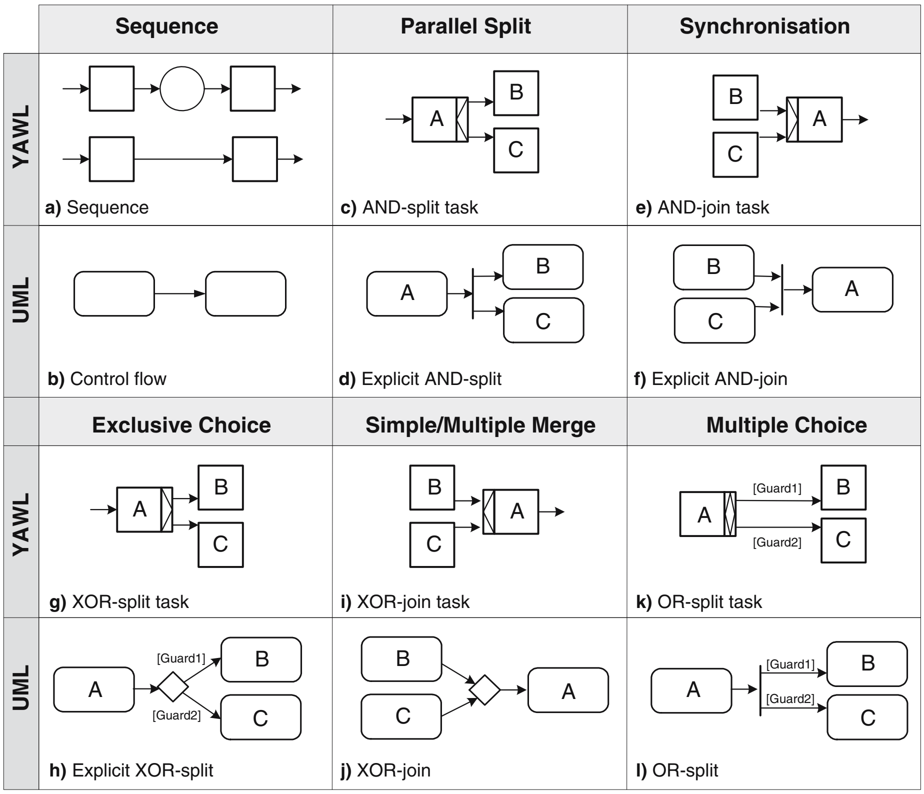


Fig. 2. Basic Control-flow Constructs in UML AD and in YAWL

The **Parallel Split** is captured in YAWL by an AND-split (see Figure 2c). In UML it is captured by a ForkNode, represented as a bar with one incoming edge and two or more outgoing edges (p. 404) (see Figure 2d). Furthermore, as "an action may have sets of incoming and outgoing activity edges..." (p. 336) and "when completed, an action execution offers [..] control tokens on all its outgoing control edges" (p. 337) the parallel split can also be modelled implicitly, by drawing the outgoing edges directly from the action node and omitting the fork node.

Synchronisation in YAWL is captured through the AND-join (see Figure 2e). In UML AD, the construct used for synchronisation is the JoinNode, i.e. a control node depicted as a bar with multiple incoming edges and one outgoing edge (p. 411) as shown in Figure 2f. A JoinNode may be associated with a condition (also called joinSpec). A joinSpec typically refers to the names of the incoming edges of the joinNode to which it is associated, but it may also be an arbitrary boolean expression. By default (i.e. if no joinSpec is provided as in Figure 2f), the joinSpec is taken to be an "and" of all the incoming edges, that is, a token has to be available at each of the incoming edges before the join node can emit a token, thus guaranteeing synchronisation of all incoming edges. Similarly to the parallel split solution, UML offers an "implicit" join notation: If the node that directly follows a join node is an action node, then the join node can be omitted and instead all the edges to be "joined" can be directly connected to the action node in question. The meaning of this implicit join is stated to be: "an action execution is created when all its [..] control flows prerequisites have been satisfied" (p. 337).

The **Exclusive Choice** in YAWL is captured by the XOR-split (see Figure 2g). In the YAWL environment, predicates specified for outgoing arcs of an XOR-split may overlap. In case multiple predicates evaluate to true, the arc with the highest preference (which is specified at design time) is selected. If all predicates evaluate to false, the default arc is chosen. The treatment of the XOR-split in YAWL guarantees that no matter what predicates are specified, exactly one outgoing branch will be chosen. In UML, a DecisionNode, graphically depicted by a diamond with one incoming edge and multiple outgoing edges, is used to represent this pattern (p. 387). The decision condition can be defined through "guards" attached to the outgoing edges (see Figure 2h). If the guard of more than one of the outgoing edges evaluates to true and if multiple edges accept the token and have approval from their targets for traversal at the same time, then the semantics of the construct depicted in Figure 2h is not defined (p. 388) and hence the "guards" should be made exclusive. A predefined "else" branch can be used which is chosen when none of the guards of the other branches evaluates to true. However, the use of "else" is optional.

In YAWL, the **Simple Merge** pattern is expressed using the XOR-join (see Figure 2i). In UML this pattern is represented by a MergeNode, that is, a diamond with several incoming edges and one outgoing edge (see Figure 2j). These solutions, for both YAWL and UML AD, also constitute a solution to the **Multiple Merge** pattern where parallelism may occur in the branches preceding the join and each completion of such a branch leads to (another) execution of the branch following the join.

In the **Multiple Choice** pattern, in contrast with the exclusive choice, zero, one, or multiple outgoing branches may be chosen. The multiple choice in YAWL is captured through the OR-split (see Figure 2k). It should be noted that in the YAWL environment at least one outgoing branch will be chosen, which makes its OR-split slightly less general than the pattern. In YAWL, the selection of at least one branch is guaranteed by the specification of a default branch which is chosen if none of the predicates evaluate to true (including the predicate associated with the default branch). In UML AD, the solution for this pattern is the same as the solution for the parallel split, except that in addition guards controlling which branches should be started have to be defined for the edges departing from the ForkNode (see Figure 2l).

3.2 Synchronising Merge

Description. A form of synchronisation where execution can proceed if and only if one of the incoming branches has completed and from the current state of the process it is not possible to reach a state where any of the other branches has completed.

Solution in YAWL. The main challenge of achieving this form of synchronisation is to be able to determine where more completions of incoming branches are to be expected. In the general case, this may require a computationally expensive state analysis.

In YAWL a special OR-join symbol, directly captures this pattern (see the task D in Figure 3a and in Figure 3b). The semantics of the OR-join are such that it is enabled if and only if an incoming branch has signaled completion and from the current state it is not possible to reach a state where another incoming branch signals completion. While this can handle workflows of a structured nature like that in Figure 3a, it can also handle non-structured workflows such as the one displayed in Figure 3b.

As a possible scenario for the example in Figure 3b, consider the situation where after completion of activity A both activities B and C are enabled. Now, if activity C completes while activity B is still running, activity D has to wait for B's completion. More precisely, it has to wait for the outcome of the decision whether activity E or activity F shall be enabled (as activity B is an AND-split only one of the activities E and F will be enabled). If activity F is enabled, activity D has to wait for F's completion. If, instead, activity E is enabled (and it is not possible that any of the other outgoing branches of the OR-join will be enabled) then activity D is enabled as well.

For a more complete treatment of OR-joins in YAWL see [19].

Solution in UML. No direct support is provided for this pattern. White [15] provides a tentative solution. However, there are two problems with this solution. Firstly, it assumes the existence of a corresponding OR-split (i.e., as the example in Figure 3a),

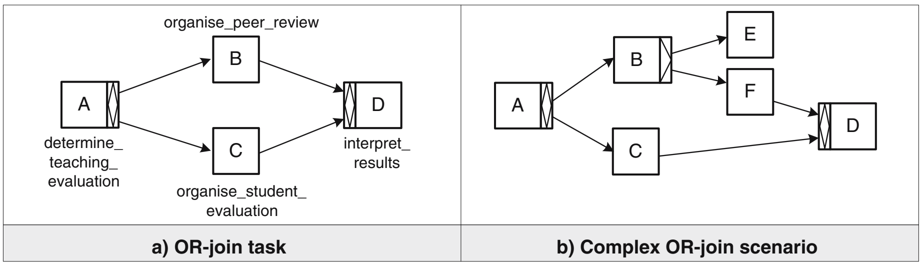

organise_peer_review
A
B
C
D
determine_ teaching_ evaluation
organise_student_ evaluation
interpret_ results
a) OR-join task
A
B
C
D
E
F
b) Complex OR-join scenario

Fig. 3. Synchronising Merge in YAWL

hence it would not be general enough to work in an unstructured context (as exemplified in Figure 3b). Secondly, the solution proposed by White includes the following joinSpec: “a *condition expression* that controls how many Tokens must arrive from the incoming control flow before a Token will continue through the outgoing control flow” ([15], p. 11). This *condition expression* is, however, not specified and it is not clear how it could be determined how many tokens to expect. In addition, even if somehow this could be detected, how can one deal with multiple tokens arriving on the same branch as a result of loops?

3.3 Discriminator

Description. A form of synchronisation for an activity where out of a number of incoming branches executing in parallel, the first branch to complete initiates the activity. When the other branches complete they do not cause another invocation of the activity. After all branches have completed the activity is ready to be triggered again (in order for it to be usable in the context of loops). The discriminator is a special case of the N-out-of-M Join (also called *partial join* [4]) as it corresponds to a 1-out-of-M Join.

Solution in YAWL. In YAWL, one of the ways to capture the discriminator involves the use of cancellation regions [2]. The discriminator is specified with a multiple merge and a cancellation region encompassing the incoming branches of the activity (see Figure 4a). In this realisation, the first branch to complete starts the activity involved, which then cancels the other executing incoming branches. This is not in exact conformance with the original definition of the pattern as it actually cancels the other branches. However, this choice is motivated by the fact that it is clear in this approach what the region is that is in the sphere of the discriminator giving it a clearer semantics.

Solution in UML. The solution in UML AD (see Figure 4c) uses the concept of InterruptibleActivityRegion (p. 409) which is very similar to the notion of cancellation region in YAWL. Hence, the solution is very close to the solution in YAWL. Furthermore, due to the use of weights (p. 352) on the interruptingEdge it also easily generalises to the N-out-of-M Join. YAWL provides direct support for N-out-of-M Join too, but the solution there is based on the concept of thresholds within the multiple instances task construct. This solution is shown in Figure 4b and the multiple instances task concept is further discussed in subsection 3.5.

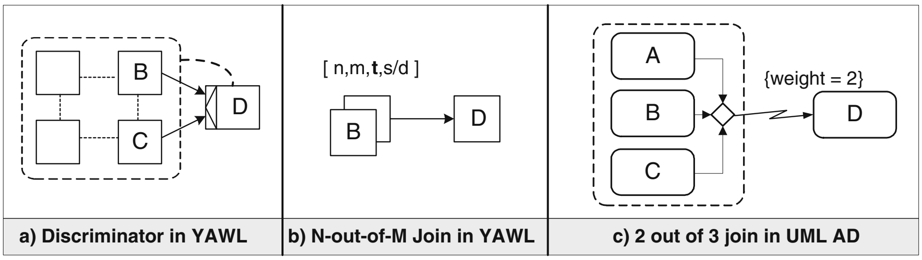


Fig. 4. Solutions for the Discriminator pattern

White [15] presents a solution which uses an expression which checks for each incoming branch whether it has completed. He claims that the first token to arrive will progress the flow and the other tokens will not. The expression given seems to be an annotation which is not part of the UML AD notation. In addition, it is unclear how this would work if the discriminator is to be activated more than once (e.g. because it appears in a loop).

3.4 Structural Patterns

In this section we briefly consider the patterns involving arbitrary cycles and implicit termination.

Description of Arbitrary Cycles. Some process specification approaches only allow the specification of loops with unique entry and exit points. *Arbitrary cycles* are loops with multiple ways of exiting the loop or multiple ways of entering the loop.

Both YAWL and UML AD (also pointed out by White [15]) support arbitrary cycles.

Description of Implicit Termination. A given subprocess should be terminated when there is nothing else to be done. In other words, there are no active activities in the subprocess and no other activity can be made active (and at the same time the subprocess is not in deadlock). This termination strategy is referred to as *implicit termination*.

Solution in YAWL. YAWL deliberately does not support implicit termination in order to force workflow designers to think carefully about workflow termination.

Solution in UML. UML AD provides direct support for this pattern. There are two notions for capturing termination namely, ActivityFinalNode and FlowFinalNode (see Figure 1). "A flow final destroys all tokens that arrive at it" (p. 403). It does not terminate the whole activity but only a flow within it. Implicit termination is then captured by ending every thread within an activity with a FlowFinalNode (same as in [15]).

3.5 Multiple Instances Patterns

This section focuses on the class of so-called "multiple instances" (MI) patterns. These patterns refer to situations where there can be more than one instance of a task active at the same time in the same case. The first of these patterns is concerned with the creation of multiple instances.

Description of MI without Synchronisation. Within the context of a single case (i.e., process instance) multiple instances of an activity can be created, i.e., there is a facility to spawn off new threads of control. Each of these threads of control is independent of other threads. Moreover, there is no need to synchronise these threads.

Solution of MI without Synchronisation in UML AD. Consider the UML AD example in Figure 5a which is taken from [9] (Figure 267, p. 404). This UML AD example provides a partial solution to the pattern. Instances of "Install Component" are "spawned-off" through a loop and the conditions associated with the DecisionNode will determine how many such instances will ultimately be created.

The next three patterns deal with the synchronisation of multiple instances. The first such pattern is **Multiple Instances with a Priori Design Time Knowledge** which can typically be supported by replicating the activity involved as many times as required.

This is possible in UML AD. The other two patterns deal with synchronisation of multiple instances where the number of instances is not known at design time.

The pattern **Multiple Instances with a Priori Runtime Knowledge** captures the situation when for one case an activity is enabled multiple times. The number of instances of a given activity for a given case varies and may depend on characteristics of the case or availability of resources, but is known at some stage during runtime, before the instances of that activity have to be created. Once all instances are completed some other activity needs to be started.

The pattern **Multiple Instances without a Priori Runtime Knowledge** is based on the previous pattern with the further complication that the number of instances to be created (and later on synchronised) are not know at any stage during runtime, before the instances have to be created. Even while some of the instances are being executed or have already completed, new ones can be created.

Solutions in YAWL. YAWL provides direct support for the multiple instance patterns. A multiple instance task in YAWL has four attributes: the minimum number of instances to be created; the maximum number; a threshold for continuation (where the semantics is that if all created instances have completed or the threshold has been reached the multiple instance task can complete); and an attribute with the possible values *static* and *dynamic* indicating whether or not it is possible to create new instances when a multiple instance task has been started (see Figure 4b).

Solution of Multiple Instances with a Priori Runtime Knowledge in UML AD. Here too we consider a UML AD solution taken from [9] (Figure 262, p. 401) as the basis for our discussion (see Figure 5b). In this example, the notion of ExpansionRegion, where the region consists of a single action, is used twice, once for $BookFlight$ and once for $BookHotel$. The small rectangles, divided into compartments and attached to a region, are meant to represent the input/output collections of elements, for the region. The action/s in the region is/are executed once for each element from the input collection ("or once per element position, if there are multiple collections" (p. 396)). "On each execution of the region, an output value from the region is inserted into an output collection at the same position as the input elements" (p. 395). The semantics of differing numbers in the inputs and their corresponding output collections is not clear. Furthermore, the way in which the multiple instances are executed, i.e., "parallel", "iterative", or "stream", is defined through an attribute of the ExpansionRegion node.

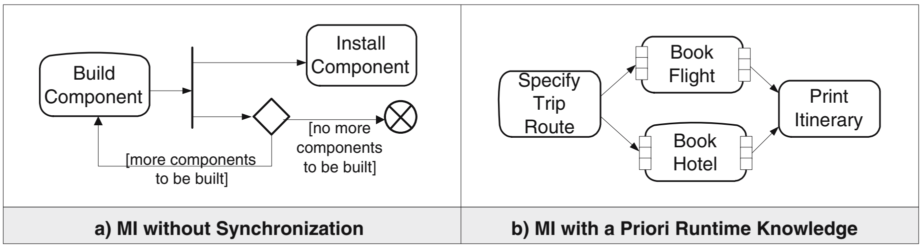

Build Component
Install Component
[more components to be built]
[no more components to be built]
a) MI without Synchronization
Specify Trip Route
Book Flight
Book Hotel
Print Itinerary
b) MI with a Priori Runtime Knowledge

Fig. 5. Multiple Instances in UML AD (solutions reprinted from Figures 267 and 262 from [9])

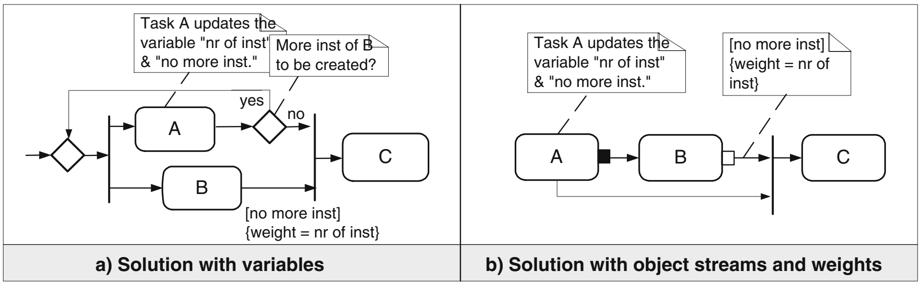

Task A updates the variable "nr of inst" & "no more inst."
More inst of B to be created?
yes
no
A
B
C
[no more inst] {weight = nr of inst}
a) Solution with variables
Task A updates the variable "nr of inst" & "no more inst."
[no more inst] {weight = nr of inst}
A
B
C
b) Solution with object streams and weights

Fig. 6. MI without a Priori Runtime Knowledge in UML AD

The UML specification is not explicit about the completion of an ExpansionRegion, only about its initiation. We assume that it is completed when all its instances have completed.

Solution of Multiple Instances without a Priori Runtime Knowledge in UML AD. This pattern is not directly supported in UML AD. The notion of expansion region can not be used here as once an expansion region receives the required input collection(s) no values can be added afterwards. There are however workarounds that achieve the required functionality. The first solution, which is depicted in Figure 6a, is inspired by Figure 265, p. 403, from the UML specification [9] and by the solution provided by White [15]. The idea is to keep track of two variables, one representing the number of instances created so far, and one, a boolean, capturing whether there is a need to create more instances. The solution in Figure 6a is however more precise as to how synchronisation is to occur than both the solution provided by White [15] and the solution shown in [9]. Another workaround is to use object streams and weights. This solution is depicted in Figure 6b and exploits the fact that both the guard and the weight of an edge need to be satisfied (p. 352).

3.6 Deferred Choice

Description. A point in the process where one of several branches is chosen. In contrast to the XOR-split, the choice is not made explicitly (e.g. based on data or a decision) but several alternatives are offered to the environment (this is akin to the pick construct in BPEL4WS[4], the choice construct in BPML[5], or the event-based decision gateway construct in BPMN[6]). However, in contrast to the OR-split, only one of the alternatives is executed. This means that once the environment activates one of the branches, the other alternative branches are withdrawn. It is important to note that the choice is delayed until the processing in one of the alternative branches is actually started, i.e. the moment of choice is as late as possible.

Solution in YAWL. YAWL is based on Petri nets and therefore directly supports the deferred choice construct. A condition (the YAWL term for a place) is specified as

[4] `www-128.ibm.com/developerworks/library/specification/ws-bpel`
[5] `www.bpmi.org`
[6] `www.bpmn.org`

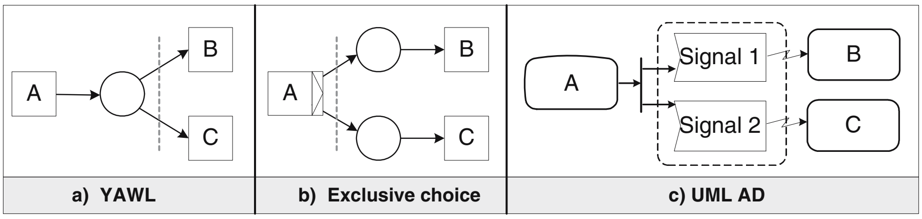


Fig. 7. Deferred Choice (in UML AD the solution is identical to the one presented in [15])

input to the activities that can result from the choice. At runtime, the alternative that is chosen consumes the token thus disabling the other alternatives.

Figure 7a illustrates the solution and contrasts it with the solution of the Exclusive Choice pattern in Figure 7b. The vertical dotted lines drawn in these figures are only meant for emphasising the moment of choice and they are not part of the language.

Solution in UML. This pattern is captured in UML AD through a fork and a set of accept signal actions, one preceding each action in the choice. In addition, an interruptible activity region encircling these signals is defined (see Figure 7c). The semantics is that the first signal received will enable and trigger the activity following it (which follows from the definition of AcceptEventAction on p. 250) and disable the rest of the activities included in the deferred choice by terminating all other remaining receive signal actions in the region (which follows from the definition of InterruptibleActivityRegion on p. 409). This solution is identical to that proposed by White [15].

In UML AD 1.4 this pattern can be captured using a "waiting state" (see [5]), but this is not applicable in UML AD 2.0 due to its lack of support for the notion of state.

3.7 Interleaved Parallel Routing

Description. Several activities are executed in an arbitrary order: The order is decided at runtime, and no two activities are executed concurrently (i.e. no two activities are active for the same process instance at the same time).

Solution in YAWL. Given that YAWL is based on Petri nets, the idea of a mutex place can be used as presented in [3]. This solution is shown in Figure 8a. The finesse of this solution is that it is general enough to also capture the case where sequences of activities have to be interleaved. In the figure, the sequences to be interleaved are A, B with C, D. It is important that A is always executed before B and likewise, C before D (on top of the requirement that no two activities are executed at the same time).

Solution in UML. Similar to UML AD version 1.4, this pattern is not directly supported in UML 2.0 although a workaround solution can be designed using signals that act as semaphores. This is due to the absence of the notion of state (or the notion of "place" as supported in Petri nets). A workaround solution is shown in Figure 8b. Before an action can start this signal needs to have been received, and after the completion of an action this signal needs to be sent as to indicate that another action may now execute. In this solution, an action can start after the preceding AcceptEven action received the signal S. After it completes, the following action sends the signal again so that another

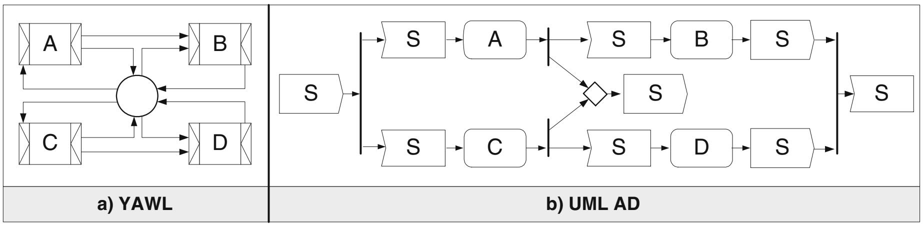


Fig. 8. Solutions for Interleaved Parallel Routing

action can be executed. As this other action may be in the same thread (e.g. after A it should be possible to execute not only C, but also B) there is a subtle issue of avoiding that an action of another thread will always "grab" the signal. This would occur if the SendSignal and the AcceptEvent actions were put in sequence, rather than in parallel, after completion of activities not last in a thread. The solution presented in Figure 8b assumes that 1) a signal can be sent from an action in a flow to an action in the same flow, 2) even though there may be multiple receivers ready to receive a signal only one of them will actually consume it (this is supported by the statement "[..] only one action accepts a given event occurrence, even if the event occurrence would satisfy multiple concurrently executing actions.", p. 250), 3) subthreads of a flow really execute in parallel, and 4) a signal can be sent before anyone is ready to receive it (this is the case as signals are stored in the objects associated with send/receive signal actions).

The two solutions presented by White [15] are not considered to be satisfactory. The first solution models the pattern by putting a verbal constraint on a couple of parallel activities stating that they are not to be run in parallel. The second solution provided by White uses the deferred choice pattern to capture the interleaved parallel routing pattern (as outlined in [3]). This solution suffers from combinatorial explosion as it boils down to enumerating all possible execution sequences of the activities involved.

3.8 Milestone

Description. A given activity can only be enabled if a certain milestone has been reached which has not yet expired. A milestone is defined as a point in the process where a given activity has finished and an activity following it has not yet started.

Solution in YAWL. YAWL directly supports the milestone pattern as it is based on Petri nets and therefore it can exploit the notion of state. A milestone can be realised through the use of arcs back and forth to a condition (which corresponds to the notion of place in Petri nets) testing whether a thread has reached a certain state (see Figure 9a).

Solution in UML. There is no direct support for the milestone in UML AD as the concept of state is not directly supported. A workaround can be devised with the use of signals, see Figure 9b. In the solution depicted in this figure, there is a race after the completion of A between continuing B, which has to await the receipt of $Signal1$ indicating that continuation of the thread is appropriate, and performing some other activity C. Activity C can only be performed after A has completed and before B has started. This is achieved by sending $Signal2$ which triggers $Signal3$ if indeed

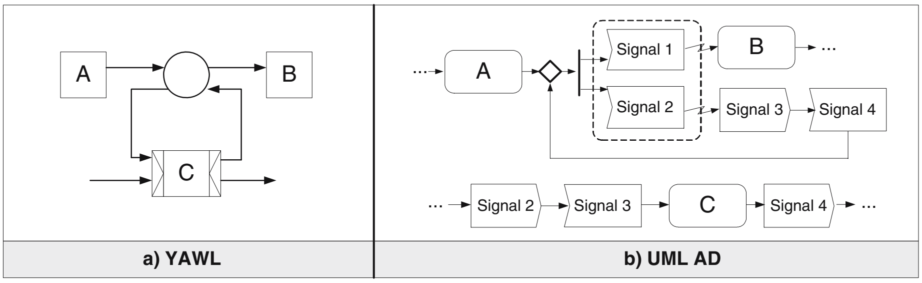


Fig. 9. Solutions for Milestone

the corresponding thread is in the correct state. If it is allowed to execute, then after completion, C issues $Signal4$ for indicating this.

The solution proposed by White [15] does not capture this pattern, as it does not model the expiration of the milestone. According to this solution an activity which *potentially* can be executed at a certain milestone, is *always* executed.

While workarounds exist for the state-based patterns, it is clear that mimicking the concept of a place as it exists in Petri nets through the use of signals may add a lot of complexity and could lead to models that are significantly less comprehensible. Moreover, many of the workarounds assume specific semantics for the constructs in UML AD. As yet, there are no formal semantics for UML AD and the workarounds may turn out to be invalid. Interpretations used by other authors suggest that there is currently no consensus on the semantics of the more advanced constructs.

3.9 Cancellation Patterns

There are two cancellation patterns: cancel activity and cancel case. As their semantics is straightforward we immediately focus on their solutions in YAWL and UML AD.

Solutions in YAWL. In Figure 10a execution of task B implies cancellation of task A, as this task is in the cancellation set of task B. In fact, any region can be chosen for cancellation so cancellation sets allow for cancellation of a single task, a whole case, and anything in between.

Solutions in UML. In UML AD, the cancel activity pattern can be captured as shown in Figure 10b. In this solution an interruptable region is used where apart from activity

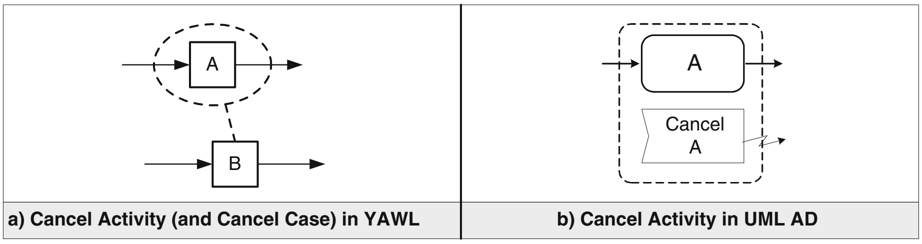

A
B
a) Cancel Activity (and Cancel Case) in YAWL
A
Cancel A
b) Cancel Activity in UML AD

Fig. 10. Cancellation concepts

A there is an AcceptEventAction ready to accept a signal indicating that A should be cancelled. If such a signal is received during the execution of activity A, and as an interruptingEdge is used, everything in the region (in this case only activity A) will be cancelled (p. 409). The solution in Figure 10b is inspired by Figure 274 on p. 410 of the UML specification [9]. It is also identical to the solution presented by White [15]. Note that due to the statement "If an AcceptEventAction has no incoming edges, then the action starts when the containing activity or structured node [i.e. the interruptible region in this case] does..."(p. 334) no incoming edge is used for the cancellation event.

In UML AD, the cancel case pattern is captured by the ActivityFinalNode: "A token reaching an activity final node terminates the activity [..], it stops all executing actions in the activity, and destroys all tokens in object nodes, except in the output activity parameter nodes."(p. 357). White [15] offers two solutions: one along the lines of the approach to cancel activity (by making the process to be cancelled an activity and running it in parallel with the cancellation event) and another using ActivityFinalNode.

4 Conclusion

Table 1 summarises the evaluation in terms of the control-flow patterns. A '+' indicates *direct support* for the pattern (i.e. there is a construct in the language that directly supports the pattern). The evaluation of UML 2.0 is contrasted with a previous evaluation of UML 1.4[7]. Overall, UML 2.0 is a clear improvement over UML 1.4 in terms of *direct support* for the control-flow patterns. In regards to the patterns that UML 2.0 AD does not directly support we would like to make the following recommendations:

– Given the difficulties in supporting state-based patterns, most notably the Interleaved Parallel Routing pattern and the Milestone pattern, it may be worthwhile to provide direct support for the notion of place as it exists in Petri nets. Petri net places capture the notion of "waiting state" in a much less restrictive way than AcceptEventAction do. Similar to YAWL, one could then allow for implicit places thereby avoiding places that unnecessarily clutter up the diagram.
– UML AD currently does not support the creation of new instances of an activity while other instances of that activity are already running. This could be resolved through extensions to the ExpansionRegion construct along the lines of the "multiple instance" tasks in YAWL.
– Given the lack of support for the Synchronising Merge, a concept similar to the OR-join as it exists in YAWL could be added to UML AD.

During this pattern-based analysis, we have identified several ambiguities in the current UML specification, for example regarding the behaviour of expansion regions when the size(s) of the input and output collections do not match as mentioned in section 3.5, or the behaviour of signals that are raised before any accept signal can consume them (section 3.7). In general, such ambiguities can be resolved by identifying relevant passages in the specification and giving them an interpretation, as we have done in this paper, but a formalisation would help in making more precise and reliable interpretations. Unfortunately, the UML AD notation is not yet formalised (although work in this direction is

[7] This evaluation is based on [5] and the table presented at `www.workflowpatterns.com`.

Table 1. Comparison of UML AD version 2.0 and version 1.4

Nr	Pattern	2.0	1.4	Nr	Pattern	2.0	1.4
1	Sequence	+	+	11	Implicit Termination	+	–
2	Parallel Split	+	+	12	MI without Synchronization	+	–
3	Synchronisation	+	+	13	MI with a priori Design Time Knowledge	+	+
4	Exclusive Choice	+	+	14	MI with a priori Runtime Knowledge	+	+
5	Simple Merge	+	+	15	MI without a priori Runtime Knowledge	–	–
6	Multiple Choice	+	–	16	Deferred Choice	+	+
7	Synchronising Merge	–	–	17	Interleaved Parallel Routing	–	–
8	Multiple Merge	+	–	18	Milestone	–	–
9	Discriminator	+	–	19	Cancel Activity	+	+
10	Arbitrary Cycles	+	–	20	Cancel Case	+	+

ongoing e.g. [13]) and there are inherent difficulties in assessing a language that does not have a commonly agreed upon formal semantics nor an execution environment. We hope, however, that the analysis reported here, where different solutions are presented and discussed in both UML and a formalised language, namely YAWL, will serve to clarify (even if not directly formalise) the semantics of many of the language constructs, and thereby motivate further improvements to the language.

References

1. W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and Implementation of the YAWL System. In A. Persson and J. Stirna, editors, *Proc. of the 16th Int. Conf. on Advanced Information Systems Engineering (CAiSE'04)*, volume 3084 of *LNCS*, pages 142–159. Springer, 2004.
2. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
3. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
4. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual Modelling of Workflows. In M.P. Papazoglou, editor, *Proc. of the 14th Int. Object-Oriented and Entity-Relationship Modelling Conf. (OOER)*, volume 1021 of *LNCS*, pages 341–354. Springer, 1998.
5. M. Dumas and A. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In M. Gogolla and C. Kobryn, editors, *Proc. of the 4th Int. Conf. on the Unified Modeling Language (UML01)*, volume 2185 of *LNCS*, pages 76–90. Springer Verlag, 2001.
6. P. Green and M. Rosemann. Applying Ontologies to Business and Systems Modeling Techniques and Perspectives: Lessons Learned. *Journal of Database Management*, 15(2):105–117, 2004.
7. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture, and Implementation.* International Thomson Computer Press, London, UK, 1996.
8. J-H. Kim and C. Huemer. Analysis, Transformation, and Improvements of ebXML Choreographies Based on Workflow Patterns. In R. Meersman and Z. Tari, editors, *Proc. of the OTM Confederated Int. Conf. (CoopIS, DOA, and ODBASE), Part I*, volume 3290 of *LNCS*, pages 66–84. Springer, 2004.
9. OMG. UML 2.0 Superstructure Specification. UML 2.0 Superstructure FTF convenience document ptc/04-10-02, 2004. `www.omg.org/cgi-bin/doc?ptc/2004-10-02`.

10. A.L. Opdahl and B. Henderson-Sellers. Ontological Evaluation of the UML Using the Bunge-Wand-Weber Model. *Software and System Modeling*, 1(1):43–67, 2002.
11. N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resouce Patterns: Identification, Representation and Tool Suppport. In O. Pastor and J. Falcão e Cunha, editors, *Proc. of 17th Int. Conf. on Advanced Information Systems Engineering (CAiSE05)*, volume 3520 of *LNCS*, pages 216–232. Springer, 2005.
12. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. In L. Delcambre, H.C. Mayr, J. Mylopoulos, and O. Pastor, editors, *to appear in Proc. of 24th Int. Conf. on Conceptual Modeling (ER05)*, LNCS. Springer Verlag, Oct 2005.
13. H. Störrle. Semantics of Control-Flow in UML 2.0 Activities. In P. Bottoni, C. Hundhausen, S. Levialdi, and G. Tortora, editors, *Proc. of IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'04)*, pages 235–242. Springer Verlag, 2004.
14. WfMC. Workflow Management Coalition Terminology & Glossary, Document Number WFMC-TC-1011, Document Status - Issue 3.0. Technical report, Workflow Management Coalition, Brussels, Belgium, 1999.
15. S. White. Process Modeling Notations and Workflow Patterns. In L. Fischer, editor, *Workflow Handbook 2004*, pages 265–294. Future Strategies Inc., Lighthouse Point, FL, USA, 2004.
16. P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Analysis of Web Services Composition Languages: The Case of BPEL4WS. In Il-Y. Song, S. W. Liddle, T. W. Ling, and P. Scheuermann, editors, *Proc. of 22nd Int. Conf. on Conceptual Modeling (ER 2003)*, volume 2813 of *LNCS*, pages 200–215. Springer, 2003.
17. P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-based Analysis of UML Activity Diagrams. BETA Working Paper Series, WP 129, Eindhoven University of Technology, Eindhoven, The Netherlands, 2004. `http://www.bpm.fit.qut.edu.au/projects/babel/docs/p242.pdf`.
18. P. Wohed, E. Perjons, M. Dumas, and A.H.M. ter Hofstede. Pattern Based Analysis of EAI Languages - The Case of the Business Modeling Language. In *Proc. of the 5th Int. Conf. on Enterprise Information Systems (ICEIS 2003)*, volume 3, pages 174–184, 2003.
19. M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a General, Formal and Decidable Approach to the OR-Join in Workflow Using Reset Nets. In G. Ciardo and P. Darondeau, editors, *Proc. of 26th Int. Conf. on Applications and Theory of Petri Nets 2005*, volume 3536 of *LNCS*, pages 423–443. Springer, 2005.

A Three-Layered XML View Model: A Practical Approach

Rajugan R.[1], Elizabeth Chang[2], Tharam S. Dillon[1], and Ling Feng[3]

[1] eXel Lab, Faculty of IT, University of Technology, Sydney, Australia
{rajugan, tharam}@it.uts.edu.au
[2] School of Information Systems, Curtin University of Technology, Australia
Elizabeth.Chang@cbs.cutin.edu.au
[3] Faculty of Computer Science, University of Twente, The Netherlands
ling@ewi.utwente.nl

Abstract. Since the early software models, abstraction and conceptual semantics have proven their importance in software engineering methodologies. For example, Object-Oriented conceptual modeling offers the power in describing and modeling real-world data semantics and their inter-relationships in a form that is precise and comprehensible to users. Conversely, XML is becoming the dominant standard for storing, describing and interchanging data among various Enterprises Information Systems and databases. With the increased reliance on such self-describing, schema-based, semi-structured data language/(s), there exists a requirement to model, design, and manipulate XML data and associated semantics at a higher level of abstraction than at the instance level. But, existing Object-Oriented conceptual modeling languages provide insufficient modeling constructs for utilizing XML schema like data descriptions and constraints, and most semi-structured schema languages lack the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans. To this end, it is interesting to investigate conceptual and schema formalisms as a means of providing higher level semantics in the context of XML-related data engineering. In this paper, we use XML view as a case in point and present a three-layered view model with illustrated examples taken from a real-world application domain. We focus on conceptual and schema view definitions, view constraints, and the conceptual query operators.

1 Introduction

In software engineering, many methodologies have been proposed to capture real-world problems into manageable segments, which can be communicated, modeled, and developed into robust maintainable software systems. Since the early software models, abstraction and conceptual semantics have proven their importance in software engineering methodologies. For example, in Object-Oriented (OO) conceptual models, they have the power in describing and modeling real-world data semantics and their inter-relationships in a form that is precise and comprehensible to users [17, 23]. With the emergence of semi-structured data, Semantic Web (SW) [44], web services [46], and ubiquitous systems, it is important to investigate new data models for Enterprise Information Systems (EIS) that can correlate and co-exist with heterogeneous, multifunctional schemas under the context of Enterprise Content

Management [5]. Any such data models should accommodate heterogeneous schemas and changing model and data requirements at a higher level of abstraction, yet adoptable to the current software engineering paradigm.

Conversely, since the introduction of eXtensible Markup Language (XML) [47], it is fast emerging as the dominant standard for storing, describing, and interchanging data among various EIS and heterogeneous databases. In combination with XML Schema [49], which provides rich facilities for constraining and defining XML content, XML provides an ideal platform and flexibility for capturing and representing complex EIS data formats. But, OO modeling languages (such as UML [34], Extended-ER[19]) provide insufficient modeling constructs for utilizing XML schema based data descriptions and constraints. Also, XML Schema lacks the ability to provide higher levels of abstraction (such as conceptual models) that are easily understood by humans [20, 21].

In data engineering, some forms of data "abstraction" and perspectives have been provided by the view formalisms and have been supported by data model specific query language. Since the introduction of relational data model [16, 19], motivation for views has changed over the last two decades. At present, views are widely used in, (i) user access and user access control applications [41]; (ii) defining user perspectives/profiles [10]; (iii) designing data perspectives; (iv) dimensional data modeling [32]; (v) providing improved performance and logical abstraction (materialized views) in data warehouse/OLAP and web-data cache environments [22, 24, 31, 40]; (vi) web portals & profiles; and (vii) Semantic Web, for sub-ontology or Ontology views [42, 51]. From this list, it is apparent that the uses and applications of views are realized more than their originally intended purpose proposed by Date et al. (i.e., the 2-Es - data Extraction and Elaboration [52]), with extensive research being carried out by both researchers and industry to improve their design, construction, and performance. Yet, in our view, the view concept still remains as a data language and model dependent low-level (instance) construct. In the OO paradigm, the modeling languages provide minimal or no semantics to capture abstract view formalisms at the conceptual level [35, 52] and the existing XML technology standards have no support for concrete view formalisms.

To tackle this issue, in our early work, we have proposed to extend the concept of semi-structured XML view with conceptual and schemata notions [35]. In this paper, we present in a systematic way, (i) a view formalism for XML with there levels of abstractions, namely, *concept, schema,* and *instance*; (ii) detailed modeling primitives for such a view formalism, including constraints and conceptual operators; and (iii) the transformation methodology across various abstraction levels. Please note that the intention of this paper is neither to propose a new view standard for XML nor extensions to XML query languages. Rather, we focus on providing XML view mechanism at the conceptual, schema, and document levels by means of OO conceptual modeling. To help illustrate our concepts, we conduct a real-world case study in a fictitious global logistic company called LWC & e-Solutions Inc., e-Sol in short. Also, our three-layered view model has been utilized in real-world, data intensive application scenarios [36, 39] such as; (i) design of XML document warehouses and distributed FACT repositories, (ii) design of websites and web portals, (iii) design of User Access Control (UAC) and UAC middleware, and (4) design of (Ontology) views for Semantic Web.

The rest of this paper is organized as follows. In section 2, we review early work in view related domains, followed by the description of our case study example in section 3. This is followed by the formal introduction of our three-layered XML view model in section 4. We detail each of the three layers in section 5, 6, and 7, respectively. We conclude the paper and outline future work in section 8.

2 Related Work

Here we first briefly look at the history of the view mechanisms available today and some of the proposals for new view mechanisms supporting semi-structured data and constraint specification for views. Existing view models can be grouped into four categories, namely; (a) early (namely relational) view models, (b) OO view models, (c) semi-structured (namely XML) view models and (d) views for SW.

2.1 Early View Models

The relational view formalism [18, 19, 24, 26, 27] has been discussed extensively in many industrial and research forums since its proposal by Date [16]. The relational (classical) definition of a view is based on ANSI/SPAC three-schema architecture (Tsichritzis & Klug 1978) [16, 19, 27], where a view is treated as a *virtual relation*, constructed by a query which is executed on one or more stored relations [16, 19]. Later the concept of view was extended to support complex queries and/or aggregate/summary queries. Generally, a relational view definition is persistent which contains the list of attributes or elements that are incorporated within the view, together with the declarations on how to extract those elements from the underlying stored relations or classes, or from another view [16, 19]. A relational view can be queried, joined with another relations or classes, and be included in another view definition.

During the OO revolution, the relational view definitions were extended to OO data models by Won Kim et al. [26, 27], Abiteboul et al. [2], and Chang [13]. Here the views were defined in a synonymous manner to the relational model and/or extending the relational definition [26] when needed (supporting the 2-Es; data Extraction, Elaboration and some research directions towards data Extension). They included the idea of the *virtual class*. Both relational and OO view concepts make two implicit assumptions; that the underlying data is structured and there exists a fixed data model and a data access/query language. But only Chang et al. allows some form of abstraction at a higher level, a view definition in the form of *abstract views* [13]. All other view definitions are defined at the data manipulation language level. This we argue is not enough to provide a real-world scenario and/or abstraction to complex domain. We argue that, providing view formalism at the conceptual level will improve the resulting view implementation, similar to a conceptual model of a software system.

2.2 Views for Semi-structured Data

Since the emergence XML, the need for semi-structured data models, which have to be independent of the fixed data models and data access, violates fundamental properties of persistent data models. Many researchers attempted to solve these issues by using graph based [55] and/or semi-structured data models [3, 28]. Again, the actual view definitions are only available at the lower level of the implementation and not at the conceptual level.

One of the early discussions on XML view was by Serge Abiteboul [1] and later more formally by Sophie Cluet et al. [15]. They proposed a declarative notion of XML views. Abiteboul et al. pointed out that, a view for XML, unlike classical views, should do more than just providing different presentation of underlying data [1]. This, he argues, arises mainly due to the nature (semi-structured) and the usage (primarily as common data model for heterogeneous data on the web) of XML. He also argues that, an XML view specification should rely on a data model (like ODMG [8] model) and a query language. In the paper [15], they discuss in detail on how abstract paths/DTDs are mapped to concrete paths/DTDs. These concepts, which are implemented in the Xyleme project [29], provide one of the most comprehensive mechanisms to construct an XML view to-date. The Xyleme project uses an extension of ODMG Object Query Language (OQL) to implement such an XML view. But, in relation to conceptual modeling, these view concepts provide no support. The view formalism is derived from the instantiated XML documents (instant level) and is associated with DTD in comparison to flexible XML Schema. Also, the Xyleme view concept is mainly focused on web based XML data. Other view models for XML include; (a) the MIX (Mediation of Information using XML) view system [30] and (b) an intuitive view model for XML using Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) [14]. This is one of the first view model that supports some of abstraction above the data language level.

In related work in Semantic Web (SW) [45] paradigm, some work has been done in views for SW [42], where the authors proposed a view formalism for RDF document with support for Resource Description Framework (RDF) [43] schema (using a RDF schema supported query language called RQL). This is one of the early works focused purely on RDF/SW paradigm and has sufficient support for logical modeling of RDF views. But RDF is an object-attribute-value triple, where it implies object has an attribute with a value [21]. But RDF makes only intentional semantics and not data modeling semantics. Therefore, unlike views for XML, views for such RDF (both logical and concrete) have no tangible scope outside its domain. In related area of research, the authors of the work propose a logical view formalism for ontology [51] with limited support for conceptual extensions, where materialized ontology views are derived from conceptual/abstract view extensions.

2.3 View Constraints

In data modeling, specifications often involve constraints. In the case of views, it is usually specified by the data language in which they are defined in. For example, in relational model, views are defined using SQL and a limited set of constraints can be defined using SQL[16, 19], namely, (1) presentation specific (such as display headings, column width, pattern order etc), (2) range and string patterns for aggregate fields, (3) input formats for updatable views, and (4) other DBMS specific (such view materialization, table block, size, caching options etc).

In Object-Relational and OO models, views had similar constraints but they are more extensive and explicit due to the data model. The views here are constructed and specified by DBMS specific (such as OQL [8]) and/or external languages (such as C++, Java or O_2C [2]). It is a similar situation in views for semi-structured data paradigm, where rich set of view constrains are defined using languages such as OQL based LOREL [4]. But the work by authors of [14] provides some form of higher-level view constraints (under ORA-SS model) for XML views, while the work in [42] provides some form of logical level view constraints to be defined in views for in SW/RDF paradigm. Here, for our view formalism, we look into using UML/OCL as our view constraint specification language. Also, our work should not be confused with work such as [7], where authors use OCL to "model" (not to specify) relational views, which utilizes OCL from a data engineering point of view than a for constraint specification.

3 An Illustrative Case Study

The e-Sol Inc. aims to provide logistics, warehouse, and cold storage space for its global customers and collaborative partners. The e-Sol solution includes a standalone and distributed Warehouse Management System (WMS/e-WMS), and a Logistics Management System (LMS/e-LMS) on an integrated e-Business framework called e-Hub [11] for all inter-connected services for customers, business customers, collaborative partner companies, and LWC staff (for e-commerce B2B and B2C). Some real-world applications of such company, its operations and IT infrastructure can be found in [11, 12, 25].

In WMS (Fig. 3 & 4), customers book/reserve warehouse and cold storage space for their goods. They send in a request to warehouse staff via fax, email, or phone, and depending on warehouse capacity and customers' grade (individual, company or collaborative partner), they get a booking confirmation and a price quote. In addition, customers can also request additional services such as logistics, packing, packaging etc. When the goods physically arrive at the warehouse, they are stamped, sorted, assigned lots numbers and entered into the warehouse database (in Lots-Master). From that day onwards, customers get regular invoices for payments. In addition, customers can ask the warehouse to handle partial sales of their goods to other warehouse customers (updates Lots-Movement and Goods-Transfer), sales to overseas (handled by LMS) or take out the goods in full or in partial (Lots-Movement). Also customer can check, monitor their lots, buy/sell lots and pay orders via an e-Commerce system called e-WMS. In LMS, customers use/request logistics services (warehouse or third-party logistics providers) provided by the warehouse chains. This service can be regional or global including multi-national shipping companies. Like e-WMS, e-LMS provide customers and warehouses an e-Commerce based system to do business. In e-Hub, all warehouse services are integrated to provide one-stop warehouse services (warehouse, logistics, auction, goods tracking, payment etc) to customers, third-party collaborators and potential customers. A context diagram of the system is given in Fig. 1, followed by some detailed models in Fig. 3-4.

In e-Sol, due to the business process, data have to be in different formats to support multiple systems, customers, warehouses and logistics providers. Also, data have to be duplicated at various points in time, in multiple databases, to support collaborative business needs. In addition, since new customers/providers to join the system (or leave), the data formats has to be dynamic and should be efficiently duplicated without loss of semantics. This presents an opportunity to investigate how to use our XML conceptual, schema and instance views to design e-Sol at a higher level of abstractions to support changing business, environments, and data formats.

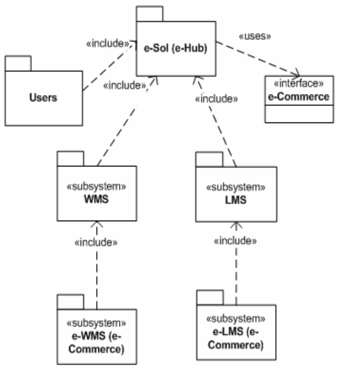


Fig. 1. e-Sol, context diagram

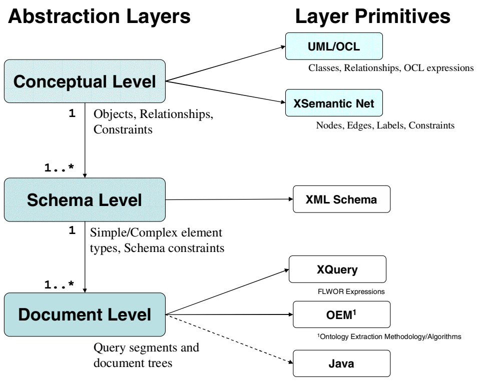

Abstraction Layers
Layer Primitives
Conceptual Level
UML/OCL
Classes, Relationships, OCL expressions
XSemantic Net
Nodes, Edges, Labels, Constraints
1
Objects, Relationships, Constraints
1..*
Schema Level
XML Schema
1
Simple/Complex element types, Schema constraints
1..*
Document Level
XQuery
FLWOR Expressions
OEM¹
¹Ontology Extraction Methodology/Algorithms
Java
Query segments and document trees

Fig. 2. Three-layered view model for XML

4 A Three-Layered XML View Model

Our XML view study is based on the following two postulates about the real world.

Postulate 1: The term ***context*** refers to the domain that interests an organization as a whole. It is more than a measure, and implies a meaningful collection of objects, relationships among these objects, as well as some constraints associated with the objects and their relationships, which are relevant to its applications. For example, "`people`", "`order`", and "`customer`" can be examples of context in the e-Sol system.

Postulate 2: The term ***view*** refers to a certain perspective of the context that makes sense to one or more stakeholders of the organization or an organization unit at a given point in time. For example, "`processed-order`" and "`overdue-order`" are two contrasting views in the "`order`" context of the e-Sol system.

Fig. 2 outlines our XML view model, which is comprised of three different levels, namely, *conceptual level, schema level*, and *instance level.*

(i) The top conceptual level describes the structure and semantics of XML views in a way which is more comprehensible to human users. It hides the details of view implementation and concentrates on describing objects, relationships among the objects, as well as the associated constraints upon the objects and relationships. This level can be modeled using some well-established modeling language such as UML [34], or our developed XML-specific XSemantic Net [20, 39], etc. Thus, the modeling primitives include object, attribute, relationship, and constraint. The output of this level is a well-defined *valid* conceptual model in UML, XSemantic Net, or even OMG's MOF (Meta-Object-Factory), which can be either visual (such as UML class diagrams) or textual (in the case of UML/XMI models).

(ii) The middle scheme level describes the schema of XML views for the view implementation, using the XML Schema language. Views at the conceptual level are mapped into the views at the schema level via the transformation mechanism developed in our previous work [20, 21]. The output of this level will be in either textual (such as XML Schema language) or some visual notations that comply from the schema language (such as graph).

(iii) The bottom instance level implies a fragment of instantiated XML data, which conforms to the corresponding view schema defined at the upper level.

For simplicity, we also call XML view at conceptual level *XML conceptual view*, XML view at schema level *XML schema view*, and XML view at instance level *XML instance view*. We elaborate each of the three levels in the following sections.

5 Conceptual Views

Context is presented in UML using modeling primitives like object, attribute, relationship and constraint in this study. To enable the construction of a valid XML conceptual view from a context, we introduce the notion of *conceptual operator* whose details can be found in our work [37]. These conceptual level operators are comparable to relational operators in the relational model, but they operate on conceptual level objects and relationships. They are grouped into set operators, namely union, difference, intersection, Cartesian product and unary operators namely projection, rename, restructure, selection and joins, and can facilitate systematic construction of conceptual views from context. These conceptual operators can be easily transformed into query segments, user-defined functions and/or procedures for implementation. By doing so, they help the modeler to capture view construct at the abstract level without knowing or worrying about query/language syntax.

***Definition 1*:** An **XML conceptual view V^c** is a 4-ary tuple $V^c = (V^c_{name}, V^c_{obj}, V^c_{rel}, V^c_{constraint})$, where V^c_{name} is the name of the XML conceptual view V^c, V^c_{obj} is a set of objects in V^c, V^c_{rel} is a set of object relationships in V^c, and $V^c_{constraint}$ is a set of constraints associated with V^c_{obj} and V^c_{rel} in V^c.

***Definition 2*:** Let $C = (C_{name}, C_{obj}, C_{rel}, C_{constraint})$ denote a context which consists of a context name C_{name}, a set of objects C_{obj}, a set of object relationships C_{rel}, and a set of constraints associated with its objects and relationships $C_{constraint}$. Let $\bar{\lambda}$ be a set of conceptual operators. $V^c = (V^c_{name}, V^c_{obj}, V^c_{rel}, V^c_{constraint})$ is called a **valid XML conceptual view of the context C**, if and only if the following conditions satisfy:

(i) For any object $\forall o \in V^c_{obj}$, there exist objects $\exists o_1, \ldots, o_n \in C_{obj}$, such that $o = \lambda_1 \ldots \lambda_m (o_1, \ldots, o_n)$ where $\lambda_1 \ldots \lambda_m \in \bar{\lambda}$. That is, o is a newly derived object from existing objects $o_1, \ldots, o_n$ in the context via a series of conceptual operators $\lambda_{1,} \ldots \lambda_m$ like select, join, etc.

(ii) For any constraint $\forall c \in V^{c}_{constraint}$, there exists a constraint $\exists c' \in C_{constraint}$ or a new constraint c'' constraints associated with V^{c}_{obj} or V_{rel}.

(iii) For any hierarchical relationship $\forall r^{h} \in V^{c}_{rel}$, there *does not exist* a relationship between one or more and V^{c}_{obj} and C_{obj}.

(iv) For any association relationship/dependency relationships $\forall r^{a} \in V^{c}_{rel}$, there *may exist a relationship between one or more* V^{c}_{obj} *and* C_{obj}.

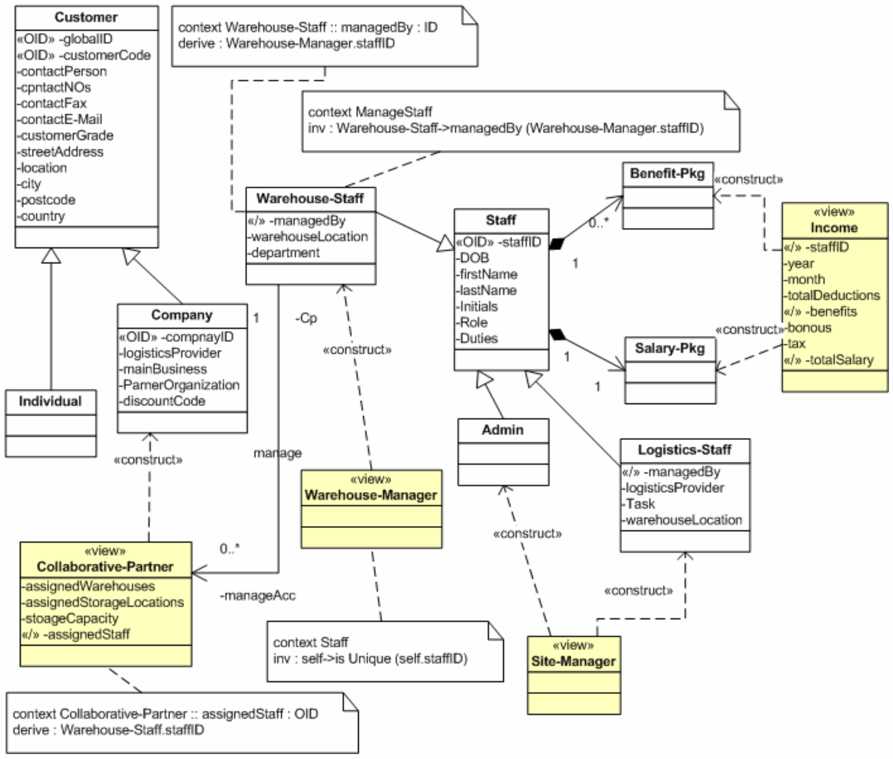


Fig. 3. e-Sol, core user model, relationships and constraints

Example 1: In Fig. 3, "`Warehouse-Manager`" is a valid XML conceptual view, named in the context of "`Staff`". Its objects are taken from the e-Sol "`Core-Users`" UML objects via the conceptual select [37] operator $\sigma_{warehouse\text{-}Staff.Role=\text{“}manager\text{”}}$ (*Core-Users*). Similarly, the conceptual view of name "`Site-Manager`" (Fig. 3) in the given context "`Staff`" is constructed via both SELECT and PROJECT operators.

Example 2: Conceptual views (Fig. 4), "`Customer-History`", "`Lot-Master-Charge-History`" and "`Rent-Warehouse-Space-History`" are perspectives / views in the context of "`Warehouse-History`" of the e-Sol system.

Example 3: Conceptual view (Fig. 3), "`Collaborative-Partner`" is a perspectives / view in the context of "`Customer`" in e-Sol.

Example 4: In the case of conceptual view "`Income`" (shown in Fig. 3), the conceptual construct is a conceptual JOIN operator with join conditions, where x = `Staff`, y = `Salary-Pkg` and z = `Benefit-Pkg`:

$$(x \rightarrow_{(x.staffID=y.staffID)} y) \text{ AND } (x \rightarrow_{(x.staffID=z.staffID)} z).$$

Example 5: In Fig.4 illustrates another valid conceptual view "`Lot-Master-Charge-History`" in the given context "`Lot-Management`". Here, at the conceptual level, it is stated as a *materialized* conceptual view, implying that it is a persistence view during the life time of the system. This characteristic is also stated in the OCL statement (Fig. 4).

It should be noted that, in our model, when referring to conceptual view OCL statements, it is provided only as a complement (such stating derived attributes, uniqueness etc.) to the *visual* UML model. Our aim is to keep the view definitions as *visual* as possible (in direct contrast to the work [7]) and avoid ambiguous and imprecise textual specifications of the view definitions.

5.1 Modeling Conceptual Views

The modeling of XML conceptual views can be done using UML, plus a set of stereotypes [38] and newly introduced conceptual operators [37]. In addition, to make view constraints more explicit and visible, we use OMG's Object Constraint Language (OCL) [33].

As our conceptual view mechanism is defined at a higher-level of abstraction, we can provide an explicit view constraint specification model, as most high-level OOCM languages (such as UML, XSemantic nets, E-ER) provide some form of constraint specification. In the case of XSemantic nets, constraints are provided as part of the model elements [20, 39]. In the next section, we will look at specifying constraints using UML/OCL.

5.2 Modeling Conceptual View Constraints

In UML, OCL, which is now a part of the UML 2.0 standard [34], can support unambiguous constraints specifications for UML models. In our conceptual view model, we incorporate OCL (in addition to built-in UML constraints features) as our view constraint specification language to explicitly state view constraints. It should be noted that, we do not use OCL to *define* views, rather state additional constraints using OCL. OCL supports defining *derived* classes [33, 50], which is close to a view concept [7]. It is of the form of;

```
context: <derived-class-name>::<new-attribute-name>: Type
derive: <source-stored-class>.
        [some expression representing the derivation rule]
          / [<source-stored-class-attribute>
```

To define our conceptual views, we show view classes visually, with the `<<view>>` stereotypes and the relationship between the stored class and the view as `<<construct>>` stereotype. Therefore, we do not require non-visual OCL view specification as shown above, but can be used to show some of the derivations rule for the attributes and/or operations to make the view definition more explicit and precise. It also supports specifying derived values and attributes in already existing views (and stored classes) and specified in the form of;

```
context Typename::assocRoleName: Type
derive: -- some expression representing the derivation rule
```

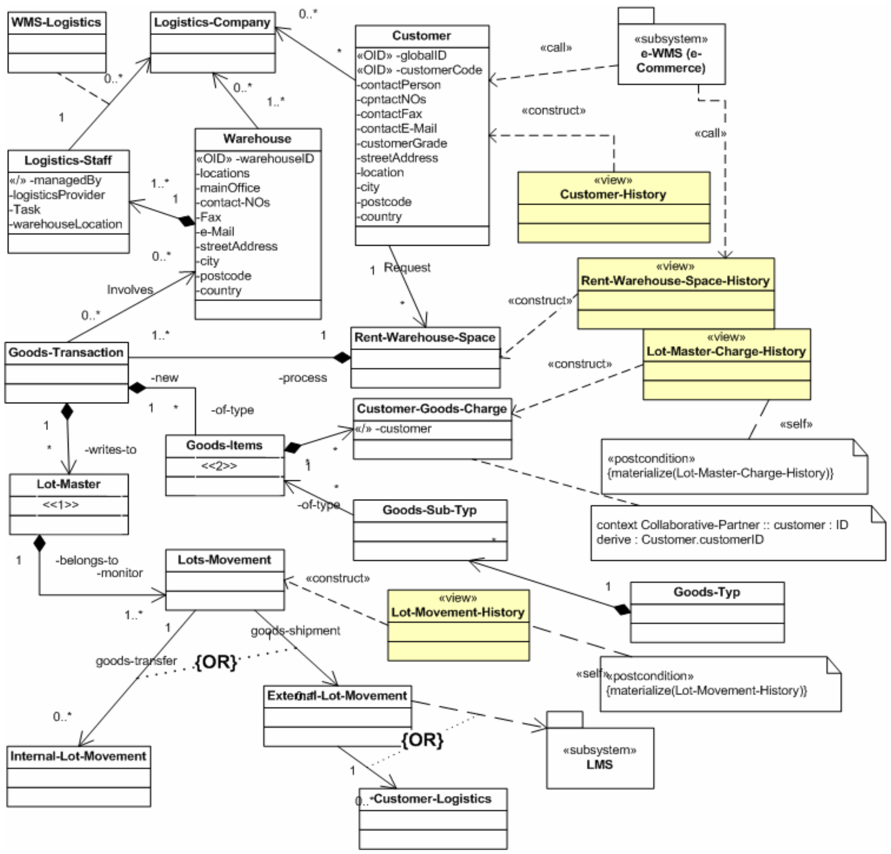


Fig. 4. e-Sol, core data stores, relationships and constraints

In addition, further constraints can be defined for conceptual views including; (1) domain constraints (range of values, min, max, pattern etc), (2) constructional contents (set, sequence, bag, ordered-set), (3) ordering (4) explicit homogenous composition/heterogeneous compositions, (5) adhesion and/or dependencies (6) exclusive disjunction and many more. Specifying these constraints using OCL expression in conceptual views are similar to that of stored domain objects.

Example 6: In the case of conceptual view "`Income`" (Fig. 3), the following OCL statements hold true;

```
context Income :: Staff : ID          context Income :: totalSalary : Real
derive : Staff.staffID                derive : totalSalary =
                                          (self.baseSalary - self.tax)
context Income :: benefits : Real        + benefits -
derive : Benefit-Pkg.totalBenefits        self.totalDeductions

context Income :: baseSalary : Real
derive : Salary-Pkg.baseSalary
```

Example 7: In the case of conceptual view "`Warehouse-Manager`" (Fig. 3), we indicate the unique `staffID` by the following OCL expression;

```
context Staff
inv : self->isUnique(self.staffID)
```

Example 8: In the case of conceptual views "`Warehouse-Manager`" and "`Warehouse-Staff`", in the context of "`Staff`" (Fig. 3), we indicate the adhesion relationship between them using the following OCL statements given below.

```
context Warehouse-Staff :: managedBy : ID
derive : Warehouse-Manager.staffID

context Warehouse-Manager
inv: self.responsibleFor := Set(Warehouse-Staff.staffID)

context ManageStaff
inv : Warehouse-Staff->managedBy (Warehouse-Manager.staffID)
```

Example 9: In the case of conceptual views "`Lot-Movement`" (Fig. 4), the exclusive disjunction between `Internal-Lot-Movement` (stored goods change owners) and `External-Lot-Movement` (goods shipped outside the warehouse) can be show via the OCL statement "OR" between the relationships as shown in Fig. 4.

Example 10: Similarly, the exclusive disjunction between conceptual views `Customer-Logistics` (where customers provide/use their own logistics service provider to move goods out of the warehouses) and `LMS` (where customers utilize the warehouses' own logistics services), can be show via the OCL "OR" statement (Fig. 4).

6 XML Schema Views

An XML conceptual view can be transformed/mapped into a corresponding XML schema view (in XML Schema language) using the techniques we developed in [20, 21, 53, 54]. Table 1 provides a very brief overview of our transformation rule mechanism. Due to space limitation, we refer readers to [20, 21, 38, 39, 53, 54] for detailed descriptions of mapping Object-Oriented conceptual models to XML Schema.

Formally, let $\aleph_s^c$ denote the transformation mechanism which can translate a set of objects, their relationships and constraints into a set of simpleType/complextType definitions for XML elements/attributes and associated element/attribute constraints in the XML Schema language.

***Definition 3*: An XML schema view V^s** is a triple $V^s = (V^s_{name}, V^s_{simpleType}, V^s_{complexType}, V^s_{constraint})$, where V^s_{name} is the name of the XML schema view V^s, $V^c_{simpleType}$, $V^s_{complexType}$ are simple and complex type definitions for XML elements/attributes, and

$V^s_{constraint}$ is a set of constraints upon the defined XML elements/attributes. Here, $V^c_{simpleType}$, $V^s_{complexType}$, and $V^s_{constraint}$ are expressed in the XML Schema Language, and V^s_{name} is also the name of the resulting XML schema file, i.e., a valid W3C XML document name.

Definition 4: Given an XML conceptual view $V^c = (V^c_{name}, V^c_{obj}, V^c_{rel}, V^c_{constraint})$, $V^s = (V^s_{name}, V^s_{simpleType}, V^s_{complexType}, V^s_{constraint})$ is a **valid XML schema view of V^c**, if and only if V^s is transformed from V^c by $\aleph^c_s$. That is, $\aleph^c_s : V^c \rightarrow V^s$.

To get the flavor of such a transformation, in the following, we exemplify how the OCL constraints at the conceptual level are mapped to the XML schema level.

Table 1. Transformation rules from conceptual view to schema view

Conceptual view (in UML)	Schema view (in XML Schema)
View class and View class hierarchy (View-of-view)	Complex type construction (`xsd:complexType`) and complex type/element hierarchies
View class constraints (e.g. Ordering, homogenous composition etc.	Built-in XML Schema constraints.
View attributes	XML Schema simple types (integer, float, string, date, time etc.)
View attribute constraints (e.g. Object Identifier (OID)	XML Schema constructs using "facet" and XML Schema specific element/attribute constraints
Attribute grouping (constructional contents such as set, bag)	XML Schema list (`NMTOKENS`, `IDREFS`, & `ENTITIES`), or new list types using `<xsd:list>` construct and union by using `<xsd:union>` construct.
Structural Relationships (IS-A, composition, association)	`<xsd:sequence>`, `<xsd:choice>` & `<xsd:all>` with `<xsd:extension>` / `<xsd:restriction>` and `ID`, `IDREF`/`IDREFS` or `KEY` and `KEYREF` constructs
Domain constraints	"facets" mechanism to create new types, apply restriction etc.
Uniqueness Constraint	XML Schema `<xsd:unique>` construct
Cardinality Constraint	XML Schema `maxOccurs`/`minOccurs` construct
Referential Constraint	XML Schema ID, `IDREF`/`IDREFS` or `KEY` and `KEYREF` constructs.
Stereotype <<view>>	Complex type construction (`xsd:complexType`) and complex type/element hierarchies.
Stereotype <<OID>>	Combination of XML Schema simple types ID, `IDREF`/`IDREFS` or `KEY` and `KEYREF` constructs and XML Schema unique constraint.

Example 11: The ordered/unordered compositions (e.g. in Fig. 4, "`Lot-Master`" & "`Goods-Items`"), shown using the stereotype (`<<1>>, <<2>>,`…), are mapped using the `<xs:sequence>` construct; (a) at the conceptual view attribute level and (b) at the conceptual view class level. This is shown below in code listing 1.

Example 12: As shown in Example 9, the exclusive disjunction constraint (Fig. 4) between "`Internal-Lot-Movement`" and "`External-Lot-Movement`" can be mapped XML Schema using the `<xs:choice>` schema construct, as shown below in code listing 2.

```
<!-- additional nesting -->
<xs:complexType  name="Composite_Object/attrbute">
    <xs:sequence>
        <xs:element name="Componant-A/attribute-a"
                    type="xs:OID"/>
          <!-- additional nesting -->
          <!-- additional nesting -->
    </xs:sequence>
</xs:complexType>
<!-- additional nesting -->
```

Code Listing 1: Code sample for example 11

```
<!-- additional nesting -->
<xs:element name="Lot-Movement">
   <xs:complexType>
      <xs:complexContent>
        <xs:extension base="LotMovementType">
        <!-- additional nesting -->
          <xs:choice>
            <xs:element name=
               "Internal-Lot-Movement"
                  type="InternalLotMovementType"/>
            <!-- additional nesting -->
            <xs:element name=
               "External-Lot-Movement"
                  type="ExternalLotMovementType"/>
            <!-- additional nesting -->
          </xs:choice>
        <!-- additional nesting -->
        </xs:extension>
       </xs:complexContent>
   </xs:complexType>
</xs:element>
<!-- additional nesting -->
```

Code Listing 2: Code sample for example 12

Example 13: The unique constraint (the `<<OID>>`) in `Staff` can be mapped to the view schema as shown in the code fragment below (Fig. 3 & 4).

```
<!-- additional nesting -->
 <xs:complexType name="staffType">
       <xs:sequence>
           <xs:element name="staffID" type="OIDType"/>
           <!-- additional nesting -->
           <xs:element name="managedBy">
               <xs:key name="manager">
                   <xs:selector/>
                   <xs:field xpath="staffID"/>
               </xs:key>
           </xs:element>
       </xs:sequence>
   </xs:complexType>
   <!-- additional nesting -->
   <!-- additional nesting -->
   <xs:complexType name="OIDType">
       <xs:sequence>
           <xs:element name="ID">
               <xs:unique name="OID">
                   <xs:selector xpath="OIDType"/>
                   <xs:field xpath="ID"/>
               </xs:unique>
           </xs:element>
       </xs:sequence>
   </xs:complexType>
 <!-- additional nesting -->
```

7 XML Instance Views

An XML instance view is an instantiated imaginary XML document which conforms to the XML schema view defined at the schema level. An instance view can be used for many purposes such as database views, materialised semantic Web-views, etc., and can be generated from simple projection of selected document tags to provide dynamic window into complex heterogenous documents. Based on how these documents are constructed/behave, we classify XML instance views into three categories

[35], namely, *derived instance views, constructed instance views,* and *triggered instance views.*

XML instance views can be constructed via a native XML query language (e.g. XQuery [48], SQL 2003 [6]) or some specific algorithms such as Ontology Extraction Methodology (OEM) [52] (the MOVE [51] system), which can be achieved by the transformation of conceptual operators λ defined at the conceptual level [37] into a language-specific query fragment λ_{Query}, say FLWOR expressions in XQuery etc. In a related work [52], the authors have shown how conceptual operators can be transformed and mapped to query algorithms in the MOVE system.

In this paper, we briefly discuss XQuery as the document view construct language. Here, we choose XQuery as document view constructor as it is gaining momentum as the language of choice for XML databases and repositories. It is a functional language [9] and its functionalities are comparable to early/mid SQL standards (in the relational model). In addition it is well-suited for our purpose better than other XML query languages (such as XPath or XSLT) as; (a) XQuery is easy to read and write in comparison XPath and XSLT, (b) in direct contrast to XQuery's powerful For-Let-Where-Order-Return (FLWOR) expression, XPath/XSLT are purely presentation oriented, (c) XQuery provides User-Defined Functions (UDF) and variable binding and (d) XQuery support XML Schema. Table 2 below shows a brief summary of such mappings (including some built-in XQuery operators).

Table 2. Transformation of conceptual operators to instance view query expressions

Conceptual Operator	XQuery Equivalent	Illustrative Example Code
Union	Built-in operator (`union` or `|`)	`let $a := document ("d1.xml")` `let $b := document ("d2.xml")` `return <union-result> {$a | $b}</union-result>`
Intersection	Built-in operator `intersect`	`let $a := document ("d1.xml")` `let $b := document ("d2.xml")` `return <intersect-result>` `{$a intersect $b}</intersect-result>`
Difference	Built-in operator `except`	`let $a := document ("d1.xml")` `let $b := document ("d2.xml")` `return <difference-result>` `{$a except $b}</difference-result>`
Cartesian Product	In any FLWOR expression, when more than one variable is bound to a `FOR` clause (no where clause)	`for $a in document("d1.xml"),` `$b in document("d2.xml")` `return <simple-CP-example> <a>{$a}</a>` `<b>{$b}</b>` `</simple-CP-example>`
Join		`for $a in document("d1.xml"),` `$b in document("d2.xml")` `where $a/ID = $b/ID` `return <simple-JOIN-example> <d1>{$a}</d1>` `<d2>{$b}</d2> </simple-JOIN-example>`
Project		Any valid FLWOR expression. Simplest PROJECT expression is `doc("d1.xml" )`
Select, Rename, Restruct(ure)	User defined functions and/or Built-in operates	Any valid FLWOR expression, with selective/conditional clause

Formally, we can have the following definition for XML instance views.

Definition 5: Given an XML schema view V^s and a set of XML source documents *S*, V^i is called a **valid XML instance view of V^s over *S*** if and only if V^i is a well-formed XML document, extracted from *S* by certain query operators in λ_{Query}, and conforming to the XML schema V^s.

Example 14: As described in Examples 1, 7 and 8, the conceptual view operators of the view "`Warehouse-Manager`" can be mapped to the document view construct (XQuery expression) as shown below in the code segment.

```
for       $role in document ("staff.xml")//Role
where     $role = "Manager"
return <Warehouse-Manager> {$role} </Warehouse-Manager>
```

Example 15: If a new domain requirement exists to add new conceptual view Management-Memo send to all "`Warehouse-Manager`", we can do that using Cartesian Product conceptual operator. Here in document level, we can map that conceptual operator to the following XQuery expression;

```
for       $a in document("Warehouse-Manager.xml")//Role,
          $b in document("Management-Memo.xml")//Message
return <send-memo> <W-Mgr>{$a}</W-Mgr>
             <Mgr-Memo>{$b}</Mgr-Memo></send-memo>
```

8 Conclusion and Future Work

In this paper, we presented a view model with conceptual and schema extensions. We also presented a view constraint specification model (OCL based) and conceptual operators. We showed how conceptual views are mapped to its schema and document level equivalent, with illustrated case study examples.

For future work, some further issues deserve investigation. First, the investigation of a formal mapping approach to conceptual view constraints, to automate the view constraint model transformation between the conceptual and schema level constraints. Second, the automation of the mapping process between conceptual operators to various query language expressions (such as XQuery). Third, is the investigation into dynamic perspectives of the conceptual view formalism that can be applied to traditional data, Semantic Web and web service domains.

References

1. S. Abiteboul, "On Views and XML," Proc. of the 18th ACM PODS '99, USA, 1999.
2. S. Abiteboul and A. Bonner, "Objects and Views," Proc. of the Int. Conf. on Management of Data (ACM SIGMOD '91), 1991.
3. S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge, "Views for Semistructured Data," Workshop on Management of Semistructured Data, USA, 1997.
4. S. Abiteboul, et al., "The Lorel Query Language for Semistructured Data," *Int. Journal on Digital Libraries*, vol. 1, pp. 68-88, 1997.
5. AIIM, "The ECM Association (http://www.aiim.org/index.asp)," AIIM, 2005.
6. ANSI and ISO, "ANSI - SQL 2003," ANSI / ISO 2003.

7. H. Balsters, "Modelling Database Views with Derived Classes in the UML/OCL-framework," The UML '03, USA, 2003.
8. R. G. G. Cattell, et al., "The Object Data Standard: ODMG 3.0," Morgan Kaufmann, 2000, pp. 300.
9. D. D. Chamberlin and H. Katz, *XQuery from the experts : a guide to the W3C XML query language*. Boston: Addison-Wesley, 2003.
10. E. Chang and T. S. Dillon, "Integration of User Interfaces with Application Software and Databases Through the Use of Perspectives," 1st Int. Conf. on ORM '94, Australia, 1994.
11. E. Chang, et al., "A Virtual Logistics Network and an e-Hub as a Competitive Approach for Small to Medium Size Companies," Int. Human.Society@Internet Conf., Korea, 2003.
12. E. Chang, et al., "Virtual Collaborative Logistics and B2B e-Commerce," e-Business Conf., Duxon Wellington, NZ, 2001.
13. E. J. Chang, "Object Oriented User Interface Design and Usability Evaluation," in *Department of CS-CE*: La Trobe University, Melbourne, Australia, 1996.
14. Y. B. Chen, T. W. Ling, and M. L. Lee, "Designing Valid XML Views," Proc. of the 21st Int. Conf. on Conceptual Modeling (ER '02), Tampere, Finland, 2002.
15. S. Cluet, P. Veltri, and D. Vodislav, "Views in a Large Scale XML Repository," Proc. of the 27th VLDB Conf. (VLDB '01), Roma, Italy, 2001.
16. C. J. Date, *An introduction to database systems*, 8th ed. New York: Pearson/Addison Wesley, 2003.
17. T. S. Dillon and P. L. Tan, *Object-Oriented Conceptual Modeling*: Prentice Hall, Australia, 1993.
18. J. H. Doorn, L. C. Rivero, and (eds), *Database Integrity: Challenges & Solutions*: Idea Group Publishing, Hershey, PA, 2002.
19. R. Elmasri and S. Navathe, *Fundamentals of database systems*, 4th ed. New York: Pearson/Addison Wesley, 2004.
20. L. Feng, E. Chang, and T. S. Dillon, "A Semantic Network-based Design Methodology for XML Documents," *ACM Trans. on IS (TOIS)*, vol. 20, No 4, pp. 390 - 421, 2002.
21. L. Feng, E. Chang, and T. S. Dillon, "Schemata Transformation of Object-Oriented Conceptual Models to XML," *Int. J. of Comp. Sys. Sci. & Eng.*, vol. 18(1), pp. 45-60, 2003.
22. V. Gopalkrishnan, Q. Li, and K. Karlapalem, "Star/Snow-flake Schema Driven Object-Relational Data Warehouse Design and Query Processing Strategies," 1st First Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK '99), Florence Italy, 1999.
23. I. Graham, A. C. Wills, and A. J. O'Callaghan, *Object-oriented methods : principles & practice*, 3rd ed. Harlow: Addison-Wesley, 2001.
24. A. Gupta, I. S. Mumick, and (eds), *Materialized views: techniques, implementations, and applications*: MIT Press, 1999.
25. ITEC, "iPower Logistics (http://www.logistics.cbs.curtin.edu.au/)," 2002.
26. W. Kim, "Research Directions in Object-Oriented Database Systems," Proc. of the 19th ACM Sym. on Principles of Database Systems, Nashville, Tennessee, USA, 1990.
27. W. Kim and W. Kelly, "Chapter 6: On View Support in Object-Oriented Database Systems," in *Modern Database Systems*: Addison-Wesley Publishing Company, 1995, pp. 108-129.
28. H. Liefke and S. Davidson, "View Maintenance for Hierarchical Semistructured," Proc. of the Second Int. Conf. on DaWak '00, London, UK, 2000.
29. Lucie-Xyleme, "Xyleme: A Dynamic Warehouse for XML Data of the Web," Int. Database Engineering & Applications Symposium (IDEAS '01), Grenoble, France, 2001.
30. B. Ludaescher, et al., "View Definition and DTD Inference for XML," Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, 1999.

31. M. K. Mohania, K. Karlapalem, and Y. Kambayashi, "Data Warehouse Design and Maintenance through View Normalization," 10th Int. Conf. on DEXA '99, Italy, 1999.
32. V. Nassis, et al., "Conceptual and Systematic Design Approach for XML Document Warehouses," *Int. J. of Data Warehousing and Mining*, vol. 1, No 3, 2005.
33. OMG-OCL, "UML 2.0 OCL Final Adopted specification (http://www.omg.org/cgi-bin/doc?ptc/2003-10-14)," OMG, 2003.
34. OMG-UML™, "UML 2.0 (http://www.uml.org/#UML2.0)," 2003.
35. R.Rajugan, E. Chang, T. S. Dillon, and L. Feng, "XML Views: Part 1," 14th Int. Conf. on Database and Expert Systems Applications (DEXA '03), Prague, Czech Republic, 2003.
36. R.Rajugan, et al., "Engineering XML Solutions Using Views," The 5th Int. Conf. on Computer and Information Technology (CIT '05), Shanghai, China, 2005.
37. R.Rajugan, E. Chang, T. S. Dillon, and L. Feng, "A Layered View Model for XML Repositories & XML Data Warehouses," The 5th Int. Conf. on CIT '05, China, 2005.
38. R.Rajugan, E. Chang, T. S. Dillon, and L. Feng, "XML Views, Part III: Modeling XML Conceptual Views Using UML," 7th Int. Conf. on Ent. IS (ICEIS '05), Miami, USA, 2005.
39. R.Rajugan, et al., "Semantic Modelling of e-Solutions Using a View Formalism with Conceptual & Logical Extensions," 3rd Int. IEEE Conf. on INDIN '05, Australia, 2005.
40. M. Rafanelli and (ed), "Multidimensional Databases: Problems and Solutions," Idea Group Inc., 2003, pp. 474.
41. R. Steele, W. Gardner, R.Rajugan, and T. S. Dillon, "A Design Methodology for User Access Control (UAC) Middleware," Proc. of the IEEE Int. Conf. on EEE '05, 2005.
42. R. Volz, D. Oberle, and R. Studer, "Views for light-weight Web ontologies," Proc. of the ACM Symposium on Applied Computing (SAC '03), USA, 2003.
43. W3C-RDF, "Resource Description Framework (RDF), (http://www.w3.org/RDF/)," 3 ed: The World Wide Web Consortium (W3C), 2004.
44. W3C-SW, "(http://www.w3.org/2001/sw/)," W3C, 2005.
45. W3C-SW, "Semantic Web, (http://www.w3.org/2001/sw/)," W3C, 2005.
46. W3C-WS, "Web Services Activity, (http://www.w3.org/2002/ws/)," W3C, 2002.
47. W3C-XML, "Extensible Markup Language (XML) 1.0, (http://www.w3.org/XML/)," 3 ed: The World Wide Web Consortium (W3C), 2004.
48. W3C-XQuery, "XQuery 1.0: An XML Query Language," in *XML Query Language (XQuery)*: The World Wide Web Consortium (W3C), 2004.
49. W3C-XSD, "XML Schema," vol. 2004, 2 ed: W3C, 2004.
50. J. B. Warmer and A. G. Kleppe, *The object constraint language : getting your models ready for MDA*, 2nd ed. Boston, MA: Addison-Wesley, 2003.
51. C. Wouters, T. S. Dillon, J. W. Rahayu, E. Chang, and R. Meersman, "Ontologies on the MOVE," 9th Int. Conf. on Db. Sys. for Adv. Apps. (DASFAA '04), Korea, 2004.
52. C. Wouters, Rajugan R., et al., "Ontology Extraction Using Views for Semantic Web," in *Web Semantics and Ontology*, USA: Idea Group Publishing, 2005.
53. R. Xiaou, et al., "Mapping Object Relationships into XML Schema," Proc. of OOPSLA Workshop on Objects, XML and Databases, 2001.
54. R. Xiaou, et al., "Modeling and Transformation of Object-Oriented Conceptual Models into XML Schema," 12th Int. Conf. on DEXA '01 2001, 2001.
55. Y. Zhuge and H. Garcia-Molina, "Graph structured Views and Incremental Maintenance," Proceeding of the 14th IEEE Conf. on Data Engineering (ICDE '98), USA, 1998.

Modeling Group-Based Education
A Proposal for a Meta-model

Manuel Caeiro-Rodríguez, Martín Llamas-Nistal, and Luis Anido-Rifón

University of Vigo, Department of Telematic Engineering,
C/ Maxwell S/N E-36310, Spain
{Manuel.Caeiro, Martin.Llamas, Luis.Anido}@det.uvigo.es
http://www-gist.det.uvigo.es/~mcaeiro

Abstract. Currently, one of the most important application domains for Software Engineering is distance education and e-learning. There exist some standards/specifications that deal with the definition of both the data models and services needed in this environment. Among them, Educational Modelling Languages (EMLs) define the content and process within a unit of instruction. Nevertheless, current EMLs do not completely cover the whole educational scenarios. This paper uses workflow management techniques to present a conceptual meta-model for group-based educational design. This meta-model includes all required elements, flows and policies needed in this domain. Lacks in current Educational Modelling Languages will be identified to model group-based education.

1 Introduction

Computer-supported learning has evolved both with technological and instructional developments. In the one side, during the last years, there has been a great evolution in the field of information sciences. The development of computer networks and the appearance of the Web have promoted the creation of interactive and group-based systems that free humans from time and place constraints. In the other side, during the last decades, instructional and pedagogical initiatives have focused their efforts on collaborative educational scenarios, trying to enhance human learning processes and capabilities [1]. Group-based education gathers a broad range of educational practices and approaches: constructivism, project-based, inquire-based, discussion-based, etc.

Accordingly to these evolutions, Educational Modeling Languages (EMLs) [2] were proposed some years ago to enable the modeling of units of instruction (e.g. a lesson, a course). The purpose of this work is to contribute to the development of an EML to support the description of group-based educational scenarios and practices. It requires a meta-model that enables the characterization of the different ways of interaction and collaboration among learners and academic staff (teachers, tutors, etc.). Eventually, the achievement of a standardized EML will enable the development of tools both for the design and enactment of units of instruction, improving the labor of instructional designers, pedagogic experts, etc., and the educational experiences of teachers and learners.

2 Educational Modeling Languages

The CEN/ISSS WS-LT [3] survey of Educational Modeling Languages proposed the following EML definition: *An EML is a semantic information model and binding, describing the content and process within a 'unit of learning' from a pedagogical perspective in order to support reuse and interoperability.* Therefore, the purpose of EMLs, of which IMS Learning Design (LD) is the most outstanding proposal [4], is to support the description of diverse teaching-learning experiences (i.e. learning designs), embodying different kind of tasks, processes, persons, tools, and contexts. Later, these designs may be used by an engine to enable, control, and support the intended learning experiences through the design enactment.

For example, the LD proposal is a meta-language that allows to codify units-of-study (e.g. courses, course components, programs of study), associating each element of content (e.g. texts, tasks, tests, assignments) with information describing its instructional use (e.g., roles, relations, interactions, and activities of students and teachers). LD is mainly concerned with supporting the coordination issues that take place in education: how to control the order of specific activities to be performed by humans and applications, and the use of resources. Basically, it is a declarative language, but it also has an important transactional component. The LD specification has an XML binding to compose learning designs.

In group-based settings, EMLs are related with CSCL Scripts [5]: A script is a story or scenario that the students and tutors have to play as actors play a movie script. In group-based educational scenarios, these scripts are proposed as activity programs that aim to facilitate collaborative learning by specifying activities in collaborative settings, eventually sequencing these activities and assigning them to learners. But, currently there is no language or EML-based proposal that completely satisfies the design requirements involved in CSCL Scripts. As a consequence, our purpose is to contribute to the development of EMLs proposing a meta-model that supports such issues.

We want to emphasize that both EMLs and CSCL Scripts involve workflow and groupware issues. It is necessary to consider: (i) the tasks that have to be performed by each participant, the control flow, the data flow, etc.; and (ii) the way how learners and academic staff interact among them. But, it is necessary to have in mind the idea that educational scenarios are different from the ones in industry or software engineering. The goal is not task achievement efficiency or effectiveness, but apprenticeship, i.e. what the student has learnt from performing a set of tasks.

3 Group-Based Educational Design

Group-based education requires that learners and academic staff (e.g. tutors, professors, etc.) participate in a great variety of experiences (problem-solving, group discussions, games, simulations, projects, etc.), where people, documents, and tools are arranged in different ways. The variety of ways can be considered in the nature of tasks [6] (gathering and distributing information; creating documents; discussing and commenting around productions; planning activities), the interaction among participants (free communication, document sharing, etc.), the number of participants, the work arrangement, etc. Some examples of tasks in group-based experiences are [5, 7]:

– *Individual.* Many times learning involves the work of individual persons (learners or academic staff) working towards particular goals (i.e. a learner has to read a text and produce some kind of summary).
– *Collaborative.* Participants work together on the same group task, either synchronously or in frequent asynchronous interaction. Learners and staff share a common working space, documents, tools, and communicating facilities and try to achieve a common goal.
– *Cooperative.* The group educational goal is spitted in sub-goals. Learners are assigned to solve sub-tasks individually or in small groups. Then, once the sub-goals have been achieved, a collaborative task enables that all the participants share their results and discuss about the experiences.
– *Collective.* There is no group goal, but individual educational goals. Each learner works alone or in small groups on his own task, but shares results and problems with the others, and therefore shares inspiration, exchanges help and so forth. The different working groups can take advantage of the common information.

In order to propose a supporting model for our intended EML, we have classified the elements and behaviors involved in the tasks, interactions, and arrangements, which can appear in a group-based educational scenario. Our analysis is based on the identification of perspectives, and for each perspective, the patterns that may be involved in it. As a result, we obtain a set of criterions that drive the development of our model. We consider a perspective (also aspect or dimension) as a feature that involves a certain purpose and that can be analyzed separately from other perspectives. Some perspectives have been identified during the last years in related domains: workflow [8], groupware [9], human-computer interaction [10], etc. A pattern is an abstraction that is frequently repeated. It can be considered as a general solution to a common problem [11]. We consider patterns as the common forms that need to be described by an EML.

We have tried to find the key perspectives and patterns involved in group-based educational scenarios. The research involved the mentioned works (workflow, groupware, human-computer interaction, etc.) and CSCL design [12-14]. In table 1 we present the identified perspectives and some of the involved patterns (organized by groups). In the next items we provide some explanations:

– *Causal.* It answers the question about *why* to perform a certain unit of instruction. It gives educational information about the learning goal or goals to be attained, the pedagogical approach, the background required, etc. This perspective is not directly related with the modeling of the unit of instruction, but it is important to facilitate their search, classification, etc.

Table 1. Perspectives and pattern groups for group-based educational design

Perspective	Pattern Groups
Causal	Educational info: Educational Goals, Pre-requirements, etc.
	Learner info: Preferences, Background, Portfolio, etc.
Functional	Composition patterns: Collaborative, Cooperative, Collective, etc.
	Constraint patterns: Pre/Post-conditions, Inter-dependencies, etc.
Behavioral	Basic control: Sequence, Parallel Split, Synchronization, etc.
	Advanced branching and synchronization: Multi-choice, etc.
	Structural: Arbitrary Cycles, Implicit Termination, etc.
	Involving multiple instances: Without Synchronization, etc.
	State-based: Deferred Choice, Milestone, etc.
Temporal	Synchronization: A before B, A starts B, A finishes B, etc.
	Scheduling: Deadline, Start Point, etc.
	Allocation: Maximum, Minimum, Average Execution Time, etc.
Informational	Data visibility: Task Data, Block Data, Scope Data, etc.
	Data interaction: Task to Task, to Multiple Instance Task, etc.
	Data transfer: by Value, by Reference, Copy, etc.
	Data-based routing: Data Existence, Data Value, etc.
Operational	Tool configuration: Parameters, Model-based, Pattern-based, etc.
	Tool interaction: Push-oriented, Pull-oriented, Events, etc.
Authorization	Static (Access Control) Authorization, Obligation, etc.
	Dynamic (Floor control): Delegate, Revoke, Cancel, Request, etc.
Interaction	Mode: Synchronous, Asynchronous, Notification, etc.
	Media Type: Text, Images, Audio, Video, Gestural, etc.
	Policies: Master-Slave, Round Robin, Hot Seat, Free, etc.
Organizational	Participant grouping: Flat, Hierarchical, Constrained, etc.
	Participant relationships: delegation, priority, etc.
Resource	Resource assignment: Single Offer, Multiple Offer, Allocation, etc.
	Resource Info: Runtime Data, Portfolio, etc.
Awareness	Asynchronous: Focused, Filtered, Aggregation, Summary, etc.
	Synchronous: Tele-pointer, Presence Indicator, etc.

- *Functional.* It answers the question about *what* has to be done in each task of the unit of instruction. This perspective characterizes the tasks that have to be performed and how these tasks are decomposed into smaller units [7].
- *Behavioral.* It answers the question about *when* to perform a task in an unit of instruction. This perspective (also named as process or control flow) describes the execution ordering of tasks. Workflow patterns have being specified to evaluate the expressiveness of this perspective in workflow [15].
- *Temporal.* It answers the question about *when* to perform a task *in time.* It adds another dimension to the control flow of tasks. Without temporal constraints, a task is initiated when its preceding tasks have finished. Allen relationships [16] describe time constraints between two tasks.
- *Informational.* It answers the question about *what information is available* to perform a task in an unit of instruction. The informational perspective (also named as data flow) describes the information used, its flow and depen-

dencies among tasks. Artifacts (e.g. Learning Objects [22]) and information can flow between tasks in different ways [9, 17]. Synchronous data flow takes place when the sender task terminates and the receiver starts. Asynchronous data flow takes place during task execution, concerning copy or transfer.

- *Operational.* It answers the question about *what operations are available* to perform a task in an unit of instruction. It comprises the applications used in the activities. It describes the methods for accessing or invoking external applications (e.g. simulators, editors, communication and collaboration services). These can be very different in nature but from the process point of view the technical details of the applications are to be kept transparent.
- *Authorization.* It answers the question about *what access rights have users* to access objects and operations in the environment. This perspective enables to establish the limits of the education environment for each participant and group (e.g. public and private workspaces), to show certain artifacts, etc.
- *Interaction.* It answers the question about *how participants can interact* during communication and co-operation. Communication encompasses the process of transfer and exchange of information. It is a basic functionality involved in any collaborative situation. Typical communication tools are: e-mail, desktop conferencing systems, chat, whiteboard, etc. Co-operation is concerned on the access and change of a shared set of data. In co-operation, we group the set of interactions related to the storage and manipulation of shared artifacts. Examples of systems that provide these functionalities are shared editors, virtual whiteboards, shared repositories, etc. The interaction perspective is very important in collaborative educational settings to organize and coordinate the way in which learners and staff interact.
- *Organizational.* It answers the question about *what organizational structure is responsible* for performing an unit of instruction. The organizational perspective describes the structure of the participants and groups devoted to perform tasks. It is possible to consider educational scenarios where group structure can vary from task to task.
- *Resource.* It answers the question about *who is responsible of performing each task* of an unit of instruction. This perspective is concerned with the management of resources in the educational process. Resources can be human (e.g. a worker) or non-human (e.g. a room, a machine), although the main focus is on human resources. One of the main important points is how a resource is assigned to tasks and grouped into teams.
- *Awareness.* It answers the question about *what runtime information have to be presented to each participant* in an unit of instruction. Awareness refers to how is made visible or available to participants what the other participants are doing or have done. Awareness can be used for educational purposes in many ways. Usually, teachers need to obtain information about the actions of their learners. In order to give to the right participant the information and to avoid information overload, awareness should be focused, customized, and temporally constrained [8, 18].

In addition to these perspectives, there are two important points to consider in group-based educational scenarios. The first one is related to the structure of the scenarios. In the one side, some instructional approaches require very well defined roles, groups, tasks, environments, etc. In the other side, there are instructional approaches that consider open scenarios, including ill-defined tasks, unstructured collaborations, etc. Therefore it should be possible to design plans with flexible structures. The second one is related to enactment and execution flexibility. Many times educational plans have to be changed during their execution. Therefore, it should be possible to enable that some authorized participants (e.g. tutors) can introduce modifications or alter the design during its enactment.

4 A Meta-model Proposal for Group-Based Educational Design

In this section, we present the meta-model proposed to support the modeling of group-based units of instruction. This meta-model has an XML binding representation that enables such design. We have tried to support the perspectives and patterns introduced in the previous section. The proposals are based on workflow, groupware, and human-computer interaction results. Especially, we have taken key ideas from [9, 15, 17, 19, 20, 21].

4.1 Basic Elements

The proposed meta-model is object-based and state-oriented. All elements in the meta-model derive from the abstract Entity element [20], cf. figure 1. An Entity contains attributes, operations, events, and is linked to other entities via relationships. In addition, any Entity can be related with Rules that can be used to perform operations in the Entity when some conditions evaluate true. We have considered the following built-in elements in the Entity:

- *Attributes.* There are at least two attributes included in all the elements of the meta-model. (a) The state attribute is used to maintain the current state of the element. All the elements will be associated with a state-diagram model in order to support their enactment. The available operations and events will be restricted by its state [20]. (b) The meta-data attribute is used to maintain information describing the element (e.g. version, creation date, etc.). Some of the elements (Educational Scenario, Environment) may include educational meta-data definitions using the LOM standard [22]. This meta-data model enables the description of the educational purpose of the elements.
- *Operations.* They are used to change the state of the element. Built in operations are devoted to change the attributes (changeState, changeMeta-data), and to change the rules (addRule, deleteRule). The management of any element by an enactment engine has to be produced through calls to its operations. Therefore, it is possible to enable that some participant can perform some enactment control.
- *Events.* Events are generated when an operation is invoked and when the state of the element changes.

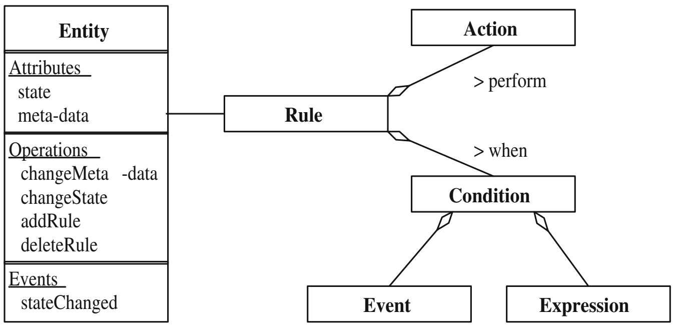


Fig. 1. The entity model. All the meta-model elements derive from this basic entity type

- *Rules.* Rules are included to provide control functionality. They present the following form: If Condition (event, expression) is TRUE then Action. This provides a generic mechanism to invoke operations over elements as needed. For example, it would be possible to specify that a participant has to be notified if a certain resource is modified.

4.2 Meta-model Structure and Description

In this section, we present the proposed meta-model. Figure 2 illustrates the core elements and the functional relationships. The most basic concept is the Educational Scenario (ES), which represents a unit of instruction at any aggregation degree. An ES is intended to perform a certain task (e.g. solve an exercise, read a document, support a learner working with a simulator), focused towards some educational objective (e.g. to get some expertise, some knowledge) and considering certain restrictions (e.g. time conditions).

The ES element is the aggregation point where all other elements are anchored. Each ES constitutes a context of elements not accessible from other ES. In this way it is possible to construct well-structured and separated educational spaces. An ES aggregates the following elements:

- *Educational Scenarios.* An ES may contain other ES devoted to breakdown a general goal into several sub-goals. All the ESs present the same organization. Relationships (Connectors and Flows) are considered to relate the elements of a Parent ES with the elements of a Child ES and the elements of Child ESs among them.
- *Roles.* The ES needs to consider some participants, at least one, to Perform the intended tasks. The number of participants is not restricted, and participants can be grouped in group hierarchies as desired. We have considered three special roles: (i) the Professor role is not mandatory, but if it is included the role can modify the prescribed ES as desired and control its enactment; (ii) the Responsible role is mandatory (but may be performed by several participants), and it has authorization to define new child ESs, providing

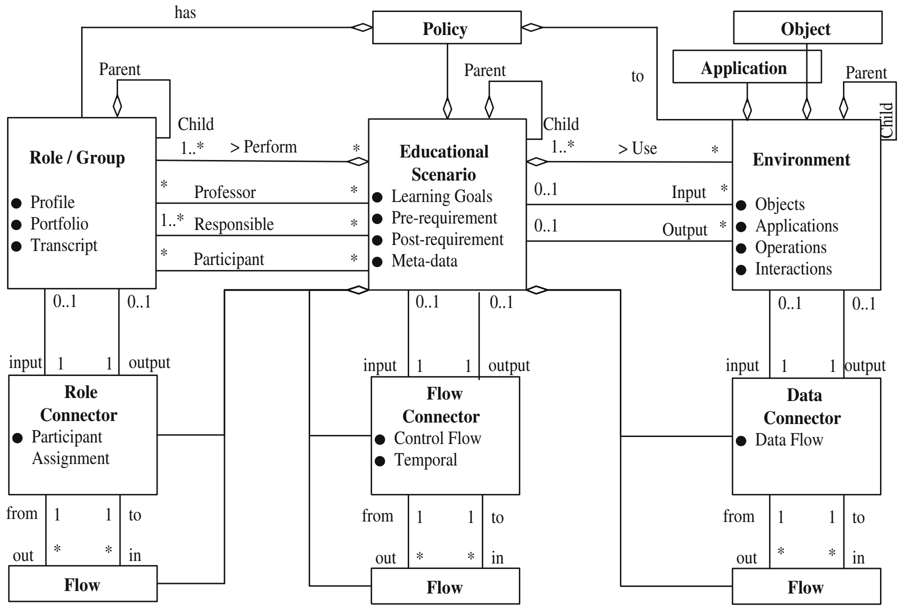


Fig. 2. The proposed meta-model

a breakdown of the current one; (iii) finally, there can be any number of Participants.

– *Environments.* The ES can consider some Environments containing the Resources to be Used to achieve the goal. Environments can be grouped hierarchical, but finally they will be composed by information documents (e.g. HTML pages, texts, figures, etc.) and tools (e.g. communication facilities, simulators, text processors, etc.). There may be two special kinds of Environments associated to an ES, namely Input and Output, that represent input and output parameters. These parameters work in association with Input and Output conditions to decide when a certain ES need to be enacted.
– *Connectors.* We consider three different kinds of connectors: Role Connector, Flow Connector, and Data Connector. Each one of them provides several mechanisms by which corresponding elements may be related: (i) the Flow Connector is used to relate the flow control among ESs (e.g. sequence, OR-join, AND-split, parameter conditions, temporal conditions, etc.); (ii) the Role Connector enables the transfer of participants through roles, supporting that a certain participant can perform different roles in different ESs (e.g. free, Responsible decides); (iii) the Data Connector supports the data flow among Environments and ESs (e.g. copy, transfer, synchronize, etc.). The Connectors use Rules to provide conditioned behaviors.
– *Flows.* They are simple links used to connect the different Connectors. Flows are associated with the Connectors, never with other elements (Roles, ES, and Environment).

– *Policies.* They are used to specify authorizations for the roles involved in an ES. Policies are used to define the permissions (rights, obligations, prohibitions, dispensations) that are provided to each role to use objects and applications of the ES.

Educational Scenario. An ES represents a unit of education at any level of granularity or specificity satisfying the functional perspective. Therefore, ESs play a central role, being both the concept for process structuring, and the corner stone on which reuse is promoted. We have taken this idea from the workflow domain, where it has been established that a system that does not include separate classes for atomic and composite tasks can more easily accommodate design-time and run-time evolution, making the language simpler and more efficient [19]. The approach is a mixture of block-based and flow-based. ESs are entities that are further described by decomposing each one. ESs are grouped via a hierarchy of ESs flows. Composite ES are either collections, which have an unordered list of sub-items, or networks where some sub-items are interconnected. Collections and networks thus refer to models with different degree of structure. The interconnection network is made up of two types of elements: Flow Connectors and Flows. Each ES may have two Flow Connectors to describe the conditions to initiate and finish in accordance with behavioral and temporal perspectives.

Each ES (either composite or atomic) can be instantiated multiple times. It involves learning goals, pre-requirements and post-requirements. Meta-data has already been considered in the Entity type and therefore may be present in all the elements of the meta-model. It is possible to specify a lower bound and an upper bound for the number of instances created after initiating the scenario. Moreover, it is possible to indicate that the ES has to be instantiate a variable number of times depending on a certain condition. For example, in many educational situations a tutor has to perform a task several times (e.g. to evaluate an exercise, to supervise a learner) that depends on a certain condition (the number of exercises and learners, respectively).

Roles. The participants that may be involved in an ES are persons, groups, and software agents. They are all considered in the model through the Role element. In general, EMLs distinguish between two generic roles (learner and staff), that can be further specialized to define new roles. We consider the definition of new roles through refinement and aggregation relationships (refinement relationships are not depicted in the figure to improve clarity):

– *Refinement relationships* enable to define more specialized behaviors. For example, it is possible to establish that there is a project leader in a group of learners. This specialized role may be used to provide higher permissions using policies.
– *Aggregation relationships* enable to compose groups, arranging other roles. There is no restriction to the aggregation level. Therefore it is possible to construct flat and hierarchical groups. This provides support for the organization perspective.

We have considered an original mechanism to facilitate the transfer of participants between roles. It is based on interconnections between roles using Role Connectors and Flows. These mechanisms enable that the participants can perform different roles at different ESs (e.g. it is possible to specify that a learner who is project leader in an ES should not be project leader in another ES). In this way we provide a solution for the resource assignment.

Learner roles have attributes that support the maintenance of information about the participants. Specifically, to support the resource perspective we consider: (i) a profile attribute, with information about the features of the learner (personal information, preferences, learning styles, etc.); (ii) a portfolio attribute, with information about tasks and works performed; and (iii) a transcript attribute, used to record information about the progress of the user in the current ES (e.g. project contributions, documents created, etc.). The information that constitutes the transcript of a role would be specified during the design of the ES, and it would be gathered automatically during runtime. In this way, a tutor could assess the performance of a learner in a course considering the transcripts of the roles played by the learner.

Environments. An ES may require several Environments containing the Objects and Applications to be Used by participants in order to Perform the intended task. Aggregation of Environments is included to support the definition of workspace hierarchies. Input and Output environments to ES are included to facilitate the establishment of conditions for its enactment. A object is any kind of document that may be used during an educational experience. Communication and collaboration perspectives are supported by the inclusion of appropriate applications.

Objects, Applications and Environments are Entities that may contain special attributes, operations, events, and rules. For educational design purposes and to support information and operational perspectives, it is very important to consider the interaction of Resources with Roles, with other Resources, and with ESs:

- *Environment to Role interaction.* By default, roles may access the Environments available in the current ES and in the parents ES. This general access rule may be constrained by appropriate Policies .
- *Environment to Environment interaction.* Data Connectors and Flows are used to connect the Environments and Resources. They enable the transfer of documents among Environments, the transfer of documents to applications, to relate documents to applications, etc.
- *Environment to ES interaction.* Environments and Resources should present clear state-diagrams to support the management of the interaction during enactment. It should be possible to configure an Application (e.g. the setup of a simulator), and to interact with it (e.g. obtain the qualification from an automatic assessment tool).

Control Flow. Flow Connectors and flows are used to specify input conditions, output conditions, and connections among ES supporting the control flow. A

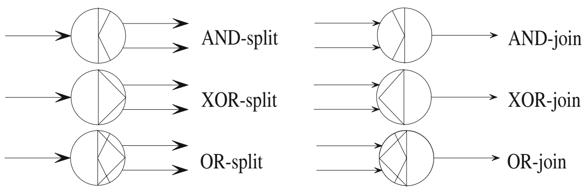


Fig. 3. Basic connection notation

Flow Connector represents an issue to resolve regarding the flow of tasks. Some decisions may be straightforward to prescribe during the early planning of a process, while others must be made by human actors during performance. The Responsible roles have to solve unresolved conditions.

Connectors (Flow, Role, and Data) are described using Rules and Expressions. The default Connector type is unspecified, which does not say anything about the relationship between inputs and outputs, it only considers a simple transfer. It is possible to specify conditions AND/XOR/OR both to join incomming flows and to split outgoing flows [21] (see Figure 3). Connectors can be linked among them to compose more sophisticated decisions. Timing and scheduling conditions may be included, providing Scheduled Tasks, Milestones, etc. It is also possible to specify that a Connector requires the intervention of participants using Policies (e.g. Tutor has Obligation to Resolve Connector).

Role Flow. We consider that an ES involves one or several Role elements. The roles involved in different ES may be different, and we propose a mechanism to transfer participants among them. This mechanism is based on Role Connectors and Flows. A Role Connector represents an issue to resolve regarding the flow of roles. Some decisions may be straightforward to prescribe during the early

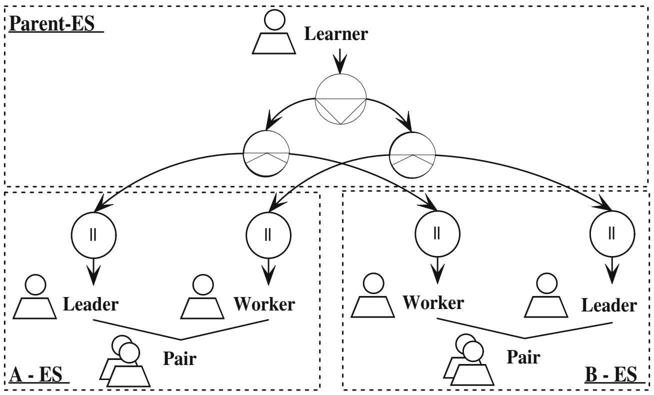


Fig. 4. Example of role transfer from one to two sub-ESs (A and B). Learners have to be grouped into pairs. It is required to maintain pair members, but they have to interchange the pair-role in each scenario (Leader and Worker).

planning of an unit of instruction, while others must be made by human actors (e.g. tutors) during performance. Role Connectors specify for each ES how its Roles are transferred to each of its sub-ES roles.

As it is explained in the previous section, Connectors are described using Rules and Expressions. Role Connectors can include conditions considering the Role Profile or Transcript. In this way, it is possible to select the learners that obtained the better and worse results in a previous ES to form pairs in a new ES. As in the previous case, it will be possible to specify Policies that require the intervention of participants to provide a solution (e.g. a responsible tutor decides how to group participants). Figure 4, shows and example of role transfer from a Parent-ES to two sub-ESs (A and B). Learners have to be assigned to pairs and it is required to maintain pair members, but they have to change the pair-role in each scenario (Leader and Worker). In the figure you can see that the leader in A-ES is the worker in the B-ES and vice versa.

Data Flow. Data flow deals with Resource transfer that may occur between different Environments. Environments are not transferred in association with the control flow, but they are moved independently. Similarly to the previous cases, we propose a mechanism based on Data Connectors and Flows. A Data Connector represents an issue to resolve regarding the flow of data among Environments. Data Connectors specify for each Environment how its Sub-environments and Resources are transferred to other Environments. In this way, we support the modeling of synchronous and asynchronous data flow. Data Connectors include the following operations: Copy, Transfer, Synchronize, Share, and Update. Rules are included to indicate when the corresponding operations have to be performed. In addition to transfer documents, this mechanism is also used to transfer documents to tools.

Policies. Policies are used to specify the authorization of participants to perform certain operations. In general, policies are the means to dynamically regulate the behavior of system components without changing code and without requiring the consent or cooperation of the components being governed. By changing policies, a system can be continuously adjusted to accommodate variations in external constraints and conditions. We consider policies as a mechanism to fit

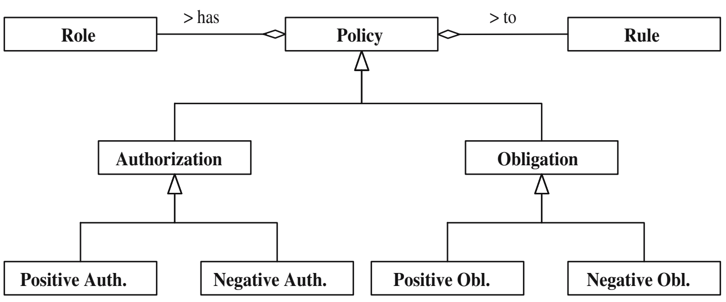


Fig. 5. Police structure

the enactment environment to the educational design. Figure 5 presents our proposed policy structure that is based on Rei [23]. Rei is a general purpose policy language that supports the description of security policies, management policies, and even conversation policies. The proposed structure includes the notions of rights, prohibitions, obligations, and dispensations. A Policy is defined relating a Role to a certain Rule. The Role is authorized/prohibited/obligated/dispensed of doing a certain Action if some conditions are true. Rei also offers other four speech acts that can be used to modify policies dynamically: delegate, revoke, cancel, and request.

4.3 Other Issues

In this section, we briefly introduce some issues that are considered in the proposed meta-model.

Awareness. Many educational and instructional strategies require that tutors receive appropriate information about the actions and progressions of their learners. We consider the definition of Awareness Specifications and Awareness Roles to support flexible, focused, and constrained awareness [18]. Awareness Specifications enable the description of what events have to be captured, filter such events, compose summaries, etc. These specifications can contain rules to constrain the monitored roles and establish temporal conditions. Awareness Roles enable to specify who should receive the awareness information.

Run-Time Control. In educational settings, plans use to change and to suffer variations. There are several practices and situations for this: (i) many times teachers do not completely describe a course plan; they consider some phases and provide some organization to produce an incomplete plan; (ii) during execution certain participants or resources are not available, and the proposed ESs have to be adapted; (iii) exceptions may appear due to learner giving up, group breaks; etc. To provide a solution two issues should be considered:

– *Model Evolution.* The meta-model proposed in this paper provides a great flexibility to define different instructional scenarios. At the same time the elements and structures are well separated and delimited. In this way, it is possible to modify certain ESs without affecting the others.
– *Enactment control by users.* Users can be given the capacity to change the enactment state. All the elements include the operations that are used by the enactment engine to control the execution automatically. Anyway, some users could be authorized to execute these operations.

4.4 A Brief Example

In this section we depict a small modelling example of a group-based learning practice that could be described using the introduced meta-model. The example is based on the subject *'Ingeniería del Software I'* (Software Engineering I), which is offered in the second year of the telecommunication engineering studies of the University of Vigo. In this subject, after some initial lectures, learners are proposed several practices that they have to carry out in pairs. We propose an ES to articulate each one of such practices, named as Software Engineering Practice. The Software Engineering Practice involves two main kinds of actors: Pair and Tutor. A pair is a group, made up of two actors: Leader and Worker. These actors are transferred from the parent ES roles (i.e.: the subject actors: Learners and Tutors) using Role Connectors and Flows. We assign Parent Learners to Leaders and Workers, and Parent Tutors to Tutors directly (cf. Figure 4).

In the subject model, we have considered a sequence of three Software Engineering Practice ESs. The Pair participants are maintained in each of the three Practice ESs, but changing the Learners role. Namely, the Leader in the first Practice will be the Worker in the second, and the Worker in the first will be the Leader in the second. The role of each Learner in the third Practice is decided according to the previous results. In relation with the children ESs, roles are transferred to each sub-ES directly, accordingly to its defined roles.

The Software Engineering Practice ES is decomposed in five sub-ESs connected by Flow Connectors and Flows. The first sub-ES is devoted to design a solution to the proposed practice. This task has to be done by the Leader role of the Pair, using a Design Environment to produce a Design document. When the first sub-ES is finished, two sub-ESs are initiated: Programming and Design Evaluation. The Design Evaluation sub-ES is assigned to the Tutor, who has to review the proposed design. As result a Design Evaluation document is provided. This review may be used by the Leader to modify and update the initial Design document in the next sub-ES: Design Revision. In parallel with these two sub-ESs, the Pair is devoted to program the intended design in the Programming sub-ES. This task may be assigned to the Worker only, but we prefer that both participants collaborate in the programming due to educational reasons. Furthermore, the Pair may decide to breakdown this task in several sub-tasks to organize and manage their programming activities (e.g. each member of the pair may program different functions and then they integrate it). This is enabled by allowing the Worker role to specify new sub-ESs. When the Pair finishes the Programming ES, the Program Evaluation sub-ES is activated. This sub-ES is assigned to the Tutor to review and evaluate the final Pair products Design and Program documents to produce a Program Evaluation.

5 Conclusions

We do believe that the EML approach to model and support the design of diverse education scenarios is going to play a key role in future e-learning systems. Our purpose is to contribute to the development of EMLs, focusing on their support for group-based education. We have considered the different forms that group-based education may present: individual, collaborative, cooperative, collective, etc., in order to provide a unique meta-model that is able to deal with all of them. Its core advantages are: (i) to provide a clear and simple way to articulate instruction; (ii) to enable different forms of structure; (iii) to support the interaction and coordination among participants, resources, and tasks in a extensible way; and (iv) to facilitate the flexibility of final models.

From the framework it is possible to check whether the current EMLs are able or not to properly deal with group-based education. The identified lacks will be overcome through extensions of the language both syntax and semantics. The next steps of our work are oriented towards this direction. The presented framework will be used to test IMS Learning Design and other EMLs. Then, the needed extensions will be proposed.

Acknowledgements

We want to thank Spanish *Ministerio de Educación y Ciencia* for its partial support to this work under grant *MetaLearn: methodologies, architectures and languages for E-learning adaptive services* (TIN2004-08367-C02-01).

References

1. Smith, P. L., Ragan, T. J.: Instructional Design, 3rd Edition, Wiley, Hossey-Bass Education (2005)
2. Koper, R.: Modeling units of study from a pedagogical perspective The pedagogical metamodel behind EML, Open University of the Netherlands (2001)
3. Rawlings, A., van Rosmalen, P., Koper, R., Rodrguez-Artacho, M., Lefrere, P.: Survery of Educational Modelling Languages (EMLs), Version 1, CEN/ISSS WS-LT (2002)
4. Koper, R., Olivier, B., Anderson, T.: IMS Learning Design Information Model, IMS Global Learning Consortium (2003)
5. Dillenbourg, P.: Over-scripting CSCL: The Risks of Blending Collaborative Learning and Instructional Design. In Kirschner, P. P. A. (Ed.): Three Worlds of CSCL. Can We Support CSCL, Open Universiteit Nederland (2002) 61–91
6. Strijbos, J. W., Kirschner, P., Martens, R. L.: Designing for Interaction: Six Steps to Designing Computer-supported Group-based Learning. Computers & Education, **42** (2004) 403–424
7. Schneider, D. K.: Conception and Implementation of Rich Pedagogical Scenarios through Collaborative Portal Sites. In The Future of Learning II, Sharing Representations and Flow in Collaborative Learning Environments, IOS Press (2004)
8. van der Aalst, W. M. P., Weske, M., Wirtz, G.: Advanced Topics in Workflow Management: Issues, Requirements, and Solutions. J. of Integrated Design, (2003)
9. Lonchamp, J.: Process Model Patterns for Collaborative Work. Proc. of the 15th IFIP World Computer Congress, Telecooperation Conference, Telecoop'98, Austria (1998)
10. Pinelle, D., Gutwin, C., Greenberg, S.: Tasks with the Mechanics of Collaboration. ACM Transactions on Computer-Human Interaction, **10(4)** (2003) 281–311
11. Alexander, C.: Pattern Language: Towns, Buildings, Construction, Oxford University Press, New York (1997)
12. Kirschner, P. A. (Ed.): Three Worlds of CSCL. Can We Support CSCL. OUNL, The Netherlands (2002)

13. Strijbos, J. W., Kirschner, P., Martens, R. L. (Eds.): What we Know about CSCL in Higher Education, Kluwer Academic Publishers, Dordrecht (2003)
14. Wasson, B., Ludvigsen, S., Hoppe, U. (Eds.): Designing for Change in Networked Learning Environments. Proceedings of the International Conference on Computer Support for Collaborative Learning. Kluwer Academic Publishers, Dordrecht (2003)
15. van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., Barros, P.: Workflow Patterns. Distributed and Parallel Databases, **14(1)** (2003) 5–51
16. Raposo, A. B., Fucks, H.: Defining Task Interdependencies and Coordination Mechanisms for Collaborative Systems. Frontiers in Artificial Intelligence and Applications, **74**, IOS Press (2002)
17. Russell, N., ter Hofstede, A. H. M., Edmond, D., van der Aalst, W. M. P.: Workflow Data Patterns, QUT Technical report, FIT-TR-2004-01, Brisbane (2004)
18. Baker, D., Georgakopoulos, D., Schuster, H.: Awareness Provisioning in Collaboration Management Cooperative Information Systems, **11(1&2)** (2002) 145–173
19. Jorgensen, H. D.: Interactive Process Models. Ph. D. Thesis, Norwegian University of Science and Technology, Norway (2004)
20. Dami, S., Estublier, J., Amiour, M.: APEL: a Graphical Yet Executable Formalism for Process Modeling. Automated Software Engineering, **5(1)** (1998)
21. van der Aalst, W. M. P., Hofstede, A. H. M.: YAWL: Yet Another Workflow Language. Information Systems (2003)
22. Duval, E. (Ed.): IEEE 1484.12.1-2002 Learning Object Metadata Standard, IEEE Official Standard (2002)
23. Kagal, L.: Rei: A Policy Language for the Me-Centric Project. HP Labs Technical Report, HPL-2002-270 (2002)

Learning Process Models as Mediators Between Didactical Practice and Web Support

Renate Motschnig-Pitrik and Michael Derntl

Department of Knowledge and Business Engineering and
Research Lab for Educational Technologies, University of Vienna,
Rathausstrasse 19, 1010 Vienna, Austria
{renate.motschnig, michael.derntl}@univie.ac.at

Abstract. Within the last decade the introduction of technology-enhanced learning ("e-learning") has become a focal strategy in several universities and organizations. While much research has been devoted to producing e-content, describing it with metadata, and to constructing e-learning platforms, relatively little attention has been paid to using patterns and conceptual modeling techniques as a means of knowledge development and communication serving to improve the learning process in terms of depth, scope, and effective tool support. Our research is targeted at filling this gap by considering conceptual models of learning processes as mediators between rich didactic elements and Web service modules that closely match students' and instructors' demands on effective support. In this paper we illustrate our pattern-based research framework by giving an example, discussing the driving role and merits of conceptual modeling, providing an overview of our pattern knowledge base, and sharing our vision for future development.

1 Introduction

Technology-enhanced learning has become a hot topic for every university and organization. E-content, its description by metadata, and its delivery via learning platforms employ the minds of many researchers, teachers and administrators. In our view, the current conception of the whole complex phenomenon of technology-enhanced learning is strong with regard to different forms of representing, sharing, and delivering learning content anytime and everywhere. However, it seems quite weak in re-engineering learning *processes* such as to exploit technology to a degree that surpasses mere representation, sharing, and delivery by offering radically novel learning scenarios [1]. These scenarios blend face-to-face and Web-supported learning such that the strengths of both settings, mediate and immediate, can be exploited and the learning process can proceed closer to the intentions and needs of individuals.

We have experienced that blended scenarios, due to their reliance on multiple media-didactic and face-to-face elements tend to be more versatile and complex than traditional lecturing. This is why structuring and abstraction mechanisms with well defined semantics, as they are typically provided by modeling languages such as the UML, are particularly well suited for model building and sharing of scenarios of technology-enhanced teaching/learning processes. However, semi-formal visual models as means of communication and tools *driving design and evaluation processes* are rare. Currently, focus is still primarily on the content, while the process and setting of learning are (almost) neglected, despite findings from various learning theories that underline the importance of process and its motivation and enactment by people. Hence the primary objective of this paper is to highlight the central role of conceptual modeling for mediating between learning processes and appropriate Web support modules.

Psychological and pedagogical theories agree on viewing lectures that solely serve to transmit information to several students as little effective in the long run, e.g. [2,3,4]. Knowledge that is not used tends to be forgotten very fast and in all but the most basic areas it is quite unlikely that different students will use the same knowledge in the near future. There is evidence that a form of learning and practice that tends to be more self-initiated and self-organized is more persistent and hence something we should strive for (e.g. [5]). In that respect our hypothesis that has been validated by three years of experience and Action Research is that modern ICT has the potential to play a significant part in approaching more individual, social, and persistent learning processes. In order to implement, research, adopt, and reuse such processes, they need to be made explicit. This is the point where conceptual models are indispensable in so far as they act as vehicles for *systematic* technological as well as educational innovation! Whereas the lead in effective learning still stays with persons, their capabilities, and social- and interpersonal values, thoughtfully designed scenarios, accompanying Web services, and easily accessible content have the potential to significantly support persons by reducing the time and effort needed for various organizational and administrative issues. This time then can be invested in engaging in a challenging, more spontaneous facilitation of learning.

Focusing on conceptual modeling, our approach to technology-enhanced learning proceeds in two major steps[1] as sketched in Fig. 1. The initial step comprises the capturing of successful learning/teaching processes in visual models taking on the form of extended UML activity diagrams. Thereby we focus on the modeling of activities and associated document flows (or “content flows"). The patterns resulting from capturing recurring process phases are organized at varying levels of abstraction and composition. Typically, they are used in presentations and discussions regarding didactical practice and applicability in specific course contexts. At the same time, the patterns form the specifications for the second step, namely the design and prototyping of interactive Web services [7,8] that implement the patterns as open-source modules and thus ease the organization and administration of courses as well as the communication and cooperation of participants. The Web services are applied in courses, reflected upon by students, discussed among instructors, and incrementally and iteratively improved, specialized and diversified, tightly following didactical practice as reflected by

[1] Note that a more detailed description of the transition process following the Blended Learning System Structure (BLESS) model is given in [6].

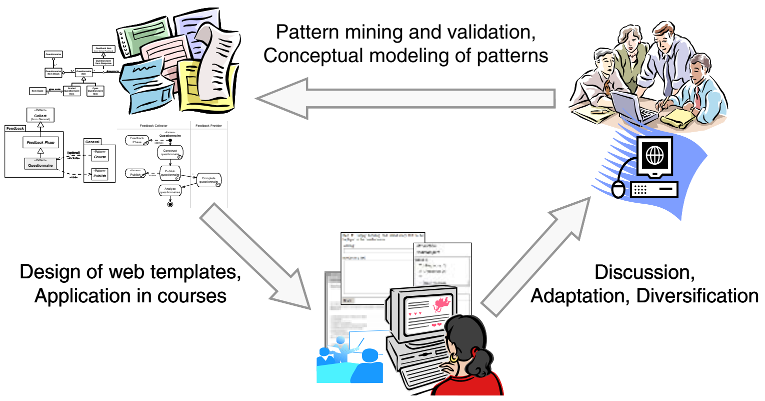

Pattern mining and validation,
Conceptual modeling of patterns
Design of web templates,
Application in courses
Discussion,
Adaptation, Diversification

Fig. 1. Conceptual models as central elements in developing effective, technology-enhanced learning support driven by action research

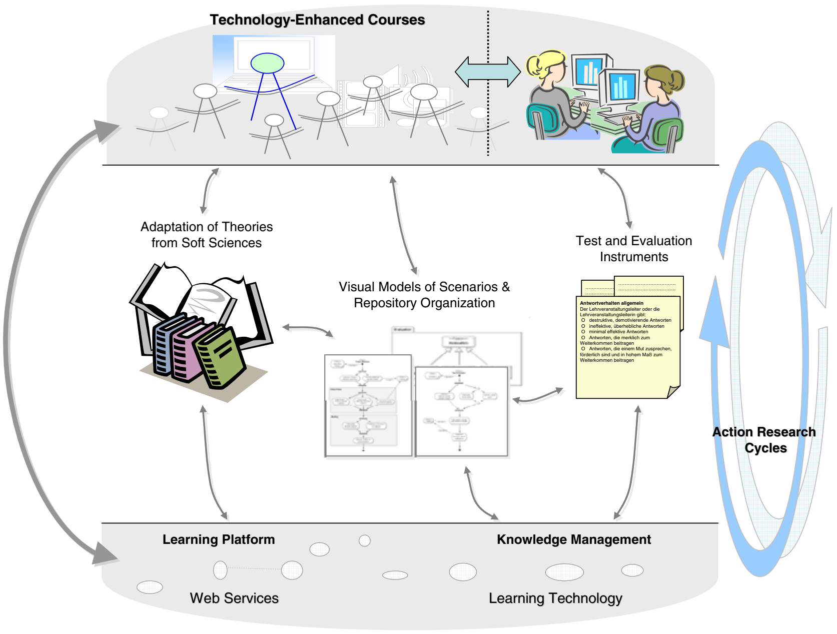


Fig. 2. Conceptual modeling centrally embedded in the technology-enhanced learning space

actual users [9]. Each course is associated with precisely those Web services that are actually needed in a given phase, no more and no less, in order to provide focus.

Briefly, our overall question and target is: How can learning processes that aim at deep, significant learning [10] be captured, co-developed by teams of experts, and communicated to practitioners? In this paper we focus on the conceptual modeling issues of blended, significant learning. Readers interested in the Action Research perspective of our pattern-based approach are referred to [11], the derivation of Web services is described in more detail in [12,13], and [14,15] put emphasis on the Person-Centered didactical baseline [10] of our technology-enhanced learning framework. The embedding of conceptual modeling in that framework is depicted in Fig. 2. An initial example in the following Section is intended to set the scene by giving a taste of a course in which significant, whole-person learning is the major goal. We will show how conceptual modeling in terms of UML activity diagrams, stereotypes, and patterns is used to capture course skeletons and provide the core for tool support and evaluation. In Section 3, the role of conceptual models in technology-enhanced learning will be elaborated. Section 4 introduces technology-enhanced learning stereotypes as extension mechanisms to UML, presents our pattern knowledge base, and relates our work to other pattern initiatives. In the final Section we will share our vision for future research and development.

2 Example: A Technology-Enhanced Course on Soft Skills in Project Management

This Section motivates the use of conceptual models for capturing didactically rich, technology-enhanced learning processes. Fig. 3 illustrates a course scenario from a practical course on soft skills in project management using a UML activity diagram with stereotype extensions. The swimlanes serve to relate activities and document flows with those actors that play the central role and frequently control the respective activities. This allows for intuitive gross estimates on the degree of instructor- or student-centeredness of courses at first glance.

In order to provide room for active interaction in class, fundamental material, links, and a list with references to further literature are supplied by the instructor over the learning platform at the time of course initialization. Also, key data on the course such as time, location, goals, brief description, etc. are provided such that students have initial information before enrolling in the course. The respective activity ("Create course space..."') primarily proceeds on the Web and hence is stereotyped with a 'W' icon in Fig. 3.

The initial meeting, stereotyped with a 'P' icon standing for *present*, is used to discuss the innovative course style, requirements, and learning methods, as well as to introduce the learning platform and to finalize the list of participants. Then students are asked to fill out an online questionnaire aimed at capturing their initial motivation, attitudes towards learning, ways they tend to profit from academic courses, etc. Furthermore, students are asked to assign themselves to small teams of about three to four students for cooperative work.

The face-to-face thread of the course consists of ten moderated workshops, 4 hours each, where individual topics within the gross framework of "soft-skills in project management" are elaborated following a strongly interactive style. The first three workshops are moderated by the instructor who introduces various didactical techniques such as team discussions, collection of issues on a flip chart,

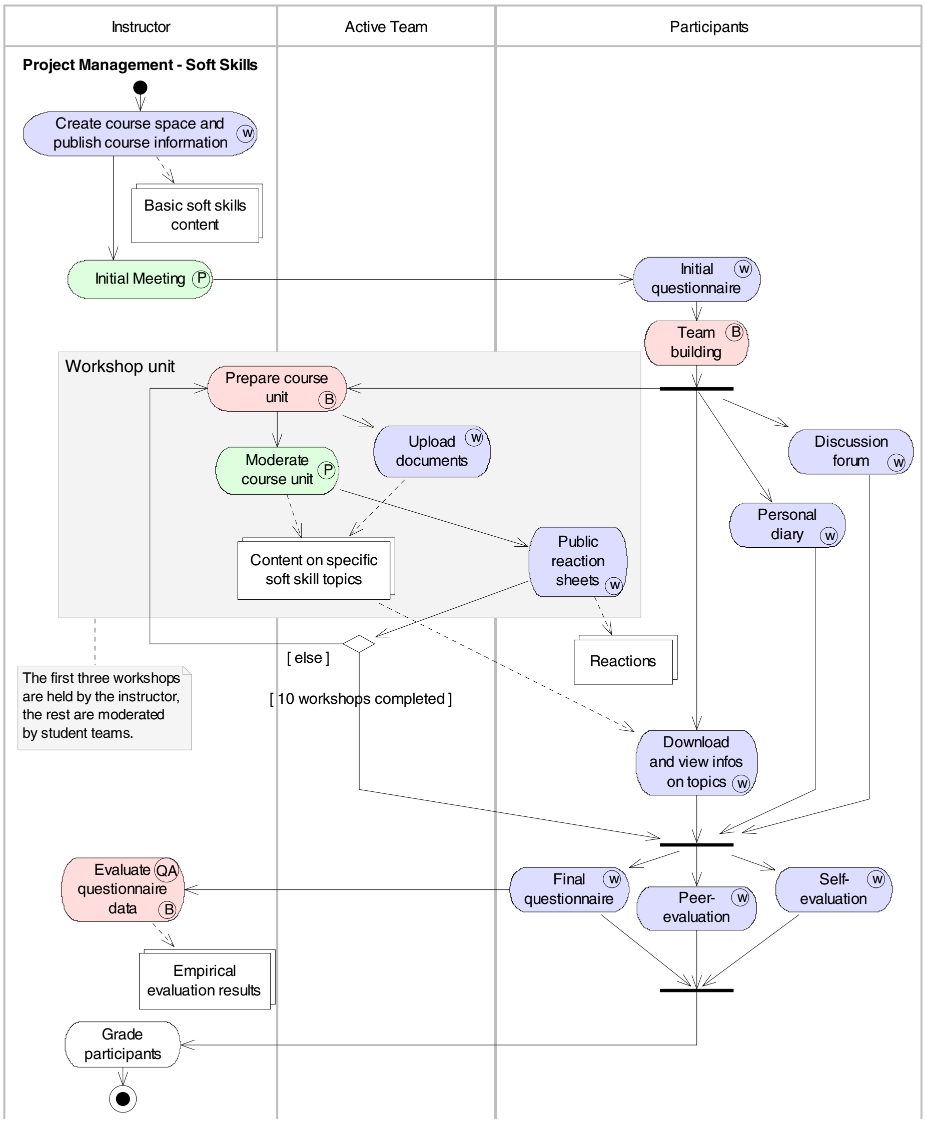

Instructor
Active Team
Participants
Project Management - Soft Skills
Create course space and publish course information W
Basic soft skills content
Initial Meeting P
Initial questionnaire W
Team building B
Workshop unit
Prepare course unit B
Upload documents W
Moderate course unit P
Content on specific soft skill topics
Public reaction sheets W
Discussion forum W
Personal diary W
Reactions
[else]
[10 workshops completed]
The first three workshops are held by the instructor, the rest are moderated by student teams.
Download and view infos on topics W
Evaluate questionnaire data QA B
Final questionnaire W
Peer-evaluation W
Self-evaluation W
Empirical evaluation results
Grade participants

Fig. 3. Course scenario blending face-to-face meetings with Web-supported elements following a Person-Centered style

moderation cards, mind maps, role playing, etc., by applying them. Students reflect how they perceived the whole situation in online reaction sheets, which are discussed in the subsequent workshop unit. Accompanying descriptions of these techniques and more theoretical background on their application is provided via the learning platform and can be inspected on demand. The remaining seven workshops are prepared by teams of students on topics we agree upon during the initial sessions. Preparation of a workshop includes the provision of e-content regarding the selected topic, as shown in the document flow object produced by “Upload documents". Preparation also encompasses consultation with the instructor with regard to the moderation sequence and elements (included in “Prepare course unit", which is stereotyped with a ‘B’ icon for denoting a *blended* activity). After each workshop, students submit online reaction sheets that can be read by all participants and are aimed at providing multi-perspective feedback to the team that moderated the workshop (the “active team" in Fig. 3).

At any time, students have access to the basic material provided on the platform, as shown in the Web-based activity “Download and view info on topics" of the course scenario. Concurrently, they are expected to briefly document their learning activities in a personal diary that shall support them in writing their self-evaluation at the end of the course. Furthermore, a discussion forum is available for communication with the instructor, Web master, and fellow students on all course relevant issues.

At the end of the course students evaluate themselves online. This is accomplished by responding to questions such as: What did I contribute? What could I take with me from the course? How intensive was my contribution with respect to my team mates? etc. In addition, an online peer-evaluation is conducted in which each team evaluates all other teams in terms of their moderated workshop and the e-content they provided. Furthermore, each student completes a final online questionnaire that is used to evaluate the course on a more objective level. The self- and peer-evaluations are used by the instructor in his or her grading, thus complementing the grading process by an individual- and a group perspective reflecting the participative, student-centered didactic strategy inherent in course conception and design.

A first look reveals that the scenario has multiple threads and didactic elements. In our view, however, this additional complexity adds significant value to the learning process, such as:

– More self-directed learning with more responsibilities of the learners and the group;
– Learning on the intellectual level, due to the elaboration of literature, as well as learning on the social and personal level due to intense teamwork and moderation of a course unit;
– More active participation and communication of students and instructor in face-to-face as well as online phases; during interactions active listening skills and attitudes like openness, respect, and the desire to deeply understand the other person are essential. They tend to be acquired through interaction if perceived by students.
– More authenticity of the problems to be tackled can be achieved, since students can select problems and material and raise questions they find worth considering;
– More perspectives on the content/theories can be discussed;
– Students take on more roles. Besides being authors, moderators, presenters, and listeners, they are peers and comment on the work of others;
– More group orientation and cooperation;

– Explicit consideration and integration of qualitative and quantitative means of quality assurance allowing for participative, formative evaluation and improvement of the course.
– Note, however, that instructors need interpersonal competencies that go far beyond being good lecturers in order to facilitate significant learning in technology-enhanced, person-centered courses.

We have experienced that the diagrammatic notation is indispensable for sharing the learning design with colleagues. Also, the choice of proper abstractions, consistent names, and the provision of multiple grain sizes have proved essential in communicating didactical practice in general and in the contribution of technology-enhanced learning elements in particular.

In the scenario depicted in Fig. 3, the activity "Peer-evaluation", for instance, refers to a *pattern*. This denotes a reoccurring activity sequence having a separate, reusable activity diagram. Patterns will be revisited and discussed in more detail in Section 4. While the results of quantitative studies based on the initial and final online questionnaire of the course on Project Management–Soft Skills are intended to be discussed in an upcoming paper, below we share excerpts from student's self-evaluations:

One student writes: "I hope I have contributed with my own inputs regarding the topic of negotiation and through active participation in all course units. I have learned to apply new moderation techniques and have acquired a balanced overview of soft skills... In addition I could, for sure, gain maximum benefit through frequently posing my own questions and thereby framing the discussions. A key experience was the workshop we held on our own: Despite intensive preparation I have realized ways of improvement that were shown up by the feedback we received."

Another student reports: "I have participated actively and have often volunteered to take part in exercises since I have seen that it is impossible to moderate a good workshop without the support of the whole group. I have delivered detailed reaction sheets since I believe that honest and specific feedback of the group can be truly facilitative."

Our experience and students' reactions substantiate the added value of the more complex, technology-enhanced scenario. The particular benefit of conceptual modeling regarding quality assurance lies in the fact that questions as well as reactions can be associated with process phases and didactic elements, thus adding structure and transparency to researching technology-enhanced learning. Generally speaking, our solution regarding knowledge communication of the learning scenario lies in the specification of conceptual models as means of information sharing and provision of blueprints for deriving support modules.

3 The Role of Conceptual Modeling in Technology-Enhanced Learning

In our research framework on technology-enhanced learning, conceptual models take on a key position that specializes, and in specific issues even surpasses, the role of conceptual modeling in software engineering and information systems. More specifically, conceptual models of technology-enhanced learning processes allow for:

– Explication of didactical knowledge, in particular of *learning strategies* such as problem-based learning, inductive derivation of knowledge, deductive learning, constructivist knowledge creation, etc.
– Specification of the *social setting* and role of learning activities, such as "team building" or "personal diary".
– Models reveal the cooperative structures in learning activities, e.g. cooperation in teams or between participants and instructors, and the interplay of present and online/distant phases in these activities.
– Presentation of learning processes at *various levels* of generalization / specialization and arbitrary switching between levels, if that leads to better understanding.
– Knowledge communication between educational scientists, educators, educational technologists, and developers. This is especially important as learning design is an inherently interdisciplinary task, where conceptual models can play an important mediator role [16].
– Derivation of patterns to capture generic and reusable practices and organization of patterns in a knowledge base (repository) [17]. In this respect, the use of conceptual modeling techniques in combination with the object-oriented paradigm can contribute to increasing the extensibility of the pattern structures in the knowledge base.
– Reuse of scenarios and supporting Web services.
– Platform independent specification of functional requirements on a supporting learning platform.
– Provision of a conceptual framework for research, most prominently *Action Research* [18]. This means that individual didactical elements (phases, threads, activities) can be researched in a targeted, participative way [19] considering different users' perspectives.
– Specification of the *interdependence between process and content flows.* This allows for the subsequent derivation of workflow- or better learnflow models and the integration of e-content modules [20].
– Identification and explicit integration of various means of *quality assurance* activities into learning scenarios in order to improve learning processes. Examples of such activities are online reaction sheets allowing for formative evaluation or final online questionnaires aiming at quantitative studies.

Whereas description of the extensions to UML diagrams in the form of stereotypes are postponed to the next Section, note, specifically, that we emphasize *process* over content, claiming that for good learning the *design of the learning process* is at least as important as content design and that these two features must go hand in hand. Therefore the patterns capture successful, largely domain-independent teaching/learning or facilitating processes and consider content, being domain-dependent, as complementary, yet *integrated* into the process view. From the modeling perspective, content flows are specified in the form of "object flows". Swimlanes in activity diagrams can be used to indicate the primary actor who elaborates and provides the content.

Furthermore, our more technical goal is to raise the *level of abstraction* of learning platforms. When we view current platforms as providing useful standard “atoms" like forum, workspace, folder, access rights management, chat, etc., our proposal can be seen as to combine these atoms into *molecules*. These are communicating Web services built from atoms and arranged to optimally support the underlying process patterns. A collection of Web services is instantiated for each course such that the course’s workflow (better learnflow) can be supported at a level that is considerably higher and more user-centered than the standard atoms provided by current platforms. Note, however, that our conceptual scenario models are *not* (yet) intended to specify automatically executable flow semantics, contrary to the IMS Learning Design specification [21] for example, which shows a very high degree of formal expression using XML. In our case, formal completeness was deliberately traded for better understandability where required. Nonetheless, the learnflow model of a pattern acts as the primary guidance system for specifying and implementing the functionality required by a Web service supporting that pattern. For instance, a Web service implementing the `Diary` pattern would emphasize particularly those Web-based activities that are arranged in the pattern’s activity model, i.e. publishing of diary requirements, diary initialization, and diary review on the side of the instructor, and updating the diary on the side of the students (see Fig. 5). However, this does *not* imply that other “gadgets" or add-on functionality, such as sorting or querying the diary log, are necessarily excluded from a concrete `Diary` implementation.

4 Conceptual Modeling of Technology-Enhanced Learning Patterns

4.1 Background and Related Approaches

Patterns are generic descriptions of solutions to frequently recurring problems or situations [22]. The pattern approach was originally developed by C. Alexander in the field of architecture, but currently most prominently considered in software design as a vehicle for efficient communication of best practice in tackling common design situations in object-oriented software systems [23]. The pattern approach found its way into many other disciplines and in recent years also into technology-enhanced learning. Surprisingly, most of these approaches rely on purely text-based knowledge communication. Few, if any, employ conceptual modeling techniques.

– The E-LEN project [24] is a European network of e-learning organizations. Among its four special interest groups, patterns are used as means of communication, development, and dissemination of effective e-learning experiences. Conceptual modeling does not play any noticeable role in the E-LEN efforts.
– The Pedagogical Patterns Project [25] provides a compilation of prose-style patterns for many educational scenarios. However, these patterns neither

use conceptual models, nor do they include or address explicitly the use of learning technology.
- The Educational Environment Modeling Language E^{2}ML [26] defines a complex, pedagogically neutral modeling notation for instructional design. With respect to learning processes, this language provides a method of modeling the timeline of a course design, which produces a Gantt-chart-like visualization of the "action flow".
- IMS Learning Design includes a learning design best practice and implementation guide [27] that employs conceptual modeling techniques: it shows learning content and resources in a process-oriented, formalized way by using use cases for analysis, activity diagrams for modeling of the use cases' narratives, and IMS/LD compliant XML documents that are used for content development and packaging. Patterns are not explicitly considered – the ultimate artifacts produced compliant to this standard are XML documents.
- Other pattern approaches primarily address design and usability issues in Web-based environments while not explicitly referring to Web-based learning systems, such as patterns for hypermedia design [28], or for Human-Computer Interaction in general [29,30].
- There are some approaches that do not explicitly refer to patterns in the Alexandrian sense, but that are somehow conceptually related in their viewpoints on technology-enhanced teaching/learning activities, e.g. CSCL scripts [31] or Laurillard's Conversational Framework [32].

The approach employed in this paper makes intensive use of conceptual modeling techniques, as we believe that modeling is one of the primary means of handling and decomposing the complexity inherent in socio-technical environments. We concentrate on a balanced compromise between purely text-based means of communication and description on the one hand and highly formal representations (e.g. XML) for machine-processing on the other hand. While most of the related pattern approaches presented above focus on content and technological aspects, one distinctive asset of our approach is the concentration on the learning process and on arrangement of face-to-face and Web-based elements in (blended) learning design and, pragmatically, the transparent externalization and communication of effective technology-enhanced learning experience. Another specific and, in our view, essential feature of our approach is the existence of a psychologically well founded theory, namely the Person-Centered Approach [33], which provides the didactical baseline for pattern design.

4.2 UML Extensions for Technology-Enhanced Learning

For explicitly depicting the face-to-face and online elements involved in technology-enhanced learning processes the standard UML meta-model was extended by adding the following custom stereotypes for action states (better known as "activities") in activity diagrams:

«web-based» Means that the activity primarily proceeds on the Web (or distant with technology support). If an activity stereotyped this way occurs in an activity diagram, the respective pattern typically provides a Web template for online-support of that activity. This stereotype is presented as an icon (a circle containing the letter 'W') at the right-hand side of the activity. The activity is additionally filled in light blue to increase visual effect.

«present» Indicates that the activity primarily takes place in a face-to-face setting. The icon for this stereotype is a circle containing the letter 'P' at the right-hand side of the activity. The activity carrying this stereotype is filled in light green.

«blended» Indicates a mix of the former two stereotypes: the activity is conducted in a blended style, mixing or alternating online and face-to-face modes and delivery channels. The icon for this stereotype is a circle with the letter 'B'. An activity carrying this stereotype is filled in light red.

«quality-assurance» This stereotype is attached to activities that produce documents or data which can subsequently be used for the (predominantly formative) evaluation of the learning process and learning support (e.g. online questionnaires or reaction sheets). The icon for this stereotype is a circle containing the acronym 'QA'.

«pattern» This stereotype is typically attached to subactivity states. A subactivity points to another activity diagram that shows a more detailed flow of the activities. This helps to avoid overloading the diagrams with too many activities, allowing different levels of granularity and aggregation in the models. If the subactivity points to a pattern sequence, it carries the stereotype «pattern». Note that this stereotype may be omitted in diagrams for reasons of brevity and readability, as for example in Fig. 3.

Examples showing the association of these stereotypes with concrete activities appear in the scope of a complete course scenario in Fig. 3, and as part of a pattern's activity model in Fig. 5.

4.3 Pattern Organization and Modeling

Currently about 50 patterns, which were mined from the technology-enhanced teaching / learning practices of the authors' institution over the last three years, are available in our pattern repository [17]. The repository offers an initial, rich pool of patterns that can flexibly be combined and extended in response to the situation at hand. The patterns in the repository are arranged at different levels of detail and abstraction. Unlike most other pattern approaches that specify pattern inter-relations textually, we provide a conceptual model of the repository and the relations among the patterns using UML static structure diagrams. The patterns describe courses and course modules / phases being composed of smaller, reusable process elements, such as publishing of electronic content, knowledge construction in groups, team exercises, online discussion, various forms of feedback and evaluation, and other techniques suited for technology-enhanced learning. Each pattern is hosted in a pattern package, which is used to group related patterns together. Currently the repository includes seven packages, which are listed in alphabetical order and briefly described in the following:

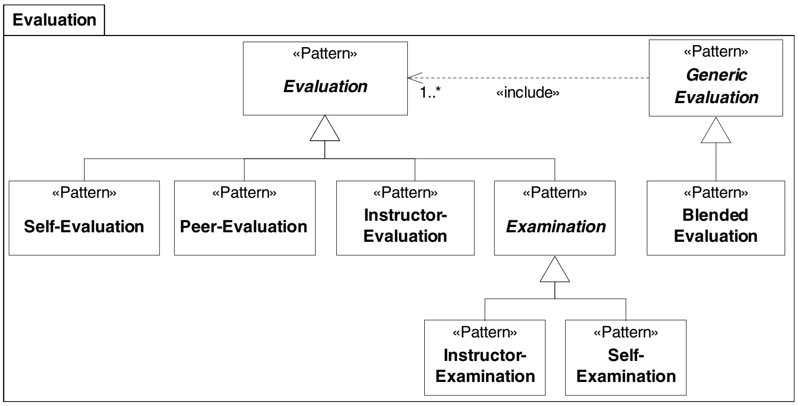

Evaluation
«Pattern» Evaluation
1..*
«include»
«Pattern» Generic Evaluation
«Pattern» Self-Evaluation
«Pattern» Peer-Evaluation
«Pattern» Instructor-Evaluation
«Pattern» Examination
«Pattern» Blended Evaluation
«Pattern» Instructor-Examination
«Pattern» Self-Examination

Fig. 4. The Evaluation pattern package

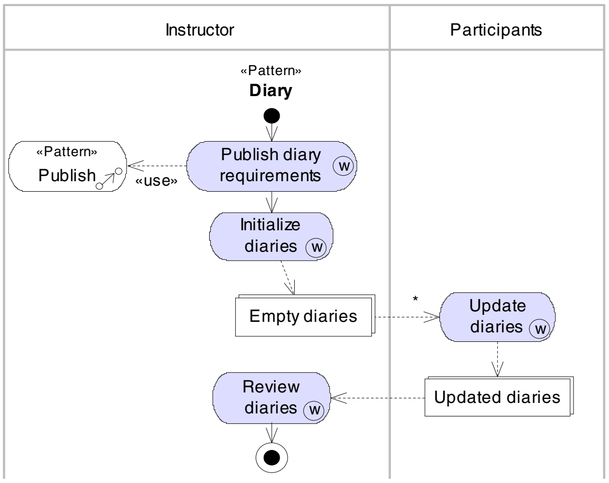


Fig. 5. Activity model of the `Diary` pattern

Assessment. Methods of assessing participants' achievements with the ultimate goal of determining a grade for each participant.

Course types. Describes familiar course types in terms of technology-enhanced practices. Examples: `Lab Course` or `Seminar`.

Evaluation. Different methods of evaluating participants' contributions in a learning activity, whereby evaluation means valuing judgment on the performance of participants. This package is depicted in Fig. 4.

Feedback. Different ways of collecting feedback from course participants. Examples: `Reaction Sheets` for written, unstructured feedback, or `Feedback Forum`, which uses discussion forums for collecting more structured feedback.

General. Generally reusable patterns or patterns not matching any one of the specific purposes defined for other packages. `Diary` is an example of a general

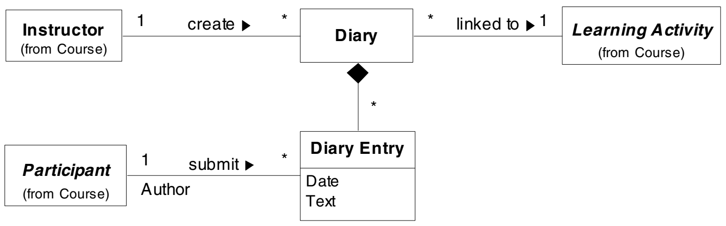


Fig. 6. Structural model of entities and relationships involved in the `Diary` pattern

pattern, which can be employed in almost any learning scenario. It is used in the scenario in Fig. 3, and its activities are depicted in Fig. 5. Other general patterns include `Achievement Award`, `Publish`, or `Preliminary Phases`.

Interactive elements. The largest package, hosting patterns used to foster interaction and interactivity among participants, instructors, tutors, and/or external guests. Examples: `Brainstorming`, `Theory Elaboration`, `Online Discussion`, `Consultation`, etc.

Project-based learning. Patterns describing some sort of iterative and/or incremental learning process which can be expressed through several successive (project) milestones, for example `Learning Contracts`.

In the structural repository model, the patterns are modeled using stereotyped classes. Relationships between patterns take on two different types:

– *Generalization*, connecting a concrete lower-level pattern with a more abstract higher-level pattern (e.g., `Evaluation` as a generalized form of the `Peer-Evaluation` pattern in Fig. 4).
– *Dependency*, modeling the inclusion, usage, or adaptation of a pattern by another pattern.

Each pattern also includes a structural model of the entities involved in the activity model of the pattern when appropriate (see Fig. 6 for an example). Besides supporting the user in understanding the underlying concept of a pattern, this is particularly useful when deriving requirements of Web support for the pattern. Subsequently, a combination of the structural models may serve to define a data model when implementing a learning platform particularly dedicated to blended, process- and person-centered learning.

Note that each pattern additionally includes a detailed textual description regarding the pattern's intent, motivation, parameters, relations to other patterns, examples of use, results of previous evaluations, and literature references.

5 Conclusions and Further Work

We have illustrated the central role of conceptual, semiformal process models in our framework on technology-enhanced learning (Fig. 2). We have outlined why visual, conceptual models are indispensable for capturing, promoting, researching, and improving rich didactical practices on several grounds. Firstly, they offer instruments that allow one to decompose and manage the complexity inherent in socio-technical practice. In this way they offer vehicles for knowledge communication that have proved effective in dialogues with psychologists, educational-, communication-, and computer scientists. Secondly, the latter can use the process models as functional specifications for deriving Web templates and prototypes of Web services implementing didactical practices being derived directly from students' and instructors' needs. Our strategy is to offer these Web services (CEWebS, Cooperative Environment Web Services) as open-source modules such as to allow for broad adaptation, improvement and reuse. A vision in this respect is the promotion and Web-support of person-centered didactical practice to contribute to making technology-enhanced learning more effective, significant, and enjoyable.

Aside of these traditional roles of conceptual models, technology-enhanced learning scenarios and patterns serve to capture and explicate theoretically founded didactical models such as inductive, deductive, and problem-based learning [34]. Furthermore, they can be used to denote those phases in the learning process, where specific document- or content flows immerse into the learning process or are produced as material- or content output. Finally and importantly, stereotyped activities in the process models allow one to drive attention to focal perspectives, such as activities being performed solely on the Web, in presence phases, or serving quality assurance. The integration of the latter, in particular, has opened up a new dimension, namely the explicit consideration and consequent improvement of the quality of learning processes as integral constituents of these same processes. Means such as transparent online reaction sheets or online questionnaires have proved to be simple yet effective and feed into our *participatory action research* initiative [35].

We have argued that modeling processes and artifacts of teaching and learning in the form of patterns allows one to reuse proven didactic principles and thus saves time for course design. This benefit is further strengthened in the case that the patterns are implemented the form of open-source Web services that significantly reduce the effort spent on organizational and administrative issues.

Further research follows multiple threads. One of them addresses the capturing and implementation of further patterns with a particular focus on international courses where students and instructors are truly distributed. We are also in the process of developing instruments for evaluating the effects of technology-enhanced, person-centered learning and the impact of instructor's attitudes, process design, and amount of online phases on issues such as learning outcome, motivation, effort, and personal relevance of learning. If this research provides a path to a more meaningful way of learning in technology-enhanced and immediate environments, it will have served its purpose well.

References

1. Papert, S.A.: Mindstorms. 2nd edn. Basic Books, New York (1999)
2. Rogers, C.R.: On Becoming a Person - A Psychotherapists View of Psychotherapy. Constable, London (1961)

3. Tausch, R., Tausch, A.M.: Erziehungs-Psychologie. Hogrefe, Göttingen, Germany (1998)
4. Jonassen, D.H., ed.: Handbook of research on educational communications and technology. 2nd edn. Lawrence Erlbaum Associates, Mahwah, NJ (2004)
5. Wenger, E.: Communities of practice - Learning, meaning, and identity. Cambridge University Press, Cambridge (1998)
6. Derntl, M., Motschnig-Pitrik, R.: The role of structure, patterns, and people in blended learning. The Internet and Higher Education **8** (2005) 111–130
7. Curbera, F., Nagy, W.A., Weerawarana, S.: Web services: Why and how? In: OOPSLA'2001 (Workshop on Object-Oriented Web Services), Tampa, Florida (2001)
8. W3C: Web Services Architecture - W3C Working Draft 8 August 2003. `http://www.w3.org/TR/ws-arch/` (2003)
9. Motschnig-Pitrik, R., Derntl, M., Mangler, J.: Developing cooperative environment web services based on action research. In: 5th International Conference on Practical Aspects of Knowledge Management (PAKM 2004), Vienna, Austria, Springer (2004) 453–462
10. Rogers, C.R.: Freedom to Learn for the 80's. Charles E. Merrill Publishing Company, Columbus, Ohio (1983)
11. Derntl, M., Motschnig-Pitrik, R.: A pattern approach to person-centered e-learning based on theory-guided action research. In: 4th International Conference on Networked Learning (NLC), Lancaster, UK (2004)
12. Derntl, M., Mangler, J.: Web services for blended learning patterns. In: 4th IEEE International Conference on Advanced Learning Technologies, Joensuu, Finland, IEEE Computer Society (2004) 614–618
13. Mangler, J., Derntl, M.: CEWebS - Cooperative Environment Web Services. In: 4th International Conference on Knowledge Management (I-KNOW '04), Graz, Austria (2004) 617–624
14. Motschnig-Pitrik, R., Holzinger, A.: Student-centered teaching meets new media: Concept and case study. Journal of Educational Technology & Society **5** (2002) 160–172
15. Motschnig-Pitrik, R., Mallich, K.: Effects of person-centered attitudes on professional and social competence in a blended learning paradigm. Journal of Educational Technology & Society **7** (2004) 176–192
16. Heemskerk, M., Wilson, K., Pavao-Zuckerman, M.: Conceptual models as tools for communication across disciplines. Conservation Ecology **7** (2003) Article #8
17. Derntl, M.: Patterns for Person-Centered e-Learning. PhD Thesis, Faculty of Computer Science, University of Vienna (2005)
18. Baskerville, R.L.: Investigating information systems with action research. Communications of the Association for Information Systems **2** (1999)
19. Ottosson, S.: Participation action research - a key to improved knowledge of management. Technovation **23** (2003) 87–94
20. Bajnai, J., Steinberger, C.: Eduweaver - the web-based courseware design tool. In: International Conference WWW/Internet 2003, Algarve, Portugal, IADIS (2003) 659–666
21. IMS Global Learning Consortium: IMS learning design specification. `http://www.imsglobal.org/learningdesign/index.cfm` (2003)
22. Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.: A Pattern Language - Towns, Buildings, Construction. Oxford University Press, New York (1977)
23. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA (1995)

24. E-LEN Project: E-LEN project homepage. http://www.tisip.no/E-LEN/ (2003)
25. Pedagogical Patterns Project: Pedagogical Patterns Project homepage. http://www.pedagogicalpatterns.org (2002)
26. Botturi, L.: E^{2}ML - educational environment modeling language. In: Ed-Media'03, Honolulu, Hawaii, USA, AACE Press (2003) 304–311
27. IMS Global Learning Consortium: IMS learning design best practice and implementation guide. http://www.imsglobal.org/learningdesign/ldv1p0/imsld_bestv1p0.html (2003)
28. Nanard, M., Nanard, J., Kahn, P.: Pushing reuse in hypermedia design: golden rules, design patterns and constructive templates. In: 9th ACM Conference on Hypertext and Hypermedia, Pittsburgh, Pennsylvania, ACM Press (1998) 11–20
29. Tidwell, J.: UI patterns and techniques. http://time-tripper.com/uipatterns (2002)
30. Borchers, J.: A Pattern Approach to Interaction Design. John Wiley & Sons, Chichester (2000)
31. Dillenbourg, P.: Over-Scripting CSCL: The risks of blending collaborative learning with instructional design. In Kirschner, P.A., ed.: Three worlds of CSCL. Can we support CSCL. Open Universiteit Nederland, Heerlen (2002) 61–91
32. Laurillard, D.: Rethinking University Teaching: A Conversational Framework for the Effective Use of Learning Technologies. 2nd edn. Routledge Farmer, London (2001)
33. Rogers, C.R.: Client-centered therapy: Its current practice, implications, and theory. Houghton Mifflin, Boston, MA (1951)
34. Swertz, C.: Didaktisches Design. Ein Leitfaden für den Aufbau hypermedialer Lernsysteme mit der Web-Didaktik. Bertelsmann Verlag, Bielefeld (2004)
35. Motschnig-Pitrik, R.: An action research-based framework for assessing blended learning scenarios. In: Ed-Media'04, Lugano, Switzerland, AACE Press (2004) 3976–3981

A Fundamental View on the Process of Conceptual Modeling

S.J.B.A. Hoppenbrouwers, H.A. (Erik) Proper, and Th.P. van der Weide

Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU
{S.Hoppenbrouwers, Th.P.vanderWeide, E.Proper}@cs.ru.nl

Abstract. In an ongoing effort to better understand the process of creating conceptual models (in particular formal ones), we present a fundamental view of the process of *modeling*. We base this view on the idea that participants in such a process are involved in a deliberate and goal-driven effort to share and reconcile *representations* of their personal conceptions of (parts of) the world. This effort takes the shape of a *modeling dialogue*, involving the use of *controlled language*. We thus take a fundamental approach to *subjective* aspects of modeling, as opposed to traditional approaches which essentially consider models as *objective* entities. We position and present our initial theory of modeling, and briefly discuss how we intend to validate and further develop it.

1 Introduction

The view on conceptual modeling as presented in this paper is rooted in a number of different modeling practices and theories. We take the point of view that the act of conceptual modeling should be understood (1) at a fundamental level, (2) in context of what models are for, and (3) taking into account the capacities and goals of the individuals who create or use them [1]. In taking this view we have, first and foremost, been inspired by the conceptual modeling approach called ORM (Object Role Modeling) [2]. We have been involved in studies of the application of ORM in domain modeling and requirements engineering [3, 4]. In addition, we have drawn from theory and practical experience acquired through the ArchiMate project, which focused on Enterprise Architecture Modeling [5].

1.1 Focus and Fundamental Assumptions

The central question we address is: "Why do we model?". We attempt to answer this question in a generic fashion that nevertheless clears a path for further and more specific research. Further elaborations and directions for solutions are based on our answer to the question, which is: "We model because modeling answers questions". While this is too generic an answer to solve much, it does directly clarify our approach to conceptual modeling. By asking: "*Who asks the questions that need to be answered?*" and "*Why these people ask those questions?*", we immediately arrive at a view on conceptual modeling that is deeply rooted in *communication*, involving *language as a means to achieve communication*

[6, Chapter 3]. This entails that we are especially interested in *cooperative* aspects of modeling: we focus on what a model, and the process of creating it, achieves in terms of communication between people. Essentially, modeling concerns creative learning. It is akin to communication techniques encountered in education and knowledge management. Modeling is a learning process in which cooperating participants together construct a view on (and a model of) reality [7]. Ultimately, therefore, we see modeling as a tool for developing and sharing knowledge.

Our communication-based approach is largely inspired by the desire to understand (and eventually improve) the modelling *process.* We believe that indeed, for this purpose, communication aspects of modeling are crucial. However, we also believe that many if not all other approaches can be combined with or even integrated into such a view.

Modeling as we see it may or may not involve the use of a (semi-)formal modeling language, i.e. a modeling language the syntax and (in case of fully formal languages) semantics of which can be coherently formulated in a mathematical language. While we focus primarily on formal modeling, we include informal and semi-formal modeling in our view, and are strongly interested in the differences and commonalities that hold between the various flavours.

1.2 Questions and Answers Underlying Modeling

The vast majority of literature on modeling concerns restrictions on the form, structure, and meaning that a model (expressed in a certain language) should respect. Such restrictions may range from an iconic vocabulary for conveying a coherent set of informal notions, to a fully formal set of restrictions on syntax and semantics. We do not argue for or against any form of modeling or modeling language, but emphasize the importance of asking why a certain restriction is imposed, and what its relation is to the questions asked and answered in context of the modeling process and the use of the finished model. In fact, we are less interested here in the modeling languages per se than in the questions asked as part of the modeling process.

We observe that many of the questions asked during actual modeling are not answered if a complete, finished model is "read". Instead, many questions are asked and answered *during the process of modeling.* The finished model corresponds to the minutes of a meeting that has taken place [8]. Reading the minutes certainly answers some questions, but provides no further opportunities for asking new questions, nor to add to the answers or to verify whether what has been said is well understood (i.e. truly learned) by all parties involved. In addition, we observe that in many cases, people tend to adapt the modeling language used (the "Way of Modeling" [9]) to the needs that occur during the modeling process [10]. We can only hope to understand all these aspects of modeling by looking at the details of the process. We therefore propose a view on modeling that respects its *product* (and the intended usage thereof), while also clarifying the nature of the modeling *process* and what it might involve and achieve apart from the product as such. Our view thus is process-oriented, yet aspires to be complementary to product-oriented views.

The questioning-and-answering that takes place during modeling can fruitfully be seen as a *dialog* or *conversation*. Given the assumptions presented above, understanding the goals of modeling, and the means to match them, boils down to understanding the questions people ask during modeling, and the means they deploy to get them answered. Once this becomes clear, we can begin to work towards the formulation of basic *modeling strategies*. These are ways of proceeding in a modeling dialogue that are optimally fit to fulfill two main goals:

1. Answering all the questions the participants in the modeling process might have (explicitly or implicitly).
2. Answering all the questions asked by those who use the product (i.e. the completed model).

In this paper, we will not discuss modeling strategies as such, but merely pave the way for study of actual modeling dialogue and the strategies it involves. We focus here on the essentials of our view on modeling.

1.3 Positioning Verification and Validation

Restrictions on models are generally related to one of two sets of *demands on quality*: those related to *verifiability* (a.k.a. "internal quality") of a model, and those related to *validity* (a.k.a. "external quality") of a model. In formal modeling literature, emphasis lies mostly on verifiability, but clearly a good formal model must also be *valid*. However, in many cases validity is not a matter that can be resolved by any means of objective validation in the mathematical sense. It usually depends on subjective judgments passed and viewpoints held by humans. Because of this, though validation is considered an important and problematic issue, it is often discarded because it cannot be handled very well within the realm of mainstream computer science.

Based on our extensive personal experience in modeling, as well as our theoretical work in that area [1, 3, 11, 12, 6, and more] we expect that the validation of models (in both informal and formal modeling) can be much improved by means of better modeling processes and strategies, within a communicative approach. Along similar lines, it should also be possible to formulate dialogue-based strategies that lead to verifiable correctness in completed models.

It is not just the quality of models we are concerned about. We also hope that by finding detailed modeling strategies, we can eventually help deal with an increasingly problematic bottleneck that occurs in AI and system development: a growing demand for constant creation of formal models in specific and dynamic operational contexts, combined with a lack of people who are capable and willing to perform the modeling required.

Our main focus is on formal conceptual modeling because in terms of combined validation and verification, it poses the biggest challenge and is most urgent. Also, the modeling bottleneck mostly concerns formal models. We strive for an integrated approach to achieving validation and verification: a good process, resulting in a valid model which is also verifiably correct in the end. The key then is to achieve a careful and systematic exchange of questions and answers, guided and restricted by the particular demands on both validity and verifiability as posed by the context in which and for which a model is created.

1.4 Approach

Though science has since long embraced and studied the product of formal modeling (the models and modeling languages), the details of the underlying production process (modeling) still lie mostly in the realm of art. We aspire to be more scientific about the modeling process as such. This requires a study of modeling in terms of participant behavior. More in particular, we intend to find and develop well-formulated *strategies* as a means to describe modeling processes, in order to better understand what courses of action lead to good (valid, verifiable) formal models in line with specific demands posed by their contexts.

In finding answers to the questions raised above, we are currently in the stage of developing a theoretical framework. The *plausibility* and *soundness* of this framework is argued initially in terms of its linguistic [6, 13], epistemic [14] and semiotic [15, 16] foundations. Once a plausible and sound theoretical framework is created, we will conduct a series of modeling experiments to confirm the *validity* of the framework.

2 Conceptual Modeling

The aim of this section is to more closely investigate the process of conceptual modeling. In defining precisely what we mean by modeling a domain, we first need to introduce a framework describing the essential process that takes place when a person (for example, a stakeholder) observes a domain (for example, a work situation to be supported by an information system).

Let us first consider what happens if some *viewer* observes 'the universe'. Our central underlying assumption is that viewers perceive a universe and then produce a *conception* of that part they deem relevant. The conceptions harbored by a viewer cannot be communicated and discussed with other viewers unless they are articulated somehow (the need for this ability in the context of system development is evident). In other words, a conception needs to be represented. Following Peirce, we embrace the idea that both perception and conception of a viewer are strongly influenced by her interest in the observed universe. The *viewer* is an actor perceiving and conceiving the universe (the 'world' around the viewer), using her senses. A *conception* is that which results, in the mind of a viewer, when she observes the universe –using her senses– and interprets what she perceives. Finally, a *representation* is the result of a viewer denoting a conception, using some language and medium to express herself.

2.1 Viewers and Their Frame of Reference

From a modeling point of view, a viewer could metaphorically be seen as an observation tool (a telescope) used to obtain information from the observed universe. The modeler may observe the universe directly, but still depends on the viewer *and the representations she brings forth* to get (more) accurate information. An observation tool should provide a trustworthy image of the universe in such a way that structure that can be derived from the image corresponds to the structure of the observed universe. Different observation tools (or even different observations) may yield different images (representations), all reflecting the same universe.

In our context, a viewer is assumed to be *competent* (i.e. knowledgeable) [12] and *trustworthy* (i.e. not tell lies). The viewer is also capable of providing a *verbalized image of the observed world*, consisting of *statements* in some language (either verbal or graphical; ORM, for example, covers both). We also assume that the structure of the statements uttered has at least some correspondence with the structure of the world observed. Without referring to particular universals, we assume there to be some underlying commonality in how people perceive and conceptualize the world. Both the bio-cognitive make-up of people and their experiences of living as and among humans create at least some common ground, reflected in their language [6, 13].

As mentioned above, in conceiving a part of the universe, viewers will be influenced by their particular interest in the observed universe. In the context of system development (more in particular, enterprise architecture), this corresponds to what tends to be referred to as a *concern* [17]. For example, a viewer may be concerned with safety issues within a domain. Though we acknowledge that a concern may influence the choice of modeling language, we abstract for the moment form such peculiarities, and see a viewer purely as a *language source with a personal syntax.* Sentences delimited by this syntax convey the meaning of the associated (personal) world. The underlying semantical function is an unknown and possibly informal function. We call a language (intended for communication) informal if it has no well-defined syntax, or no semantic interpretation in terms of some underlying formal (i.e. mathematically expressed) model.

Concerns are not the only factors that influence a viewer's conception of a domain. Another important factor concerns the pre-conceptions a viewer may harbor as they are brought forward by their social, cultural, educational and professional background. More specifically, in the context of formal modeling, viewers will approach a domain with the aim of describing it in terms of some predefined set of meta-concepts, such as classes, activities, constraints, etc. The set of meta-concepts a viewer is used to using (or trained to use) when modeling a domain will strongly influence the conception of the viewer (the well known Sapir-Whorf hypothesis in its weak and commonly accepted form [18]). We therefore presume that when viewers model a domain, they do so from a certain perspective; their *Weltanschauung* (German for "view of the world") [19]. The Weltanschauung can essentially be equated to the notion of a *viewpoint* [17].

Viewers may decide to zoom in on a particular part of the universe they observe, or to state it more precisely, they may zoom in on a particular part of *their* conception of the universe. This allows us to define the notion of a *domain* as: any subset of a conception (being a set of elements) of the universe, that is conceived of as being some 'part' or 'aspect' of the universe. In the context of (information) system development, we have a particular interest in unambiguous abstractions from domains. This is what we refer to as a *model*: a purposely abstracted and unambiguous conception of a domain. Note that both the domain and its model are *conceptions* harbored by the same viewer. We are now in a position to define more precisely what we mean by *modeling*: the act of purposely abstracting a model from (what is conceived to be) a part of the universe.

For practical reasons, we will understand the act of modeling to also include the activities involved in the *representation* of the model by means of some language and medium. In line with [16], we consider modeling to boil down to "making statements in some language".

2.2 Participants in the Modeling Process

In this and the following sections, we will use the generic term *participant* for all actors taking part in the modeling process. Importantly, all such participants are viewers as defined above.

For the sake of the argument, let us consider a basic (and admittedly over-simplified) situation in which two participants in the modeling process play the following roles. One is the *domain expert*, who is competent and trustworthy; she knows all there is to know about the target domain, or can find out more if need be. In other words, she can *generate* and *validate* statements about the domain,

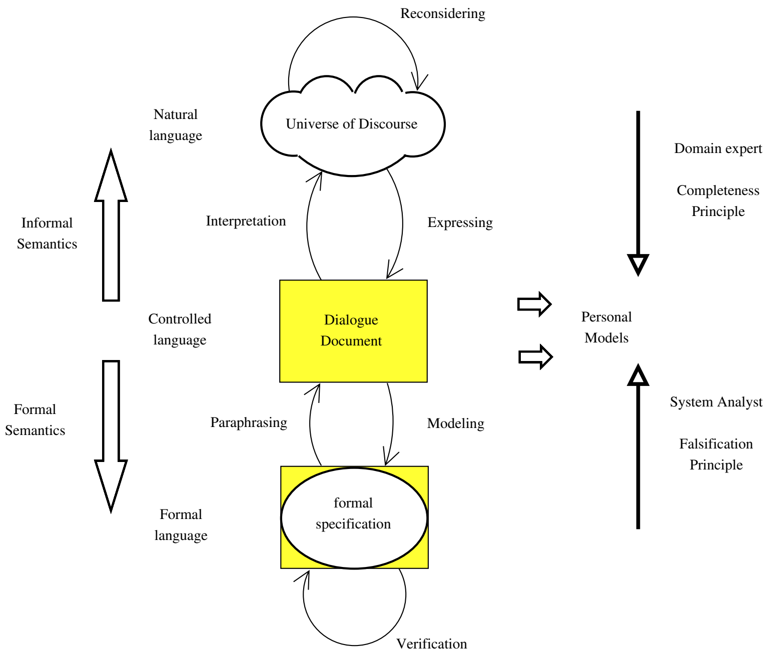


Fig. 1. The classic view on the conceptual modeling process

but she has no formalization skills. The other participant is the *system analyst*, who has no knowledge of the target domain but does know how to create a verifiably correct formal model. The interactive relation between the example roles of domain expert and system analyst is depicted in figure 1, which represents the classic view on modeling. The upper half of the figure shows the "informal" world of the domain expert, statements about this world which are typically expressed in natural language. The lower half of the figure shows the "formal" world of the system analyst, statements about which are typically expressed in some formal language. The link between the two worlds is achieved through a dialogue (and a *dialogue document* that records it). We presume the dialogue to be conducted using some form of controlled language (see discussion below).

In the activity of cooperatively creating a formal model on the basis of informal information, there is a parallel and a symmetry between the Completeness principle and the Falsification principle (positioned on the upper and lower right in figure 1). In the formal world, a model may be deemed falsifiable because it is semantically or syntactically incorrect. While such formal falsifiability is impossible in the informal world, this world allows for judgments of (in)completeness: has everything that needs to be said been said (and has no more than what is relevant been said)? Though the parallel may not be an ultimate, philosophical one, it does hold for the practice of formal modeling: the best we can do as we provide informal input is be complete (within the boundaries of relevance); the best we can do in formal articulation is be formally correct. The marriage between the two makes for good formal models.

In our view, then, the domain expert typically harbors an informal semantic function (natural language), while the system analyst's language may be expected to be governed by a formal semantic function. However, both are "language sources"; it is just that their syntax and semantics differ in structure and nature. Thus, beyond this example, it seems justified to indeed use the more neutral concept *participant* as a generic term for domain expert and system analyst. Participants all have their personal syntax and a formal or informal semantic function, depending on the roles they play in modeling.

In the context of communication resulting in formal models (in particular, the traces of communication recorded in the dialogue document; see figure 1), we strictly focus on written expressions. Though the document is linear, and therefore the order in which the text has been uttered is captured, further aspects of communication and medium (time, location, gestures and facial expressions, technologies used to communicate, etc.) are discarded and abstracted from. There is one exception to this: it is recorded which participants uttered a particular sentence. Also, we consider the possibility to accept, at a more advanced stage or our research, dialogue logs involving the use of graphical utterances (drawings) that are (in syntactic terms) translatable 1:1 to verbalizations.

2.3 Controlled Language

Formal and informal language may be hard to fully reconcile, but a classic meeting point between natural and formal language lies in similarities between the basics of their grammar and meaning, in particular in predicates and predication. After all, formal languages have historically been derived from their natural counterparts. It has since long been recognized that when we use simple, elementary sentences in natural language, we can relatively easily bridge the gap between formal and informal [20, 10], even if the bridge can only bear very light traffic. Such simple, elementary language can be described by a relatively simple grammar and can yet be realistically used in a modeling conversation (see, for example, [21]). We referred to it as *controlled language* [22]. Our notion of controlled language is related to that of *simplified English*; see [23]. The syntax (not necessarily the semantics!) of a controlled language is limited and fixed, and therefore it can expected to be both known to and agreed on by its various users [6, p26-31].

The competency of a participant [12] may then be defined as:

1. transform model into controlled language,
2. validate a description in controlled language.

It is assumed that a participant can express statements in this controlled language, but is also capable to express statements *about* that language. In system development, it is crucial to reach clarity and agreement about terminology, concepts, and sometimes syntax used in communication between members of the development team [4]. Controlled language can be used to reach clarity and agreement about any other type of language that might be used (for example, full-fledged natural language, schematic language, or (semi-)formal language). Thus it becomes possible to discuss any model through controlled language, but also to discuss the modeling languages –both formal and informal– through controlled language.

2.4 The Goal of Modeling

The goal of the modeling process can be described as: trying to reach a state where all participants agree that they have some degree of common understanding (for a similar stance in context of requirements engineering, see [24]). The participants try to derive from their personal semantics a group semantics. Participants will be convinced this goal has been achieved if they have validated their assumptions to contentment of everyone involved. For example, a system analyst will be convinced that the derived model is complete if the model has been validated against the real situation. In our view, this means that the domain expert, harboring the semantics of her conception of the universe, has positively responded to the controlled language description of the model provided by the system analyst, which may be rooted in a formal language. Various semantic functions come into play, but the shared, controlled language (which may be cooperatively constructed) performs an intermediary function.

The goal of the interaction can thus be seen as the construction of (1) a grammar for representations that are acceptable to all participants, and (2) semantic interpretation(s) in terms of some model(s). The grammar produced in the interaction is a generative device. It can also be used as a parsing device.

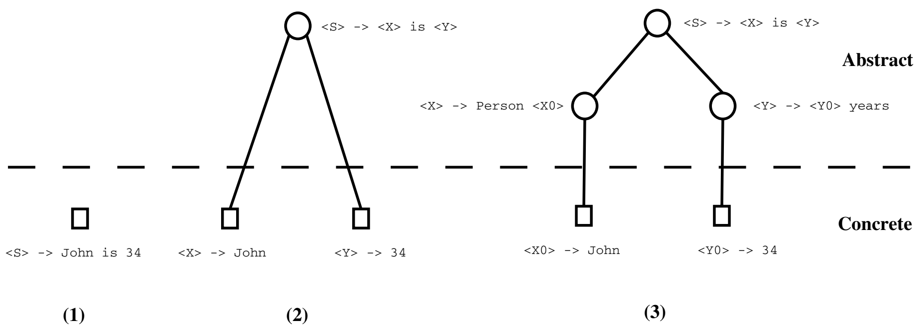


Fig. 2. Parsing levels

The grammar is correct when all sample sentences can be generated. From the point of view of the system analyst, the target model is restricted by the (formal) semantics of the modeling technique used. From the point of view of the domain expert, validation of the model may be seen as assigning meaning (interpretation) to the representations generated by the system analyst. A more symmetric way of putting this is that for each party (a), the other party (b) agrees with the controlled language statements provided by party (a).

Example 1. A simple sentence like *John is 34* is the initial statement verbalizing a fact occurring in the domain, provided by a domain expert (see case 1 in figure 2). The parsing structure of this sentence leads to case 2 of this figure. A simple modeling strategy is to defoliate parse trees, i.e. remove labels and constants like John and 34. These defoliated parse trees provide an example of the grammatical structure of the expert language. The leaves are concrete instances. During modeling, we are interested in acquiring the expert grammar, and therefore we are (only) interested in the defoliated parse trees.

The 3rd case provides a fully qualified version of the sentence in NIAM normal form. Fully qualified sentences are well suited as a basis for modeling. However, domain experts are more inclined to produce statements as in case (2). A main goal of a modeling dialog aiming for a NIAM model is to detect and resolve unqualified constants, hence asking and answering questions related to qualification.

Initially, the dialogue may use a modeling technique that accepts statements that do not convey all the information needed for a formalization. However, the system analyst will eventually require statements that match a more demanding and restrictive modeling technique like NIAM. During modeling, the model will have to migrate from the first (informal) to the second (formal) modeling technique.

2.5 Modeling as Interaction Between Viewers

The modeling process is seen as a goal-driven dialogue between a number of participants. Each participant is a viewer. The only way the participants can achieve their modeling goals is to communicate with each other, and remember and build on what has been discussed. An explicit way to do this would be to keep "modeling minutes" that are agreed on by the participants (figure 1). Taking a dialogue perspective on a modeling process is in line with a constructionist world view [25, 16].

As discussed, formal modeling can be succesfully captured by recording restricted aspects of communication. In this vein, in the NIAM method, the type of communication that takes place between modeling participants depicted as the *telephone methaphor* (two participants who communicate via a telephone line). Following this image, the modeling minutes consist of a recording or logbook of the telephone conversation. In order to capture the more rich and complex dialogue patterns in larger groups, we propose the so-called *chatbox metaphor*, assuming the participants communicate as in a chatbox. This is a real-time, teletype like communication channel that has become immensely popular among internet users; famous public applications of this type are, for example, Jabber, ICQ and MSN. In advanced use, chatbox conversations may branch off of (and rejoin) other chatbox conversations.

In view of the chatbox metaphor, the communication between the participants is assumed to be conducted entirely through a chatbox. If we view only the sentences of a particular participant, then it makes sense to interpret this view as a description of the model put forward by this participant. As the participant's model may evolve during the chat, obtaining the model from a participant involves the dynamics of dialogue, and is certainly not trivial.

For practical reasons, we will make some assumptions about the language that is used by the participants during the chat. The underlying controlled language model should be such that putting a sentence in some chosen normal form (for example, NIAM normal form) is an activity that does not require other skills from the participant than elementary knowledge of language and sufficient knowledge of the domain observed by this participant.

2.6 Some Types of Modeling Dialogue

As an illustration of various kinds of dialogue that may occur, consider the three modeling situations described below:

tabula rasa – This kind of modeling process is roughly comparable with the way in which a baby learns the basics of how the world is, from its parents and environment –with its developing language as a key item. *This is roughly the type of process compatible with the simplified domain expert - system analyst scenario discussed above. However, in that scenario the demands posed by a formal language of course imply a much stricter set of questions-to-answer than a child would harbor*

open mind – This kind of modeling process takes place when two people with their own, well-developed views on the world, are eager to learn about their mutual views –as reflected in their languages. *This type of modeling starts off with two unconnected representations/models, after which the commonalities between them may soon be discovered. This may or may not lead to a full reconciliation of the initial models in terms of a translation between them.*

colliding views – This kind of modeling process will occur when the participants have different views on the world, and the participants' priorities force them to maintain (part of) that view, while at the same time a mutually acceptable model is needed. This will usually lead to conflicts –modeling conflicts, possibly rooted in language conflicts, reflecting world view conflicts. *Modeling of this type will have to involve negotiation or argumentation about a common model. In some cases, one participant will impose his model upon the other participant; in others, one participant will be able to convince another by rational argumentation; in yet other cases, pure negotiation will take place: seeking a compromise both parties can agree with.*

Importantly, either the "open mind" or "colliding views" situation occurs whenever pre-existing conceptualizations or models (possibly, rooted in existing systems) are relevant to the modeling process, which in fact is very often the case.

The modeling strategies followed in the three types of conversation mentioned above are quite different. In addition, how the basic strategies are executed will strongly depend on the sort of model that is aimed for, and the modeling languages involved. It seems most realistic to start with investigating the "tabula rasa" type of dialogue and work up from there. The "colliding views" strategies are the most complex but seem representative of many real-life modeling situations in; arguably, they are ultimately the most interesting.

3 Modeling as a Dialogue

In the previous section (as illustrated by figure 1), we discussed our view on modeling as an exchange of statements between participants, in a modeling *dialogue.* In this section we introduce our core model for such dialogues-for-modeling.

3.1 Basics

As a starting point, we assume two participants in the modeling process, referred to as a and b respectively. Each participant harbors some volume of knowledge. For example, in a modeling process a domain expert has knowledge of the universe of discourse (i.e. some domain); as discussed in the previous section, we assume a domain expert to be fully knowledgeable: we do not question the validity of the expert's knowledge as such. We further assume the falsifiable basis (the test, as it were) for having acquired knowledge is *the capability to demonstrate it.* As a consequence, assuming participant p to have knowledge S_p corresponds to assuming p to be capable to somehow demonstrate the knowledge elements from S_p. Whether this demonstration is considered convincing depends on the judgment of the other participant, the initial contributor of the knowledge. This judgment will, generally, also depend on the goals and demands driving the modeling dialogue (as viewed by the participants).

Epistemically [14] we choose to view knowledge indeed as *knowledge* and not as *belief.* Thus, we abstract from such philosophical questions as whether or not the domain expert has knowledge that does not match reality. We collapse the notions of knowledge and belief to keep our current argument transparent.

As for the demonstration of knowledge, consider the following illustrative example (loosely based on [7]). If a teacher attempts to teach a student about something (for example, a historical episode), she tries to create in the student's mind a conception of the item taught that is equal or very similar to her own. How can she be convinced that the student has conceived (learned) the item? By asking for a demonstration of that knowledge. Such a demonstration can come in various forms. Discarding non-verbal demonstrations, we distinguish *exemplification* and *paraphrasing* as the most common techniques for convincing a teacher. These techniques can also be applied within modeling dialogues.

Traditional approaches to communication in modeling simply assume a participant who is willing to transfer domain knowledge to another participant who is eager to learn about this domain, i.e. to create a conception of it. The dialogue that brings about the transfer is assumed to be objectively meaningful: to be de-contextualized, making the message independent of time, location, and participant. This motivates a view of the modeling process restricted to the actual *symbols* used in communication, abstracting from the participants as such.

In the traditional view, subjectivity is only a relevant notion if there are disagreements between domain experts, which should be resolved. Contrary to the traditional view, we are neutral about the subjective conceptions of the participants involved in modeling.

We assume that participants are willing to communicate about their knowledge, and are willing to listen to the others. Thus, both a and b must be willing to take turns in playing the leading role of contributor and or the more passive role of receiver.

3.2 Characterizing the Dialogue

Let us first consider a the simple, "tabula rasa" view on knowledge transfer. We assume, without loss of generality, participants a and b to be in different roles (a is the contributor, b the receiver). The basic assumptions that must underly the dialogue are the following:

1. participant a is willing to transfer its knowledge to participant b,
2. participant b is willing to assimilate a's knowledge.

The above assumptions directly relate to the dialogue between a and b. The pragmatic assumptions with respect to the statements made in context of the modeling dialogue can be phrased as follows:

1. a has the intention to talk; a makes a statement s under the assumption that b seeks to know s.
2. b has the intention to listen; if a states s, then b assumes this is done with the ultimate intention to enable b know s.

Note that during modeling the participants will take turns in playing the leading role of contributor and or the more passive role of receiver. In the dialogue document the transferred statements will be registered, including the participants involved and the roles they play at that moment. This model allows dialogues to have sub-dialogues, with very specific goals that are sub-goals of the main dialogue. This is in line with the chatbox model for communication.

This analysis of the "tabula rasa" type of modeling dialogue can be extended to cover the "open mind" and "colliding views" types as introduced in the previous section:

3. a has the additional intention enable b to translate his representation to s; a makes a statement s under the assumption that b seeks map his representation to s where possible.
4. b has the additional intention to translate his representation to s; if a states s, then b assumes this is done to enable him to map his representation to s where possible.
5. a has the additional intention to negotiate, argue in favor of, or impose s on b; a makes a statement s under the assumption that b wants to negotiate, needs to be convinced of, or will have to be forced to accept s.
6. b has the additional intention to negotiate about s in view of his own, preferred representation, defend his own representation through argumentation, or resist accepting s instead of his own representation; if a states s, then b assumes this is done in order to negotiate, argue in favor of, or impose s upon him.

We consider understanding of the above intentions, and the strategies that follow from them, to be fundamental to the understanding of the modeling process. Matters may be complicated by unawareness of one participant of some goal or strategy of another participant; also, various goals and strategies may become entangled.

4 Conclusion and Future Work

We have presented a fundamental view on (formal) modeling rooted in knowledge creation and exchange, in which communication plays a central role. We have argued that to achieve a high quality combination of validity and verifiability in models, we need to look not only at the product, but *specifically* also at the process of modeling. In line with our communicative approach, the modeling process is viewed as a dialogue between participants. We have described a first, general analysis of the essential properties of modeling dialogues. In particular we discussed the central role controlled language can play in modeling dialogues, and the basic underlying intentions of such dialogues, rooted in the sharing, translation, negotiation, argumentation, and imposing of (participant-based) knowledge representations. This should provide a good starting point for more detailed, domain-specific or application-specific exploration of modeling dialogues, with as a central goal the discovery of modeling strategies and optimal selection of such strategies depending on the goals for particular situations.

As possible domains of application of the controlled use of modeling strategies, the following flavors of modeling seem particularly interesting: domain modeling, information modeling, architecture modeling, ontological modeling, and interactive querying. We plan on focused research activities in all of these areas.

Possibly, the range of application areas can be extended to include more complex forms of modeling, such as software modeling, formal business rules specification, and numerous AI applications.

Validation and improvement of our theoretical framework will be a crucial aspect of our further research. After further investigating the plausibility and soundness of our framework, we will test its validity by starting a substantial experimentation programme to validate our initial theory, the chatbox metaphor, and the use of controlled language. We thus intend to lay an empirical foundation under our exploration of the basic dynamics of modeling and the use of controlled language therein. We intend to start our experiments by investigating "tabula rasa" type modeling, and quickly move into "open mind" modeling. Understanding and improving "opposite views" modeling is more challenging, and may be successful only in the long run, but is also the Main Prize. Core focus of our theory development will be on eliciting, describing, and testing *strategies* for formal modeling (possibly also other forms of modeling).

One of our long term objectives is to investigate ways of developing a new brand of CASE tools that involves the interactive monitoring and guidance of some dedicated (i.e. situationally fitted) modeling process, integrated with the IS development process at large. Two factors underly this idea: solving the modeling bottleneck and improving model quality and grounding. The new brand of case tools can be expected to be a blend of classic, product-oriented CASE tools on the one hand, and cooperative, interactive dialogue systems on the other (probably involving issues as studied in the field of Computer Supported Cooperative Work or CSCW).

We eventually hope to extend the range of our experiments by providing an increasingly attractive digital environment for people to use during modeling, providing added value for the participants as well as data for the researchers, and enabling insightful interaction between the two groups.

References

1. Proper, H., Verrijn-Stuart, A., Hoppenbrouwers, S.: Towards utility-based selection of architecture-modelling concepts. In Hartmann, S., Stumptner, M., eds.: Proceedings of the Second Asia-Pacific Conference on Conceptual Modelling (APCCM2005), Newcastle, New South Wales, Australia. Volume 42 of Conferences in Research and Practice in Information Technology Series., Sydney, New South Wales, Australia, Australian Computer Society (2005) 25–36.
2. Halpin, T.: Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design. Morgan Kaufman, San Mateo, California, USA (2001).
3. Proper, H., Bleeker, A., Hoppenbrouwers, S.: Object-role modelling as a domain modelling approach. In Grundspenkis, J., Kirikova, M., eds.: Proceedings of the Workshop on Evaluating Modeling Methods for Systems Analysis and Design (EMMSAD'04), held in conjunctiun with the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004). Volume 3., Riga, Latvia, EU, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU (2004) 317–328.

4. Bleeker, A., Proper, H., Hoppenbrouwers, S.: The role of concept management in system development – a practical and a theoretical perspective. In Grabis, J., Persson, A., Stirna, J., eds.: Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU (2004) 73–82.
5. Lankhorst, M., others: Enterprise Architecture at Work : Modelling, Communication and Analysis. Springer, Berlin, Germany, EU (2005).
6. Hoppenbrouwers, S.: Freezing Language; Conceptualisation processes in ICT supported organisations. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (2003).
7. Pask, G.: Conversation, cognition, and learning: a cybernetic theory and methodology. Elsevier (1975).
8. Veldhuijzen van Zanten, G., Hoppenbrouwers, S., Proper, H.: System Development as a Rational Communicative Process. In Callaos, N., Farsi, D., Eshagian-Wilner, M., Hanratty, T., Rish, N., eds.: Proceedings of the 7th World Multiconference on Systemics, Cybernetics and Informatics. Volume XVI., Orlando, Florida, USA (2003) 126–130.
9. Wijers, G., Heijes, H.: Automated Support of the Modelling Process: A view based on experiments with expert information engineers. In Steinholz, B., Sølvberg, A., Bergman, L., eds.: Proceedings of the Second Nordic Conference CAiSE'90 on Advanced Information Systems Engineering. Volume 436 of Lecture Notes in Computer Science., Stockholm, Sweden, EU, Springer-Verlag, Berlin, Germany, EU (1990) 88–108.
10. Hoppenbrouwers, J., Vos, B.v.d., Hoppenbrouwers, S.: Nl structures and conceptual modelling: Grammalizing for KISS. Data & Knowledge Engineering **23** (1997) 79–92.
11. Proper, H., Hoppenbrouwers, S.: Concept evolution in information system evolution. In Gravis, J., Persson, A., Stirna, J., eds.: Forum proceedings of the 16th Conference on Advanced Information Systems 2004 (CAiSE 2004), Riga, Latvia, EU, Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, EU (2004) 63–72.
12. Frederiks, P., Weide, T.v.d.: Information modeling: the process and the required competencies of its participants. In Meziane, F., Métais, E., eds.: 9th International Conference on Applications of Natural Language to Information Systems (NLDB 2004). Volume 3136 of Lecture Notes in Computer Science., Manchester, United Kingdom, EU, Springer-Verlag, Berlin, Germany, EU (2004) 123–134.
13. Pinker, S.: The Language Instinct. Allen Lane/Penguin Press, London, United Kingdom, EU (1994).
14. Meyer, J.J., Hoek, W.v.d.: Epistemic Logic for AI and Computer Science. Cambridge University Press, Cambridge, United Kingdom, EU (1995).
15. Peirce, C.: Volumes I and II – Principles of Philosophy and Elements of Logic. Collected Papers of C.S. Peirce. Harvard University Press, Boston, Massachusetts, USA (1969).
16. Krogstie, J.: A semiotic approach to quality in requirements specifications. In Kecheng, L., Clarke, R., Andersen, P., Stamper, R., Abou-Zeid, E.S., eds.: IFIP TC8/WG8.1 Working Conference on Organizational Semiotics – Evolving a Science of Information Systems, Montréal, Québec, Canada, Kluwer Academic Publishers, Dordrecht, The Netherlands, EU (2002) 231–250.

17. The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE: Recommended Practice for Architectural Description of Software Intensive Systems. Technical Report IEEE P1471-2000, The Architecture Working Group of the Software Engineering Committee, Standards Department, IEEE, Piscataway, New Jersey, USA (2000).
18. Chandler, D.: The Act of Writing: a Media Theory Approach. University of Wales, Aberystwyth, United Kingdom, EU (1995).
19. Wood-Harper, A., Antill, L., Avison, D.: Information Systems Definition: The Multiview Approach. Blackwell Scientific Publications, Oxford, United Kingdom, EU (1985).
20. Frederiks, P.: Object-Oriented Modeling based on Information Grammars. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, EU (1997).
21. Rolland, C., Souveyet, C., Ben Achour, C.: Guiding goal modeling using scenarios. IEEE Transactions on Software Engineering **24** (1998) 1055–71.
22. Schwitter, R.: Controlled Natural Languages. Centre for Language Technology, Macquary University, Sydney, Australia. (2004).
23. Farrington, G.: An Overview of the International Aerospace Language. (1996).
24. Pohl, K.: The three dimensions of requirements engineering: a framework and its applications. Information Systems **19** (1994) 243–258.
25. Berger, P., Luckmann, T.: The Social Construction of Reality. Doubleday, New York, New York, USA (1966).

How to Tame a Very Large ER Diagram (Using Link Analysis and Force-Directed Drawing Algorithms)

Yannis Tzitzikas[1] and Jean-Luc Hainaut[2]

[1] University of Crete and FORTH-ICS, Heraklion, Greece
[2] Institut d'Informatique, University of Namur (F.U.N.D.P.), Belgium
tzitzik@csi.forth.gr, jlh@info.fundp.ac.be

Abstract. Understanding a large schema without the assistance of persons already familiar with it (and its associated applications), is a hard and very time consuming task that occurs very frequently in reverse engineering and in information integration. In this paper we describe a novel method that can aid the understanding and the visualization of very large ER diagrams that is inspired by the link analysis techniques that are used in Web Searching. Specifically, this method takes as input an ER diagram and returns a smaller (top-k) diagram that consists of the major entity and relationship types of the initial diagram. Concerning the drawing of the resulting top-k graphs in the 2D space, we propose a force-directed placement algorithm especially adapted for ER diagrams. Specifically, we describe and analyze experimentally two different force models and various configurations. The experimental evaluation on large diagrams of real world applications proved the effectiveness of this technique.

1 Prologue

It has been recognized long ago that the usefulness of conceptual diagrams (e.g. ER/UML diagrams) degrades rapidly as they grow in size. Understanding a large schema without the assistance of persons already familiar with it, is often a nightmare. Unfortunately, large conceptual schemas are becoming more and more frequent. The integration of information systems, the development or reverse engineering of large systems, the usage of ERP (the SAP database includes 30.000 tables) and the development of the Semantic Web (structured into ontologies potentially including dozens of thousands of classes) naturally lead to the building of very large schemas. Although a good drawing of a conceptual schema could aid its understanding, and several approaches for automatic placement have been already proposed (e.g. see [33,23,8,28,10]), it is a widely accepted opinion that the automatic layout facilities offered by current UML-based CASE tools are not satisfactory even for very small diagrams (for more see [14])[1].

[1] General graph drawings algorithms (e.g. see [2]) usually make some assumptions that are not always valid in conceptual graphs.

Consequently, the vast majority of layouts created today are done "by hand"; a human designer makes most, if not all, of the decisions about the position of the objects to be presented [26]. The visualization and drawing of large conceptual graphs is even less explored. The classical hierarchical decomposition techniques that are used for visualizing large plain graphs (for a survey see Chapter 3 of [29]), have not been applied or tested on conceptual graphs. Consequently, only manual collapsing mechanisms (like those described in [22]) are currently available for decreasing the visual clutter and for aiding the understanding of big conceptual graphs. In addition, the techniques that have have been proposed for reducing the size of a conceptual graph (in order to aid its comprehension), specifically ER clustering, either require human input [15,34,18,7], or they are automated but not tested on large conceptual schemas [30,1].

We decided to devise an automatic technique for identifying the major entity and relationship types of a very large conceptual graph as a means for facilitating its understanding. As Link Analysis has been proved very successful in Web Searching [6,25] and recently in several other application domains [19,20], we decided to design a similar in spirit technique for one very common and important kind of conceptual schemas, namely Entity-Relationship (ER) diagrams [9]. Concerning the drawing in the 2D space of the resulting top-k graphs, we describe a force-directed placement algorithm especially adapted for ER diagrams. Specifically, we describe and analyze experimentally two different force models and various configurations.

Both of the techniques that are presented in this paper can be applied not only to ER diagrams, but also on other kinds of conceptual graphs. The Semantic Web is one interesting application area because it is founded on ontologies (potentially including dozens of thousands of classes) that are exchanged in a layout-missing format. In this context, the provision of top-k diagrams and automatic layout services is very important as they can aid understanding that is very important for accomplishing tasks like semantic annotation, creation of ontology mappings, ontology specialization, etc.

This paper is structured as follows: Section 2 describes Link Analysis for ER diagrams, Section 3 introduces force-directed placement algorithms for ER diagrams and finally, Section 4 concludes this paper.

2 Link Analysis for ER Diagrams

Our objective here is to identify the major entity and relationship types of a very large ER diagram in order to facilitate its understanding. We designed a PageRank [6] style scoring method because PageRank is described in terms on the entire Web, while HITS [25] is mainly applied on small collections of pages (say those retrieved in response to a query). We view an ER diagram as a triple (E, R, I) where $E = \{e_1, ..., e_N\}$ denotes the entity types, $R = \{r_1, ..., r_m\}$ denotes the relationship types, and I the *isA* relationships over E (i.e. $I \subseteq E \times E$). For any given $e \in E$, we shall use $conn_R(e)$ to denote those entity types that are connected with e through relationship types, $conn_{sb}(e)$ denote the direct subtypes of e, and $conn_{sp}(e)$ the direct supertypes of e. We shall also use the following shorthands: $conn_I(e) = conn_{sb}(e) \cup conn_{sp}(e)$ and $conn(e) = conn_R(e) \cup conn_I(e)$. Since two entity types may be connected with more than one relationship types we consider $conn_R(e)$ as a *bag* for being able to record duplicates. In addition, we shall use $attrs(e)$ to denote the attributes of an entity type e.

Now the *score* (or EntityRank) of an entity type e in E, denoted $Sc(e)$, can be defined as follows:

$$Sc(e) = q/N + (1-q) * \sum_{e' \in conn_R(e)} \frac{Sc(e')}{|conn_R(e')|} \tag{1}$$

where q stands for a constant less than 1 (e.g. 0.15 as in the case of Google)[2] One can easily see that the above formula simulates a random walk in the schema. Under this view each relationship type is viewed as a *bidirectional* transition and the probability of randomly jumping to an entity type is the same for all entity types (i.e. q/N). The resulting scores of the entity types correspond to the stationary probabilities of the Markov chain.

A rising question here is how we can incorporate n-ary ($n > 2$) relationship types into the aforementioned model. This can be achieved by replacing each n-ary relationship type ($n > 2$) over n entity types $e_1, ..., e_n$ by $n(n-1)/2$ binary relationship types that form a complete graph[3] over $e_1, ..., e_n$. Consequently, an n-ary relationship type is viewed as $n(n-1)/2$ binary relationships.

An alternative approach is to assume that the probability of jumping to a random entity is not the same for all entities, but it depends on the number of its attributes. In this case we define the *score* (or BEntityRank) of an entity type e in E as follows:

$$Sc(e) = q\frac{|attrs(e)|}{|Attr|} + (1-q) \cdot \sum_{e' \in conn_R(e)} \frac{Sc(e')}{|conn_R(e')|} \tag{2}$$

where $Attr$ denotes the set of all attributes of all entity types (i.e. $Attr = \cup\{\ attrs(e) \mid e \in E\}$). This particular formula simulates a user navigating randomly in the schema who jumps to a random entity e with probability $\frac{q|attrs(e)|}{|Attr|}$ or follows a random relationship type (on the current entity). The probability $\frac{|attrs(e)|}{|Attr|}$ corresponds to the probability of selecting e by clicking randomly on a list that enumerates the attributes of all entity types of the schema.

The linear algebra version of EntityRank and BEntityRank is given in Appendix A.

Let's now discuss the differences between link analysis for ER diagrams and link analysis for the Web. Firstly, Web links are directed, while relationship

[2] If we set $q = 0.15$ or below then an iterative method for computing the scores (e.g. the Jacobi method) requires at most 100 iterations to convergence.

[3] A complete graph is a graph in which each pair of graph vertices is connected by an edge.

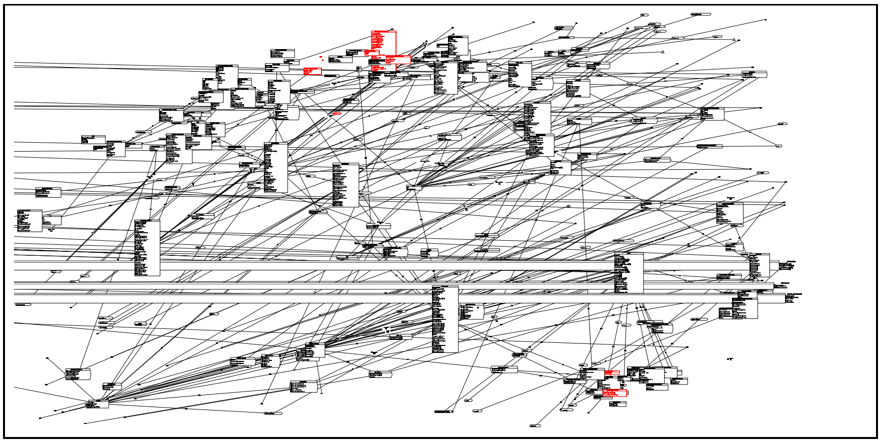


Fig. 1. Excerpt ($< 10\%$) of a large ER diagram drawn using a force-directed placement algorithm

types are not directed thus the latter are considered as bidirectional transitions. Secondly, we do not collapse all relationships types between two entity types into one (as it is done with Web links), and this is the reason why we consider $conn_R$ as a bag. Thirdly, in ER diagrams we should count "self hyperlinks" (i.e. cyclic relationship types), although the Web techniques ignore them. For example, consider a schema consisting of two entity types $\{\texttt{Person}, \texttt{City}\}$ and two relationship types $\{\texttt{Person}$ *lives* $\texttt{City}$, $\texttt{Person}$ $fatherOf$ $\texttt{Person}\}$. If we ignore the relationship type $fatherOf$, then both `Person` and `City` would be equally scored, a not so good choice. At last, although in the Web link analysis is exploited mainly for ranking the results of retrieval queries, in our case we don't need just a ranked list of entity types, but rather another *diagram* that consists of the major entity and relationship types.

We have implemented and evaluated the above scoring schemes into the $\texttt{DB} - \texttt{MAIN}$ CASE tool (for more see [17,21]). The designer provides a threshold *per* between 0 and 100. Subsequently, all entity types with score lower than $per\% * ScMax$, where $ScMax$ denotes the highest score, disappear. Controlling the visibility of entity types according to their score, and not according to their rank, is preferred as it better handles ties. Concerning relationship types, only those that connect the visible entity types are displayed. The computation of the scores takes only some seconds on a conventional PC. Specifically, to compute the scores we use the Jacobi iterative algorithm. We have noticed that 50 iterations give quite stable orderings and their application on schemas with 1000 entity and relationship types takes less than 2 seconds in a conventional PC.

Figure 1 shows a very small part of the ER diagram of a Belgian distribution company. Though the schema comprises about 450 nodes and 800 edges only, the layout is definitely useless for understanding the schema and the corresponding application.

Now Figure 2 shows in micrography the diagram of the top-11 entity types of the schema of Figure 1 according to EntityRank (for reasons of space, the

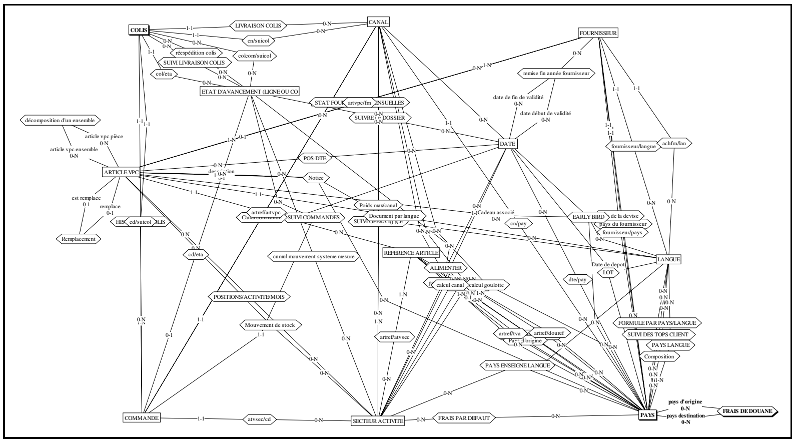


Fig. 2. The diagram of the top-11 entity types of the schema of Figure 1

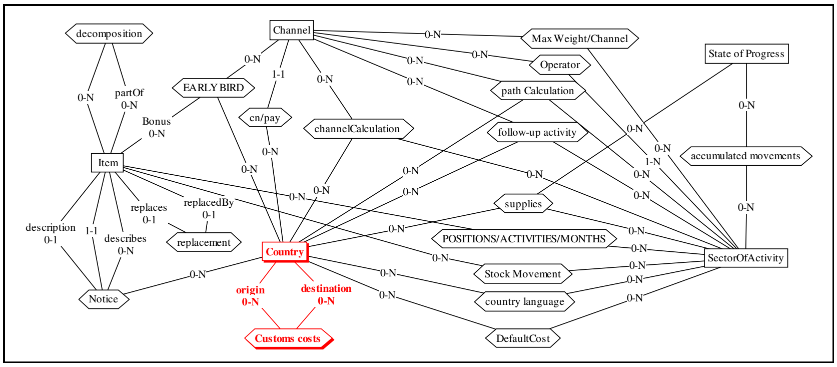


Fig. 3. The diagram of the top-5 entity types of the schema of Figure 1

attributes are not displayed in this figure). Although this schema has 54 relationship types it is extremely more easy to visualize, and thus to understand, than the original schema. Of course, one user could start from even smaller diagrams. For instance, Figure 3 shows the graph of the top-5 entity types of the same schema. It indeed contains the major entity types of this application and a user can immediately understand the application domain of this schema.

In case of diagrams with big *isA* hierarchies, some entity types, although major, may not receive high scores because their relationship types are scattered in several subentity types. To handle this case, we introduced a (optional) preprocessing step in which each *isA* hierarchy of the schema is collapsed into one entity type that collects all the attributes and relationship types of its subentity types.

Concerning evaluation, at first we have to note that the evaluation of the effectiveness of link analysis techniques for ER diagrams (for conceptual graphs in general), is more difficult than in the case of Web. In the latter case, it is not so hard to judge whether the top ranked pages are indeed relevant to the submitted query. However, in the case of conceptual graphs, one has to know well in advance the conceptual graph in order to judge whether the resulting small graph indeed contains the major concepts of the conceptual graph and of its underlying domain. Inevitably, the most reliable evaluation of such techniques can be done only in already known conceptual graphs. For this reason we applied this method to almost every conceptual schema that the DBMAIN group has produced the last 3 years. This was a quite representative test bed as it includes big schemas of existing (and non artificial) real world applications. We always obtained surprisingly good results. For reasons of space we cannot report here the exact results of the evaluation of EntityRank (and BEntityRank) using metrics coming from the area of IR (for more [35]). In addition, it is an advantage that the proposed formulas for link analysis are mathematically founded and that the underlying model (the random walk model) is quite relevant to *browsing*, i.e. to the most widely used method for understanding a conceptual graph. At last, another evidence that link analysis is indeed appropriate for ER diagrams is that all large schemas that we have tested have a small set of elements, usually less than 5% of the total ones, whose scores are significantly higher than the rest. This at least indicates that big ER diagrams tend to have a well connected kernel which, at least in our experiments, always comprised the more important concepts of the application domain.

3 Automatic ER Drawing

For drawing automatically the top-k ER diagrams that are derived by the previous technique, we shall view them as mechanical systems. Below we present two force models that combine the *spring-model* (proposed and developed in [13,24,16]) with the *magnetic-spring model* (proposed in [32,31]) in a way that is appropriate for ER diagrams.

3.1 Force Model A

Here entity types are viewed as equally charged particles which *repel* each other. Relationship types and *isA* relationships are viewed as springs that *pull* their adjacent entity types. Moreover, we assume that the springs that correspond to *isA* links are all magnetized and that there is a global magnetic field that acts on these springs. Specifically, this magnetic field is parallel (i.e. all magnetic forces operate in the same direction) and the *isA* springs are magnetized unidirectionally, so they tend to align with the direction of the magnetic field, here upwards. Figure 4 illustrates this metaphor.

Under the above force model, the force on a entity type e_i is given by:

$$F(e_i) = \sum_{e_j \in conn(e_i)} f(e_j, e_i) + \sum_{e_j \in E, e_j \neq e_i} g(e_j, e_i) + \sum_{e_j \in conn_I(e_i)} h(e_j, e_i) \qquad (3)$$

where: $f(e_j, e_i)$ is the force exerted on e_i by the spring between e_j and e_i (note that e_i and e_j are connected by a relationship type or an *isA* link), $g(e_j, e_i)$ is

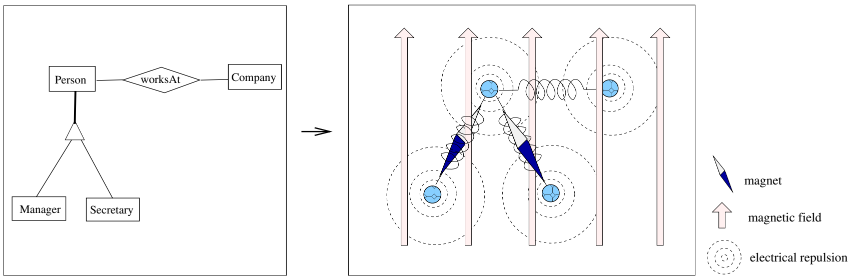


Fig. 4. Viewing an ER diagram as a mechanical system

the electrical repulsion exerted on e_i by the entity type e_j, and $h(e_j, e_i)$ is the rotational force exerted on e_i by the entity type e_j (here e_i and e_j are connected by an isA link).

Figure 5 gives some indicative examples that explain the role of the forces f, g and h. Specifically, figure (a) justifies the spring force, figure (b) justifies the electrical repulsion and shows that high electrical repulsion (high K^e) results in symmetrical drawings, and figure (c) illustrates how the magnetic field can be used in order to obtain the classical top-down drawings for isA hierarchies.

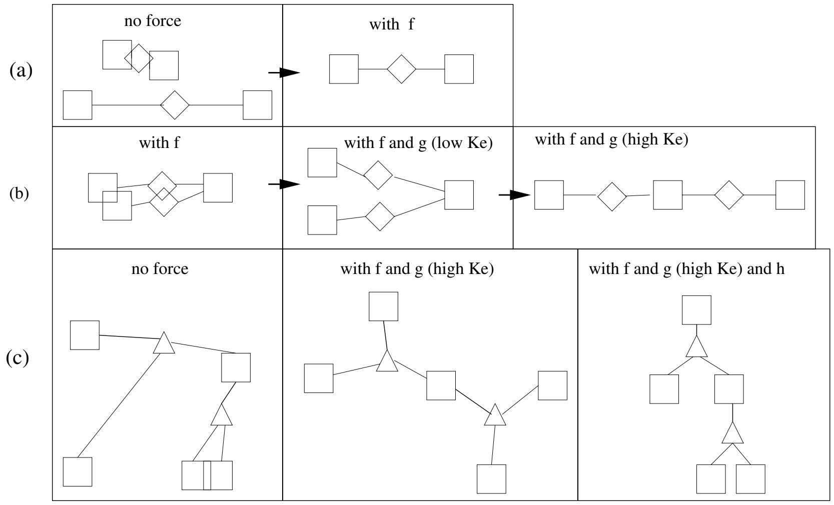


Fig. 5. Forces and ER Drawings

The spring force $f(e_j, e_i)$ follows Hooke's law, i.e. it is proportional to the difference between the distance between e_j and e_i and the zero-energy length of the spring. Let $d(p, p')$ denote the Euclidean distance between two points p and p' and let $p_i = (x_i, y_i)$ denote the position of an entity type e_i. The x component of the force $f(e_i)$ is given by:

$$f_x(e_i) = \sum_{e_j \in conn(e_i)} K^s_{i,j}(d(p_i, p_j) - L_{i,j}) \frac{x_j - x_i}{d(p_i, p_j)}$$

where $L_{i,j}$ denotes the natural (zero energy) *length* of the spring between e_i and e_j. This means that if $d(p_i, p_j) = L_{i,j}$ then no force is exerted by the spring between e_i and e_j. Now $K^s_{i,j}$ denotes the *stiffness* of the spring between e_i and e_j. The larger the value of $K^s_{i,j}$, the more tendency for the distance $d(p_i, p_j)$ to be close to $L_{i,j}$. The y component of the force $f(e_i)$ is defined analogously.

The electrical force $g(e_j, e_i)$ follows an inverse square law. The x component of the force $g(e_i)$ is given by:

$$g_x(e_i) = \sum_{e_j \in E, e_j \neq e_i} \frac{K^e_{i,j}}{d(p_i, p_j)^2} \frac{x_i - x_j}{d(p_i, p_j)}$$

where $K^e_{i,j}$ is used to control the *repulsion strength* between e_i and e_j. The y component of the force $g(e_i)$ is defined analogously.

The magnetic force $h(e_j, e_i)$ depends on the angle between the isA spring (that connects e_j and e_i) and the direction of the magnetic field and it induces a rotational force on that spring. For example, Figure 6 shows an isA link between e_i and e_j and the exerted forces on e_i and e_j due to the magnetic field. The x and y components of the magnetic force $h(e_i)$ are given by:

$$h_x(e_i) = \sum_{e_j \in conn_{sp}(e_i)} K^m \frac{x_j - x_i}{L_{i,j}} + \sum_{e_j \in conn_{sb}(e_i)} K^m \frac{x_j - x_i}{L_{i,j}}$$

$$h_y(e_i) = \sum_{e_j \in conn_{sp}(e_i)} K^m \frac{L_{i,j} + y_j - y_i}{L_{i,j}} - \sum_{e_j \in conn_{sb}(e_i)} K^m \frac{L_{i,j} + y_i - y_j}{L_{i,j}}$$

where K^m is used to control the strength of the magnetic field.

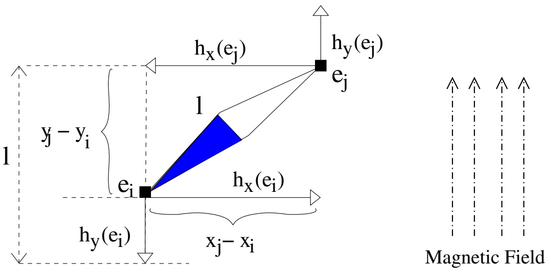


Fig. 6. Magnetic forces and isA links

The x and y components of the composed force $F(e_i)$ on an entity type e_i are obtained by summing up, i.e.: $F_x(e_i) = f_x(e_i) + g_x(e_i) + h_x(e_i)$ and $F_y(e_i) = f_y(e_i) + g_y(e_i) + h_y(e_i)$.

As in the link analysis technique, we view an n-ary relationship type as $n(n-1)/2$ springs.

3.2 Force Model B

One weakness of the above model is that the resulting drawings can have several overlaps. The reason is that: (a) there is no repulsion among relationship types, and (b) there is no repulsion between entity and relationship types. Figure 7 illustrates this problem. This drove us to introduce a different force model where each relationship type is viewed as a particle too. Clearly, the resulting electrical repulsion discourages the creation of overlaps (between entity and relationship types, or between relationship types themselves). Notice that according to this view, a relationship type does no longer correspond to one spring. Specifically, the particle of a relationship type over k entity types, is connected with one spring with each one of them. The forces on entity types and relationship types are computed analogously to the force model A.

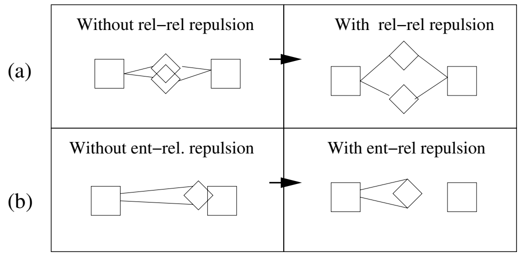


Fig. 7. Forces and ER Drawings

3.3 The Drawing Algorithm

We can reach a drawing by an algorithm that simulates the mechanical system. Such a algorithm would seek for a configuration with locally minimal energy, i.e. a drawing in which the forces on each node is zero. A variety of numerical techniques can be used to find an equilibrium configuration, and thus the final drawing. We have adopted the iterative method based on the method proposed in [13]. At first the nodes are placed at random positions. At each iteration, the force on each node is computed and then the node is moved towards the corresponding direction by a small amount proportional to the magnitude of the force. This can be continued until convergence, but we can also limit the number of iterations.

Note that if we would like to find a drawing that corresponds to a state with globally minimal energy, then we would have to resort to very general optimization methods. For instance, a method based on simulated annealing is proposed in [11], while an approach based on genetic algorithms is described in [4]. However the computational complexity of these techniques turns them not very appropriate for interactive design systems. In addition, and according to the results of the extensive empirical analysis of several force-directed algorithms (including globally minimal energy algorithms) upon plain graphs that are reported in [3], there is no universal winner and the general approach is to try several methods and choose the best.

3.4 Experimental Evaluation

We have investigated and evaluated all these issues in the context of the CASE tool DB-MAIN. The specification of the parameters L, K^s, K^e and K^m is not a trivial task as these parameters determine in a high degree how the final drawing will look like. One flexibility of the proposed approach is that we can adjust the spring length ($L_{i,j}$), spring stiffness($K^s_{i,j}$) and electrical repulsion($K^e_{i,j}$), in order to customize the appearance of the drawing according to the semantics of the ER diagram constructs. For instance, as it is desirable to keep the nodes of an *isA* hierarchy close enough and since between any two isA-related entity types we only have to draw a line (and not any hexagon-enclosed string), we can use a smaller length for *isA*-springs than that of relationship-springs. In any case, the user can change their value at run-time.

Figure 8 shows one drawing obtained by the algorithm using low electrical repulsion. Although the *isA* hierarchy is drawn as a top-down drawing and we have no overlaps, this drawing is not satisfying because a designer would hardly manually place into the space occupied by an *isA* hierarchy an entity type that does not belong to that hierarchy. After we increased the repulsion and the magnetic field we never faced again such a drawing. The lesson learned is that high repulsion not only results in symmetrical drawings but its combination with a strong magnetic field results in clear *isA* drawings. Another drawing of a diagram with 4 *isA* hierarchies that is derived by the algorithm according to force model A, is shown in Figure 9.

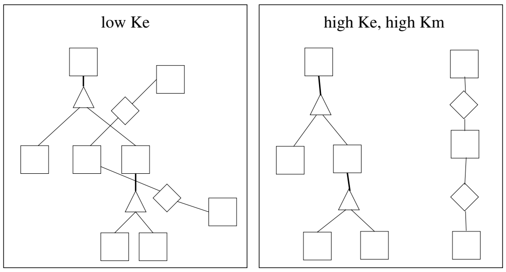


Fig. 8. How to obtain clean *isA* drawings

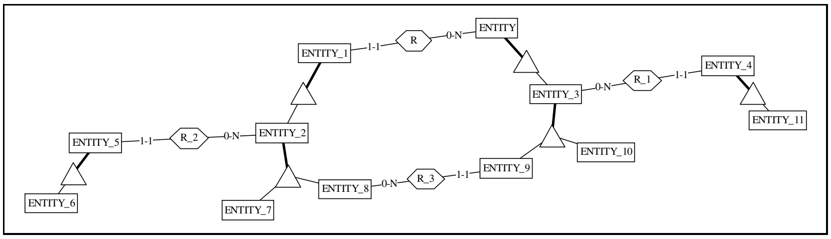


Fig. 9. A drawing of a diagram with 4 isA hierarchies according to force model A

A more complex case is shown in Figure 10. Figure 10.(a) shows a manually placed diagram where all subentity types have been placed at the outer part of the drawing. Figure (b) shows the drawing obtained according to force model B. Notice that every *isA* hierarchy now corresponds to a top-down drawing and that the entire drawing is symmetrical and satisfying.

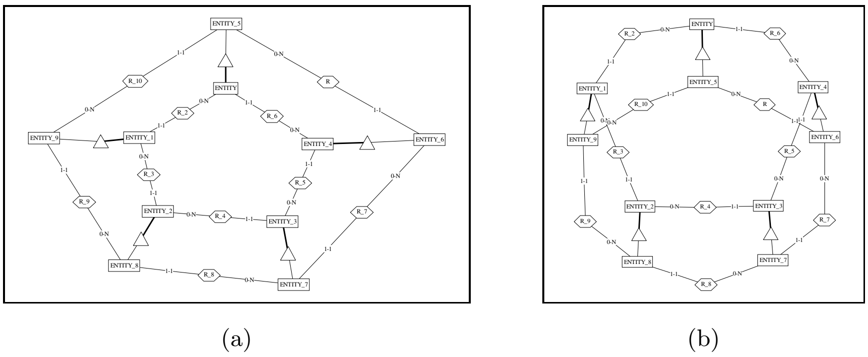


Fig. 10. Drawing of a diagram with several IsA hierarchies. (a): manual drawing where isA links are not vertical. (b): drawing obtained according to force model B.

The experimental evaluation showed that the drawings according to force model A suffer from overlaps, while those according to force model B have a few (or none) overlaps. The difference between force model A and force model B is even more evident in dense diagrams. Figure 11 shows the drawings obtained by these two models when applied on the top-5 ($|E| = 5, |R| = 17$) diagram of Figure 3. Again, the second drawing is evidently better. A noteworthy remark here is that the second diagram is more clear and intuitive than the manually specified layout that is shown in Figure 3. This indicates that in certain cases (at least when the diagram is very dense) the automatically-derived drawings can be better than the manually drawn.

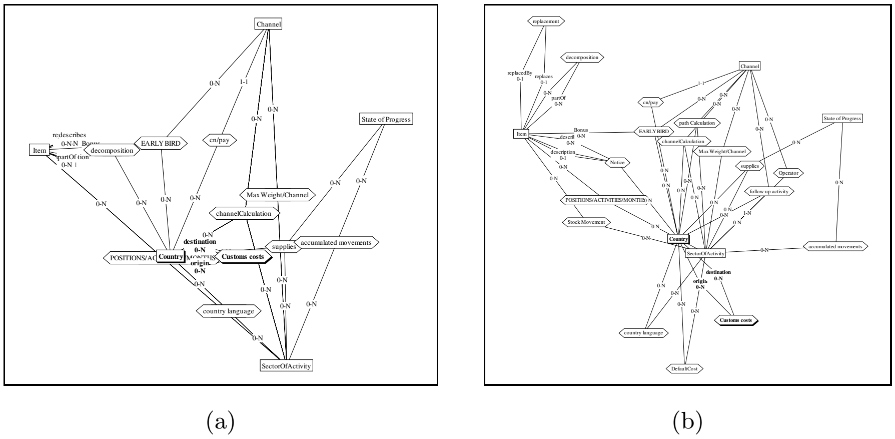
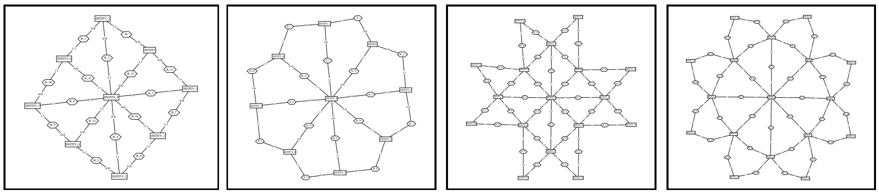


Fig. 11. Force model A vs force model B on a diagram with $|E| = 5$ and $|R| = 17$

However, we have to note that force model B has two weaknesses comparing to force model A: (i) it is computational more expensive, and (ii) in the resulting drawings the tentacles of binary relationship types are in many cases unnecessarily not aligned. This is evident in Figure 12. Although this is not a major problem it is an issue for further research.



(a) (b) (c) (d)

Fig. 12. Force model A vs force model B. (a): force model A. (b): force model B. (c): force model A. (d): force model B.

Figure 13 shows the automatic layout obtained for the top-11 diagram (that was presented in Figure 2). The high relative number of relationships makes the drawing almost unreadable. This example suggests that we should take into account the density of a diagram, in order to reach readable and clear drawings.

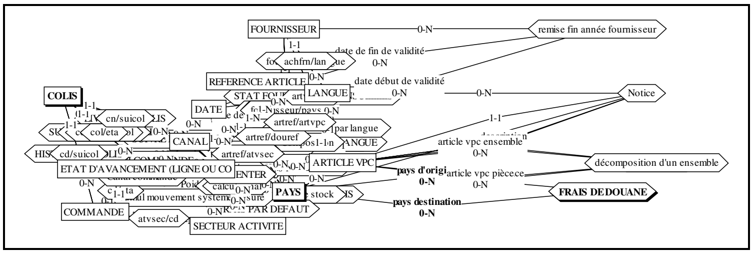
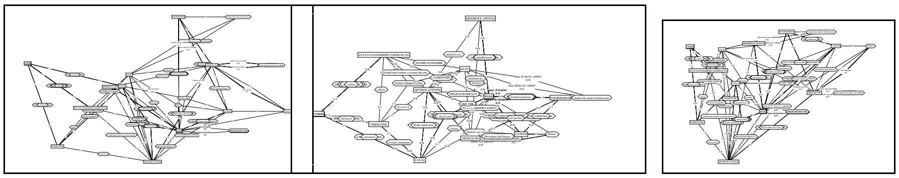


Fig. 13. Dense diagram drawing

Roughly, we could handle dense diagrams by considering: (i) larger springs, (ii) higher electrical repulsion, (iii) less stiff springs. For example, and assuming force model A, Figure 14.(a) shows the drawing obtained with spring length $L' = 5L$, Figure 14.(b) shows the drawing obtained with $K^{e'} = 100K^e$, and Figure 14.(c) shows the drawing obtained with $K^{s'} = K^s/10000$. Indeed, all are better than the original drawing shown in Figure 13. Another simple method that is both effective and efficient is to scale up the entire drawing (i.e. multiply each coordinate by a constant $c > 1$). Nevertheless, an issue that is worth further research is to investigate the effectiveness of local-density adaptations, e.g. to adapt the spring lengths according to the local density of the graph. EntityRank and BEntityRank scores could be exploited for this purpose.

As a final remark note that the above drawing techniques can be applied for drawing the structural part of ontologies expressed in RDFS [5] and OWL [12]. The only difference is that RDFS supports property specialization which



(a) (b) (c)

Fig. 14. (a): larger springs; (b): higher repulsion; (c) less stiff springs

however will be handled correctly due to the magnetic field that is applied on specialization/generalization links (also indicated by Figure 9).

4 Conclusion

We described a novel method for identifying the major elements of an ER diagram that is based on link analysis. This method can significantly aid (a) the *understanding*, (b) the *visualization*, and (c) the *drawing* of very large schemas. The proposed technique can elevate automatically the major elements and allows exploring the schema gradually: from the more important elements to the less. Consequently, it can be very useful in reverse engineering and in information integration. Moreover, the scores can be exploited for ordering the schema elements that match a keyword query of the user. In addition, and given the inability to produce automatically aesthetically satisfying layouts for large schemas, the small (top-k graphs) that can be derived by this technique can be visualized effectively and this is very useful during communication (e.g. between designers and application programmers or in requirements engineering and training). For this purpose we investigated a force-directed drawing algorithm and evaluated two different force models upon several conceptual schemas of real applications. For small and medium sized diagrams the results were satisfying in most of the cases. In the rest cases, human intervention (moving, nailing) and rerun of the drawing algorithm could rectify the problems.

Acknowledgements

The first author wants to thank Tonia Dellaporta for the several fruitful and really enjoyable discussions on this issue. Also many thanks to Jean-Rock Maurisse, Anne-France Brogneaux and Jean Herald for their help on using the `DB-MAIN` toolkit.

References

1. Jacky Akoka and Isabelle Comyn-Wattiau. "Entity-Relationship and Object-Oriented Model Automatic Clustering". *Data and Knowledge Engineering*, 20(2):87–117, 1996.

2. Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis Tollis. *"Graph drawing: algorithms for the visualization of graphs"*. Prentice Hall Englewood Cliffs (N.J.), 1999. ISBN/ISSN : 0-13-301615-3.
3. F. J. Braedenburg, M. Himsolt, and C. Rohrer. "An Experimental Comparison of Force-Direceted and Randomized Graph Drawing Algorithms". In *Procs of Graph Drawing, GD'95*, pages 76–87, 1996.
4. J. Branke, F. Bucher, and H. Schmeck. "Using Genetic Algorithms for Drawing Undirected Graphs". In *Procs of the 3rd Nordic Workshop on Genetic Algorithms and Their Applications, 3NWGA*, pages 193–2005, 1997.
5. Dan Brickley and R. V. Guha. "Resource Description Framework (RDF) Schema specification: Proposed Recommendation, W3C", March 1999. http://www.w3.org/TR/1999/PR-rdf-schema-19990303.
6. Sergey Brin and Lawrence Page. "The Anatomy of a Large-scale Hypertextual Web Search Engine". In *Proceedings of the 7th International WWW Conference*, Brisbane, Australia, April 1998.
7. L. J. Campbell, Terry A. Halpin, and Henderik Alex Proper. "Conceptual Schemas with Abstractions: Making Flat Conceptual Schemas More Comprehensible". *Data and Knowledge Engineering*, 20(1):39–85, 1996.
8. Rodolfo Castello, Rym Mili, and I. Tollis. "A Framework for the Static and Interactive Visualization of Statecharts". *Journal of Graph Algorithms and Applications*, 6(3):313–351, 2002.
9. P. Chen. "The Entity-Relationship Model - Toward a Unified View of Data". *ACM Transactions on Database Systems*, 1(1):9–36, March 1976.
10. Richard Cole. "Automatic Layout of Concept Lattices using Force Directed Placement and Genetic Algorithms". In *Proc. of the 23th Australiasian Computer Science Conference*, pages 47–53. Australian Computer Science Communications 1, IEEE Computer Society, 2000.
11. R. Davidson and D. Harel. "Drawing Graphics Nicely Using Simulated Annealing". *ACM Trans. Graph.*, 15, 1996.
12. M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L.A. Stein. "OWL Web Ontology Language 1.0 Reference", 2002. (http://www.w3c.org/TR/owl-ref).
13. P. Eades. "A Heuristic for Graph Drawing". *Congressus Numerantium*, 42, 1984.
14. Holger Eichelberger and Jurgen Wolff von Gudenberg. "UML Class Diagrams - State of the Art in Layout Techniques". In *Proceeding of Vissoft 2003, International Workshop on Visualizing Software for Understanding and Analysis*, pages 30–34, 2003.
15. P. Feldman and D. Miller. "Entity Model Clustering: Structuring a Data Model by Abstraction". *The Computer Journal*, 29(4):348–360, 1986.
16. T. Fruchterman and E. Reingold. "Graph Drawing by Force-directed Placement". *Software - Practice and Experience*, 21(11):1129–1164, 1991.
17. F.U.N.D.P. "DB-MAIN". (http://www.info.fundp.ac.be/∼dbm/).
18. Munish Gandhi, EdwardL Robertson, and Dirk Van Gucht. "Levelled Entity Relationship Model". In *Procs of the 13rd Intern. Conf. on the Entity Relationship Approach, ER'94*, pages 420–436, Manchester, U.K., December 1994.
19. Floris Geerts, Heikki Mannila, and Evimaria Terzi. "Relational Link-based ranking". In *Procs of the 30th Intern. Conference on Verly Large Data Bases, VLDB'2004*, Toronto, Canada, August 2004.
20. Zoltan Gyongyi, Hector Garcia-Molina, and Jan Pedersen. "Combating Web Spam with TrustRank". In *Procs of the 30th Intern. Conference on Verly Large Data Bases, VLDB'2004*, Toronto, Canada, August 2004.

21. Jean-Luc Hainaut. "Transformation-based Database Engineering". In *Transformation of Knowledge, Information and Data: Theory and Applications.* IDEA Group Pub., 2004.
22. Jouni Huotari, Kalle Lyytinen, and Marketta Niemela. "Improving Graphical Information System Model Use with Elision and Connecting Lines". *ACM Transactions on Computer-Human Interaction*, 10(4), 2003.
23. Yannis E. Ioannidis, Miron Livny, Jian Bao, and Eben M. Haber. "User-Oriented Visual Layout at Multiple Granularities". In *Proc. of the 3rd International Workshop on Advanced Visual Interfaces*, pages 184–193, Gubbio, Italy, May 1996.
24. T. Kamada. *"On Visualization of Abstract Objects and Relations".* PhD thesis, Dept. of Information Science, Univ. of Tokyo, Dec 1988.
25. Jon Kleinberg. "Authoritative Sources in a Hyperlinked Environment". In *Proceedings of 9th ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, USA, 1998.
26. Simon Lok and Steven Feiner. "A Survey of Automated Layout Techniques for Information Presentations". In *Procs of the 1st. Int. Symp. on Smart Graphics*, Hawthorne, NY, 2001.
27. Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms.* Cambridge University Press, 1995.
28. Arthur Ouwerkerk and Heiner Stuckenschmidt. "Visualizing RDF Data for P2P Information Sharing". In *Procs of the workshop on Visualizing Information in Knowledge Engineering, VIKE'03*, Sanibel Island, FL, 2003.
29. Aaron J. Quigley. *"Large Scale Relational Information Visualization, Clustering, and Abstraction".* PhD thesis, University of Newcastle, Australia, August 2001. (http://www.it.usyd.edu.au/~aquigley/thesis/aquigley-thesis-mar-02.pdf).
30. O. Rauh and E. Stickel. "Entity Tree Clustering: A Method for Simplifying ER Design". In *Procs of the 11th Int. Conf. on Entity-Relationship Approach, ER'92.*
31. K. Sugiyama and K. Misue. "A Simple and Unified Method for Drawing Graphs: Magnetic-Spring Algorithm". In *Procs of Graph Drawing Conference, GD'94*, pages 364–375, 1994.
32. K. Sugiyama and K. Misue. "Graph Drawing by Magnetic-Spring Model". *Journal on Visual Lang. Comput.*, 6(3), 1995.
33. R. Tamassia, C. Batini, and M. Talamo. "An algorithm for automatic layout of entity-relationship diagrams". In *Procs of the 3rd International Conference on Entity-relationship approach to software engineering*, pages 421–439. Elsevier North-Holland, Inc., 1983.
34. T. J. Teory, W. Guangping, D. L. Bolton, and J. A. Koenig. "ER Model Clustering as an Aid for User Communication and Documentation in Database Design". *Communications of the ACM*, 32(8):975–987, 1989.
35. Yannis Tzitzikas and Jean-Luc Hainaut. "Ranking the Elements of Conceptual Diagrams", 2005. (submitted for publication).

A Linear Algebra Version of (B)EntityRank

Let A be the generalized adjacency matrix of an ER diagram where $A[e_i, e_j]$ equals the number of transitions from e_i to e_j. Now the probability transition matrix M is obtained by normalizing each row of A to sum to 1. EntityRank is based on a Markov chain on the entity types with transition matrix

$$q \cdot \mathsf{U} + (1 - q) \cdot \mathsf{M}$$

where U is the transition matrix of uniform transition probabilities i.e. $\mathsf{U}[e_i, e_j] = 1/N$ for all i, j. The vector of the EntityRank scores, denoted by Sc, is then defined to be the stationary distribution of this Markov chain. Equivalently, Sc is the principal right eigenvector of the transition matrix $(q \cdot \mathsf{U} + (1-q) \cdot \mathsf{M})^T$, since by definition the stationary distribution satisfies $(q \cdot \mathsf{U} + (1-q) \cdot \mathsf{M})^T \mathsf{Sc} = \mathsf{Sc}$. On the other hand, BEntityRank (the *biased* version of EntityRank) is based on the transition matrix:

$$q \cdot \mathsf{B} + (1-q) \cdot \mathsf{M}$$

where $\mathsf{B}[e_j, e_i] = \frac{|attrs(e_i)|}{|Attr|}$ where $Attr$ denotes the set of all attributes of all entity types (i.e. $Attr = \cup\{\ attrs(e) \mid e \in E\}$).

As another remark note that in an undirected (strongly connected and non-bipartite) graph $G = (V, R)$, the stationary probability of a node u is given by $P(u) = \frac{deg(u)}{2|R|}$ where $deg(u)$ is the degree of u [27]. This means that in an undirected graph (or multigraph) the stationary probabilities can be computed very efficiently and without the need of an iterative algorithm. In our case we cannot employ the above method due to the "teleporting" transitions which are indispensable in our case for ensuring that the transition graph is strongly connected (note that large ER diagrams are not always connected). Specifically, the "teleporting" transitions of BEntityRank are not symmetric and this cannot be captured by an undirected graph. For instance, consider the case of an ER diagram consisting of two entity types $e1$ and $e2$ and one relationship type between them, where $e1$ has one attribute and $e2$ has two attributes. According to a random walk on the undirected graph both entity types have probability 1/2. According to BEntityRank if $q = 0$ then $P(e_1) = P(e_2) = 1/2$, if $q = 1$ then $P(e_1) = 1/3$ and $P(e_2) = 2/3$, and if $q = 0.5$ then $P(e_1) = 0.44$ and $P(e_2) = 0.55$.

A Multilevel Dictionary for Model Management

Paolo Atzeni[1], Paolo Cappellari[1], and Philip A. Bernstein[2]

[1] Università Roma Tre, Italy
{atzeni, cappellari}@dia.uniroma3.it
[2] Microsoft Research, Redmond, WA, USA
philbe@microsoft.com

Abstract. We discuss the main features of a multilevel dictionary based on a metamodel approach. The application is an implementation of ModelGen, the model management operator that translates schemas from one model to another, for example from ER to relational or from XSD to object. The dictionary manages schemas and, at a metalevel, a description of the models of interest. It describes all models in terms of a limited set of metaconstructs. It describes all the schemas in a unifying model, called the supermodel, which generalizes all the others. The dictionary is composed of four parts, based on the combination of two features: schema level or model level, and model specific or model generic. We also show how such a dictionary can be the basis for a model independent approach to reporting, that provides a detailed textual and XML description of schemas.

1 Introduction

The need for handling design artifacts corresponding to different models arises in many different application settings. In the database world, we often have different systems that we need to use to handle our data, which use different (data) models, or we might need to exchange or integrate schemas expressed in different models. Small variations of models are often enough to create difficulties: for example, while most designers use the ER model, the actual features adopted by different methodologies and tools almost never coincide and so any integration requires a conversion. The introduction of new technology often introduces more heterogeneity and more need for translations. This happened with respect to analysis and design settings, where the growth of UML has not really simplified things, as the formalism is very complex and most people use just a subset of it—but each uses a different subset! XML has also contributed to increased heterogeneity, as data sources are now often described by means of XML Schemas (possibly in simplified versions).

In all these situations, there is the need to translate design artifacts from one model to another. There is also a need to translate data, but we focus only on design artifacts here. The models of interest can be significantly different, including those traditionally used in databases, as well as many others, such as those for XML documents, Web site structure descriptions, data warehouses, and restrictions or variants of each of them. The requirement, in all these cases, is the capability to handle different models and to be able to translate schemas from one model to another: given a source scheme S_1 in a model M_1 and a target model M_2, the need is to be able to generate the translation of S_1 into M_2.

These translation problems have always been tackled in practical settings by means of ad-hoc solutions, for example by writing a program for each specific application. This is clearly very expensive, as it is laborious and hard to maintain. Bernstein et al. [8,9] have recently argued for generic solutions for all problems that require the management of descriptions of application artifacts. They proposed a high level approach, called *model management*, based on a set of operators to be applied to schemas. A specific operator in the family is *ModelGen*, which translates schemas from a source model to a target model, exactly as we required above. This paper reports on some new features of a recent development for ModelGen.

An early approach to ModelGen was proposed by Atzeni and Torlone [4,5] who developed a tool, called MDM, to manage heterogeneous schemas based on a notion of *metamodel*. A metamodel is a set of constructs (the *metaconstructs*) that can be used to define models. The translation of a schema from one model to another is then defined in terms of translations over the metaconstructs, in such a way that the same translation is used for a given metaconstruct in all the models where it appears. In this approach, a translation is performed by eliminating constructs not allowed in the target model, and possibly introducing new constructs. Translations are built from elementary transformations, each of which is essentially an elimination step. Other authors have proposed similar approaches, including Claypool and Rundensteiner et al. [11,12], Song et al. [19], Bézivin et al [10]. The ideas at the basis of the MDM tool and of similar approaches are interesting and useful. But they do have one weakness, namely that they hide the representation of the models and transformations within the source code of the tool. So any extension of the models or customization of the translations would be very complex.

We have recently started a completely new development for ModelGen, with various novel features. One important feature is that it makes the description of models and the specification of translations visible and easily modifiable. In this paper, we give a detailed account of the dictionary that enables this feature, with its structure and the consequent benefits. A preliminary discussion of the overall development is available in [2].

The main contribution of this paper is the structure of the dictionary, which allows the integrated management of the descriptions of models and schemas for the various models of interest. The dictionary has a relational structure, which allows for the effective development of translation steps by means of Datalog rules [2]. It also makes it easy to produce model-specific reports in a model-independent way.

The paper is organized as follows. Section 2 illustrates the background of the approach, its main features and the relationship with some related literature. The next two sections describe the dictionary in some detail: Section 3 concentrates on the lower level, which describes schemas and Section 4 concentrates on the metalevel, which describes models and thus the structure of the lower level. Then, in Section 5 we show how the approach can be the basis for effective, model-independent reporting. Section 6 is the conclusion.

2 Background, Contribution and Related Work

The starting point for our work is the MDM proposal [4], whose principles are as follows. A *metamodel* is a set of constructs that can be used to define models, which are instances of the metamodel. The approach is based on Hull and King's observation [14] that the constructs used in most known models can be expressed by a limited set of generic (i.e. model-independent) *metaconstructs*: lexical, abstract, aggregation, generalization, and function. Each model is defined by its constructs and the metaconstructs they refer to. Simple versions of popular models are as follows:

– a simplified version of the ER model involves entities (which correspond to the *abstract* metaconstruct), attributes for them (corresponding to the metaconstruct *attribute of abstract*), and relationships (*aggregations of abstracts*);
– a simplified version of the object-oriented (OO) model involves classes (which also correspond to abstracts), fields (also attributes of abstracts), and references from classes to classes (the metaconstruct *reference to abstract*).

The translation of a schema from one model to another is defined in terms of translations over the metaconstructs. A major concept in the approach is the *supermodel*, a model that has constructs corresponding to all the metaconstructs known to the system. Each model is a specialization of the supermodel. So a schema in any model is also a schema in the supermodel, apart from the specific names used for constructs. The supermodel acts as a "pivot" model, so that it is sufficient to have translations from each model to and from the supermodel, rather than translations for each pair of models. Thus, only $2n$ translations are needed between n models, not n^2 translations. Moreover, since every schema in any model is an instance of the supermodel, the only needed translations are those within the supermodel with the target models in mind; a translation is performed by eliminating constructs not allowed in the target model, and possibly introducing new constructs.

In our new ModelGen development effort, we wanted to automate as many activities as possible, and to support the rapid construction and maintenance of the others. A visible dictionary turned out to be a major contribution in this direction. We realized that a database structure for it would be effective, especially a relational one, as we felt that the translation steps could be effectively implemented in Datalog. To handle models and schemas effectively and to coordinate the individual models with the supermodel, we organized the dictionary in four parts, which can be characterized along two coordinates: the first corresponding to whether they describe models or schemas and the second depending on whether they refer to specific models or to the supermodel (see Figure 1).

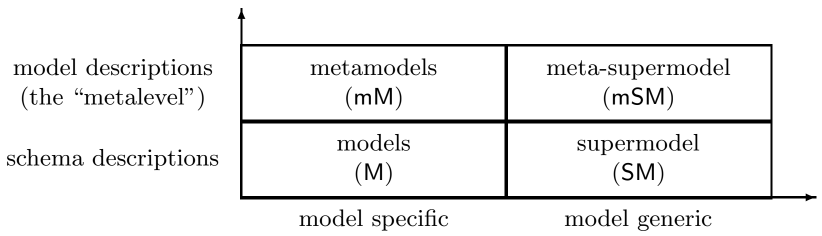


Fig. 1. The four parts of the dictionary

The most abstract part is at the model level and is model generic. It describes the supermodel, that is, the set of constructs used by the tool for building schemas. We refer to this part as mSM, the "meta-supermodel," as it is about the supermodel. The second part at the model level is model specific. It describes the individual models. It includes metamodels, referred to as mM. For each of them, mM describes the specific constructs used, each corresponding to a construct in the supermodel. We refer to these first two parts as the "metalevel" of the dictionary, as it contains the description of the structure of the lower level, whose content describe schemas. The lower level is also composed of two parts, one referring to the supermodel constructs (therefore called the SM part) and the other to model-specific constructs (the M part). The structure of the schema level is, in our system, automatically generated out of the content of the metalevel: so we can say that the dictionary is self-generating out of a small core.

In the next two sections we discuss the details of the two levels of the dictionary, proceeding bottom up. In Section 3 we concentrate on the schema level. In Section 4 we illustrate the metalevel and its relationship to the lower one.

There are many proposals for dictionary structure in the literature. The use of dictionaries to handle metadata has been popular since the early database systems of the 1970's, initially in systems that were external to those handling the database (see Allen et al. [1] for an early survey). With the advent of relational systems in the 1980's, it became possible to have dictionaries be part of the database itself, within the same model. Today, all DBMSs have such a component. Extensive discussion was also carried out in even more general frameworks, with proposals for various kinds of dictionaries, describing various features of systems (see for example [6,13,15]) within the context of industrial CASE tools and research proposals. More recently, a number of metadata repositories have been developed [17]. They generally use relational databases for handling the information of interest. There are other significant recent efforts towards the description of multiple models, including the Model Driven Architecture (MDA) and, within it, the Common Warehouse Metamodel (CWM) [18], and Microsoft Repository [7]; in contrast to our approach, these do not distinguish metalevels, as the various models of interest are all specializations of a most general one, UML based.

The description of models in terms of the (meta-)constructs of a metamodel was proposed by Atzeni and Torlone [4]. But it used a sophisticated graph language, which was hard to implement. The other papers that followed the same or similar approaches [10,11,12,19] also used specific structures.

We know of no literature that describes a dictionary that exposes schemas and instances in a highly correlated way, in both model-specific and model-independent ways. Only portions of similar dictionaries have been proposed. None of them offer the rich interrelated structure we have here.

3 The Schema Level

Let us proceed with the description of the dictionary starting from the most natural parts, those that describe schemas. The M component has the structure of traditional dictionaries: a table for each construct, with tuples corresponding to the elements in the schemas. In Figure 3 we show the dictionary for a version of the ER model, containing the descriptions of the two schemas in Figure 2.

The structure of the dictionary is rather standard and follows that often used in textbooks for describing the model (for example, Atzeni et al. [3, p.178]). We consider a simple version of the model, with n-ary relationships: this requires the separate table ComponentOfRelationship, used to specify the participation of an entity to a relationship. The version of the ER model we use includes a few of the basic features, which we use as representative properties. For example, minimum cardinalities are represented by the boolean IsOptional (abbreviated as IsOpt in the figure), so that the allowed minimum cardinality is 0 ("true" as the value for IsOpt) or 1 ("false" for IsOpt) and the maximum cardinality is represented by IsFunctional (IsFunct), with 1 and "N" as possible values. Similarly, for attributes, we have the boolean property isKey, used to represent whether they belong to the (main) identifier. Many other features are omitted here for the sake of space, such as optionality or nullability of attributes and external identification of entities. But they can be easily included and have indeed been implemented in our prototype tool. A Schema column in each table specifies the schema the constructs refer to. We could have Schema only in the Entity table and omit it from the others. But the redundancy is not a big issue and would have consequences requiring further discussion.

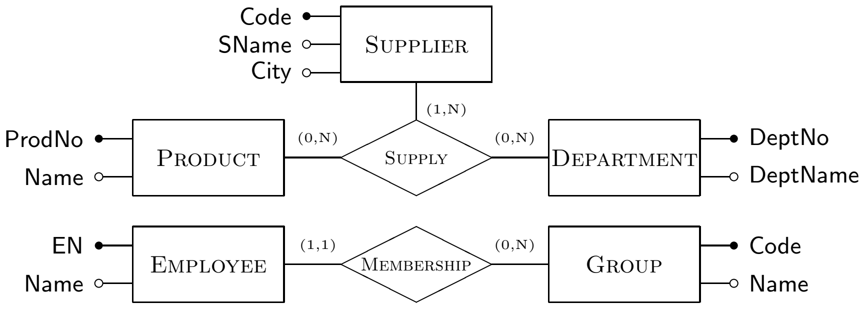


Fig. 2. Two simple ER schemas

Schema	
OID	Name
s1	1st ER Schema
s2	2nd ER Schema

Entity		
OID	Name	Schema
e1	Supplier	s1
e2	Product	s1
e3	Department	s1
e4	Employee	s2
e5	Group	s2

Relationship		
OID	Name	Schema
r1	Supply	s1
r2	Membership	s2

AttributeOfEntity					
OID	Entity	Name	Type	isKey	Sch.
a1	e1	Code	int	true	s1
a2	e1	SName	string	false	s1
a3	e1	City	string	false	s1
a4	e2	ProdNo	int	true	s1
a5	e2	Name	string	false	s1
a6	e3	DeptNo	int	true	s1
a7	e3	DeptName	string	false	s1
a8	e4	EN	int	true	s2
...	...	...	...	...	...

ComponentOfRelationship					
OID	Rel'ship	Entity	IsOpt	IsFunct	Sch.
c1	r1	e1	false	false	s1
c2	r1	e2	true	false	s1
c3	r1	e3	true	false	s1
c4	r2	e4	false	true	s2
c5	r2	e5	true	false	s2

Fig. 3. The dictionary for a simple ER model

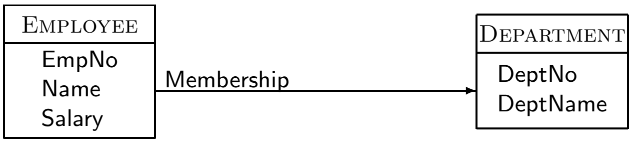


Fig. 4. A simple OODB schema

Schema	
OID	Name
s3	OODB Schema

Class		
OID	Name	Schema
cl1	Employee	s3
cl2	Department	s3

Field				
OID	Class	Name	Type	Sch.
f1	cl1	EmpNo	int	s3
f2	cl1	Name	string	s3
f3	cl1	Salary	int	s3
f4	cl2	DeptNo	int	s3
f5	cl2	DeptName	string	s3

ReferenceAttribute				
OID	Name	Class	ClassTo	Schema
ref1	Membership	cl1	cl2	s3

Fig. 5. The dictionary for a simple OODB model

In this M portion of the dictionary, we would like to be able to handle different models. Therefore, we have a portion of the dictionary like the one shown in Figure 3 for each model we are interested in. For example, if we want to handle a simple version of OODB schemas, such as that in Figure 4, we need a dictionary

Schema		
OID	Name	Model
s1	1st ER Schema	m1
s2	2nd ER Schema	m1
s3	OODB Schema	m2

Abstract		
OID	Name	Schema
e1	Supplier	s1
e2	Product	s1
e3	Department	s1
e4	Employee	s2
e5	Group	s2
cl1	Employee	s3
cl2	Department	s3

AttributeOfAbstract					
OID	Abstract	Name	Type	IsId	Sch.
a1	e1	Code	int	true	s1
a2	e1	SName	string	false	s1
...	...	...	...	...	...
a8	e3	EN	int	true	s2
...	...	...	...	...	...
f1	cl1	EmpNo	int	?	s3
f2	cl1	Name	string	?	s3
...	...	...	...	...	...

AggrOfAbstract		
OID	Name	Schema
r1	Supply	s1
r2	Membership	s2

ComponentOfAggrOfAbstract					
OID	AggrOfAbs	Abs	isOpt	isFunct	Sch.
c1	r1	e1	false	false	s1
c2	r1	e2	true	false	s1
c3	r1	e3	true	false	s1
c4	r2	e4	false	true	s2
c5	r2	e5	true	false	s2

ReferenceAttributeOfAbs				
OID	Name	Abs	AbsTo	Schema
ref1	Membership	cl1	cl2	s3

Fig. 6. A portion of the SM part of the dictionary

like the one shown in Figure 5. The OO model we use here is very simple, with just classes with scalar fields (each with a type, but with no other property; for example, we assume there is no way to specify they are optional nor to define keys) and reference attributes (fields that are references to other classes, representing 1:N relationships). The Schema table is the same as the one in Figure 3: the dictionary includes one table which refers to schemas for all the models of interest. It has an additional column that specifies the model to which the schema belongs, a reference to a Model table we will see shortly in the mM part of the dictionary.

Model-specific dictionaries are pretty standard. If we represented just the relational model, we would have something similar to some of the system tables of many DBMSs. The other parts of the dictionary are original, together with the overall structure. If we refer to Figure 1 again, starting from the M quadrant, two directions of abstraction are possible: we could consider the various models altogether, rather than each of them independently, thus moving to the SM quadrant in Figure 1. Or we can go from the level of schemas to its metalevel, that of models—quadrant mM. We consider the horizontal extension here, and the other in the next section.

As we mentioned in Section 2, a crucial issue in our approach is the use of the *supermodel*, a model that includes all the others as special cases, and can therefore be used to describe schemas in all models. The supermodel includes a rather limited set of constructs, to which the various model-specific constructs correspond. In the cases we just showed, Entity in the ER model and Class in OODB correspond to the same metaconstruct, called Abstract. The supermodel unifies in its tables all the dictionaries for the various models, by merging the tables whose constructs refer to the same metaconstruct. Entity and Class are merged, as are Attribute and Field. The schemas we saw in Figures 2 and 4 are represented in the SM portion of the dictionary by the tables shown in Figure 6. Here we see six tables rather than the nine in Figures 3 and 5. It is important to note that the number stays limited. In our current implementation we have ten tables for the SM part and more than fifty in the M part. The former number is stable whereas the latter could grow if new models are added.

The main benefit of this dictionary organization is that we can specify our translations in a simpler way, by referring to metaconstructs rather than to constructs. This is especially effective given the relational structure of the dictionary, as we write our rules in a variant of Datalog [2]. Notice that some properties in the supermodel are not used in some of the models. Also, properties need not have the same name in different models. For example, the IsId property of AttributeOfAbstract is not used in the Field table of the OO model and is used, but with the name IsKey, in the ER model. In fact, we have a null value, denoted by a question mark '?', for column IsId in tuples that correspond to fields of our OO model. We revisit this point in the next section.

4 The Metalevel

The second direction of abstraction of the basic dictionary is its metalevel, that is, the description of the structure of the dictionary itself. Essentially, this level stands with respect to the dictionary in the same way as the dictionary stands with respect to the actual data. As we have two components at the schema level (M and SM), we also have two components at the metalevel, the mM part, which describes the structure of M, and the mSM part, which describes the structure of SM. These components of the dictionary are shown in Figures 7 and 8, respectively. The structure that is shown is complete. The content refers to the specific, simplified dictionaries of the previous section, shown in Figures 6 (for the supermodel) and 3 and 5 (for the models).

In Figure 8, each row in the MSM_Construct table describes a metaconstruct, by means of a unique identifier (the OID column), a Name for the metaconstruct (which is also unique, but not really used as an identifier), and a boolean property isLexical, used to indicate whether the constructs corresponding to the metaconstruct have (visible) values or not. Hull and King [14] define as *lexical* the constructs that have values. In Figure 8, the rows of MSM_Construct list some of the main constructs we have defined, specifically, those needed for the constructs shown in the previous examples. The metamodel we are experimenting with contains a few additional metaconstructs, referring to aggregations of lexicals and their components, foreign keys over them (for handling value based

MM_Model	
OID	Name
m1	ER
m2	OODB

MM_Construct				
OID	Name	Model	MSM-Constr	IsLex
co1	Entity	m1	mc1	false
co2	AttributeOfEntity	m1	mc2	true
co3	Relationship	m1	mc3	false
co4	ComponentOfRelationship	m1	mc4	false
co5	Class	m2	mc1	false
co6	Field	m2	mc2	true
co7	ReferenceAttribute	m2	mc5	false

MM_Property				
OID	Name	Constr	Type	MSM-Pr
pr1	Name	co1	string	mp1
pr2	Name	co2	string	mp2
pr3	IsKey	co2	bool	mp3
pr4	Name	co3	string	mp4
pr5	IsOpt	co4	bool	mp5
pr6	IsFunct	co4	bool	mp6
pr7	Name	co5	string	mp1
pr8	Name	co6	string	mp2
pr9	Name	co7	string	mp7

MM_Reference					
OID	Name	Constr	IsPartOf	ConstrTo	MSM-Ref
ref1	Entity	co2	true	co1	mr1
ref2	Rel'p	co4	true	co3	mr2
ref3	Entity	co4	false	co1	mr3
ref4	Class	co6	true	co5	mr1
ref5	Class	co7	true	co5	mr4
ref6	Cl.To	co7	false	co5	mr5

Fig. 7. The mM part of the dictionary

MSM_Construct		
OID	Name	IsLex
mc1	Abstract	false
mc2	AttributeOfAbstract	true
mc3	AggregationOfAbstract	false
mc4	ComponentOfAggrOfAbstract	false
mc5	ReferenceAttributeOfAbs	false

MSM_Precedence	
Predecessor	Successor
mc1	mc2
mc1	mc3
mc3	mc4
mc1	mc5

MSM_Property			
OID	Name	Constr	Type
mp1	Name	mc1	string
mp2	Name	mc2	string
mp3	IsId	mc2	bool
mp4	Name	mc3	string
mp5	IsOpt	mc4	bool
mp6	IsFunct	mc4	bool
mp7	Name	mc5	string

MSM_Reference				
OID	Name	Constr	IsPartOf	ConstrTo
mr1	Abstract	mc2	true	mc1
mr2	Aggregation	mc4	true	mc3
mr3	Abstract	mc4	false	mc1
mr4	Abstract	mc5	true	mc1
mr5	AbstractTo	mc5	false	mc1

Fig. 8. The mSM part of the dictionary

models, especially the relational model), generalizations and their components, and a few others for handling nested structures. Note that "Abstract" is not lexical, as instances of the corresponding constructs (for example "Entity"), have no actual values directly associated with them. By contrast, "AttributeOfAbstract" is lexical, because the instances of its constructs, such as "AttributeOfEntity", do have values.

The rows in MM_Construct (in Figure 7) correspond closely to those in MSM_Construct. They describe constructs in the various models, each with a unique identifier (the OID column), a reference to the model (Model, a foreign key referencing the MM_Model table, which lists all the models currently stored in the system), a reference to the corresponding metaconstruct (MSM-Constr, a foreign key to MSM_Construct) and a Name for the construct, unique within the model. The first row in MM_Construct states that there is a construct with name "Entity," belonging to the "ER" model (as "m1" is the OID for such a model in Model), corresponding to the "Abstract" metaconstruct (as "mc1" is the OID for "Abstract" in MSM_Construct).

We can see here how mSM and mM belong to a metalevel: each tuple in MSM_Construct corresponds to a table in SM and vice versa and the same is the case for MM_Construct and the tables in M.

The other two tables in mSM, MSM_Property and MSM_Reference, describe some details of metaconstructs (and, as a consequence, of constructs), which in turn correspond to the structure of the tables in SM. The properties describe the value columns in the tables of the dictionary: each property has a Name and a Type, plus a unique OID and a reference to the construct it belongs to, Construct. For example, the second row in MSM_Property says that each "Attribute of Abstract" ("mc2" is the OID for "Attribute of Abstract") has property "Name," of type string. The third row says that each "Attribute of Abstract" also has a boolean one, "IsId." Thus, table AttributeOfAbstract in SM has a string column Name and a boolean column IsId, corresponding to these properties.

References in table MSM_Reference describe the columns in the dictionary tables that contain identifiers of other constructs. Specifically, each reference has a name, a construct it belongs to (Construct), a constructs it refers to (ConstructTo) and a boolean IsPartOf, which we will discuss soon. The second and third row in MSM_Reference say that each "Component of Aggregation of Abstracts" ("mc4" for Construct) has a reference named "Aggregation" to an "Aggregation of Abstract" ("mc3" for ConstructTo in the second row) and a reference named "Abstract" to an "Abstract" ("mc1" for ConstructTo in the third row). Again, this describes features of the schema level dictionary. The table ComponentOfAggregationOfAbstract has two references, one to AggregationOfAbstract and the other to Abstract.

The boolean IsPartOf specifies whether the construct which is the source of the reference is a component of the target and has no autonomous existence. This notion is used as a criterion for the automatic construction of reports for schemas in the various models. For example, the first and the second tuple have value "true" for IsPartOf, to specify that each "Attribute Of Abstract" is part of an "Abstract" and that each "Component of Aggregation of Abstract" is part of an "Aggregation of Abstract." The third row has "false," to specify that each

"Component of Aggregation of Abstracts" has a reference to an "Abstract," without being part of it. Clearly, if a construct has multiple references, at most one of them can have "true" for IsPartOf. Otherwise, the occurrences of this construct would be required to be "physical" components of two different objects at the same time.

The table MSM_Precedence specifies that a Successor construct can appear in a model only if a Predecessor one also appears. For example, a model can have "Aggregation of Abstract" only if it also has "Abstract." Even if most precedences follow from references, this need not be the case. In the cases shown in the figure, we have four precedences. The one we just commented on could not be inferred from references. In the mathematical sense, MSM_Precedence is a partial order, so a construct cannot be an indirect predecessor of itself. This is used by our model definition tool to allow for the definition of constructs in the proper order: we can introduce a concept only after its predecessor(s) have been defined.

In summary, the structure of the SM dictionary is completely described by the content of the mSM dictionary and can therefore be generated automatically out of it. Apart from the Schema table, SM contains a table for each row of MSM_Construct. Let c be the metaconstruct in one such row. The table for c has the following columns:

– the "service" columns OID and Schema
– one column for each row of MSM_Property that has c as the value for Construct
– one column for each row of MSM_Reference that has c as the value for Construct; this column is a foreign key reference to the table for the ConstructTo construct in the same row
– a Type column if the row of MSM_Construct has the value "true" for IsLexical

In the tool we are developing [2], the structure of SM is automatically generated once mSM is defined. Suitable features have been defined for changing it when needed, without losing the contents.

Tables MM_Property and MM_Reference in mM play the same role as MSM_Property and MSM_Reference play in mSM, in the sense that they describe the properties and references of the constructs in the various models. Each table of the M dictionary can be generated from mM in the same way as the tables of SM can be generated from mSM: one table for each row in MM_Construct, with "service" columns plus those indicated in MM_Property and MM_Reference.

Notice the relationships between the tables in mM (Figure 7) and mSM (Figure 8). A conceptual schema for the mM and mSM components of the dictionary is shown in Figure 9. A row in MM_Construct (Figure 7) states that there is a construct, with a given Name, in a certain Model (foreign key to MM_Model), corresponding to a MSM-Construct (foreign key to MSM_Construct). A row in MM_Property states that a Construct has a property with a Name and a Type. Each property here is a specialization of a property in mSM. This can be

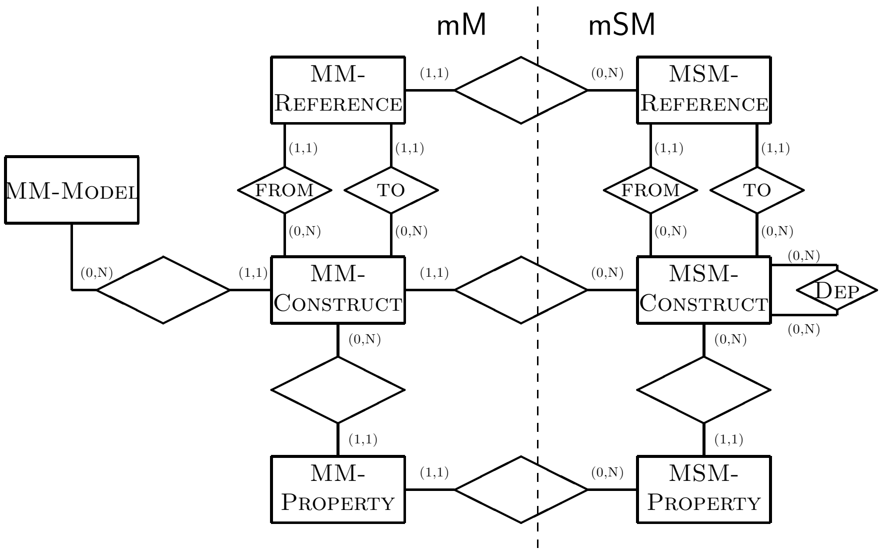


Fig. 9. A simplified ER-schema of the metalevel of the dictionary

expressed by the following constraint: for each row p in MM_Property, there is a row p' in MSM_Property such that p.Construct is a construct whose metaconstruct is p'.Construct (that is, there is a row c in MM_Construct such that c.OID=p.Construct and c.MSM-Construct=p'.Construct), and p.Type=p'.Type. A similar constraint holds between MM_Reference and MSM_Reference.

The various conditions on the tables in mM are enforced by the process used for defining models and populating the tables in mM. A model is defined by means of a set of constructs each of which is associated with a metaconstruct. At the same time, there cannot be two constructs in a model corresponding to the same metaconstruct. For each construct, one can define a property corresponding to each of the properties of the metaconstruct and a reference for each of the metaconstruct's references. However, in general, properties of metaconstructs need not all be used in the constructs that correspond to them. For example, "Attribute of Abstract" has a "IsKey" property, which is not used by "Field" in the OO model, whereas it is used in the ER model. This is the metalevel description of what we already commented on at the end of Section 3 regarding the schema level. Both properties and references can change their name from mSM to mM. For properties, the case we have in the examples seen so far is "IsId" which becomes "IsKey." In the examples all references change their names, as they correspond to the names of the constructs rather that those of the metaconstructs.

Notice that the structure and the content of both MM_Property and MM_Reference are redundant. Some of the information in them could be derived from that in MSM_Property and MSM_Reference, respectively. This is clearly the case for columns Construct and Type in MM_Property and Construct, IsPartOf, and ConstructTo in MM_Reference. However, in this way they can be used directly to generate the structure of the tables for the M dictionary, which is indeed their main use.

Also, it is worth noting that most of the tables in mM and mSM could be pairwise merged, as they are very similar: for example MSM_Construct and MM_Construct could be merged. This would lead to a "self-describing type system," so that code developed in the tool to operate on user-defined types would also operate on system-defined types. This would be true for example for display methods in an interactive tool and for reporting features. Here we preferred to show them separately, to emphasize the symmetry with the schema level. Another reason is that the process for defining a model reads values in mSM and inserts new ones into mM, and we wanted to emphasize the different roles. At the implementation level, the two tables could be merged, or even with separate tables, the code that accesses them could be shared.

5 Reporting

Whenever we have schemas in a model, we are interested in producing *reports* for them. Such reports give detailed textual documentation of the organization of schemas, in a readable and machine-processable way. This issue is delicate here, because we handle many different models, each with its own special features. Each model has its own specific aspects that it could be important to highlight. Probably, in order to provide a suitable emphasis for the specific aspects of each model, we would need a reporting facility expressly defined for it. Still, in a framework that allows for the definition of many different models, a general, flexible way of producing reports would be highly desirable.

Our metamodel approach gives the basis for a model independent reporting feature. Each model involves a set of constructs, each with properties and references. Properties are directly associated with constructs. For example, the Name of an Entity in the ER model or the Type of a Field in the OO model. References relate constructs, in at least two different ways. In some cases, the reference specifies that a given construct is so tightly associated with another construct that it is indeed a component. For example, in the ER model, each AttributeOfEntity is a component of an Entity and each ComponentOfRelationship is a component of Relationship. In other cases, the connection is looser, essentially an external reference to another construct. This is the case for the reference between ComponentOfRelationship and Entity. As we discussed in Section 3, we have the boolean IsPartOf exactly to distinguish these two types of references.

Using this idea, we have developed a simple, effective algorithm for producing reports which associates components with the constructs they belong to. It takes into account a topological order over constructs induced by MSM_Precedence and presents occurrences of constructs according to that order. A topological order over constructs always exists, as MSM_Precedence is a partial order (as we said in Section 3). For example, constructs that are not components of other constructs (i.e., *base* constructs) are presented first. Precedences are actually defined in mSM but can be extended in a straightforward way to each model in mM. Occurrences of constructs that are incomparable according to the topological order are clustered. In principle, the presentation order of such clusters is arbitrary. In the current implementation, they are presented in the order in which the constructs were defined in the model.

The reports are produced in XML, so that they are both self-documenting and machine processable if needed, for example, for better presentation by means of style sheets. Let c_1, ..., c_k be a topological ordering for the base constructs in the model. The algorithm considers the constructs in order, c_1, ..., c_k, and, for each c_i outputs the description of each of its occurrences o, with the form:

```
element named $c_i$ with attributes including OID and properties of $o$,
with the following optional subelements
    if $o$ has components, an element <components>, with subelements
      that are the (recursive) descriptions of the component occurrences
    if $o$ has references with "false" for IsPartOf
      the referenced object with its properties (but not the components)
```

The names of the constructs themselves come from the metamodel and therefore the generation is indeed model-independent.

The structure of the report for an ER schema, according to the simple version of the model we saw in Section 3, would be the following, with some syntactic sugar to make it more readable:

```
<schema name="schemaName" model="modelName">
  <constructs>
    <constructName listOfProperties>
      <components> (if any)
        <constructName listOfProperties>
          ...
        </constructName>
        ...
      </components>
    </<constructName>
    <constructName listOfProperties>
    ...
  </constructs>
</schema>
```

Therefore, the report for the first ER schema shown in Figure 2 would be the following:

```
<schema OID="s1" name="1st ER Schema" model="ER">
 <constructs>
  <entity OID="e1" name="Supplier">
   <components>
    <attributeOfEntity OID="a1" name="Code" isKey="true" type="int"/>
    <attributeOfEntity OID="a2" name="SName" isKey="false"
         type="string"/>
    <attributeOfEntity OID="a3" name="City" isKey="false" type="string"/>
   </components>
  </entity>
  <entity OID="e2" name="Product">
   <components>
    <attributeOfEntity OID="a4" name="ProdNo" isKey="true" type="int"/>
    <attributeOfEntity OID="a5" name="Name" isKey="false" type="string"/>
   </components>
  </entity>
  <entity OID="e3" name="Department">
   <components>
    <attributeOfEntity OID="a6" name="DeptNo" isKey="true" type="int"/>
    <attributeOfEntity OID="a7" name="DeptName" isKey="false"
          type="string"/>
   </components>
  </entity>
  <relationship OID="r1" name="Supply">
   <components>
    <componentOfRelationship OID="c1" isOpt="false" isFunct="false">
     <entity OID="e1" name="Supplier"/>
    </componentOfRelationship>
    <componentOfRelationship OID="c2" isOpt="true" isFunct="false">
     <entity OID="e2" name="Product"/>
    </componentOfRelationship>
    <componentOfRelationship OID="c3" isOpt="true" isFunct="false">
     <entity OID="e3" name="Department"/>
    </componentOfRelationship>
   </components>
  </relationship>
 </constructs>
</schema>
```

We experimented with stylesheets for reports generated in this way. They are reasonably effective, even when produced in a completely model-independent way. For example, a convenient way to produce reports is hypertext based, where a single HTML page is built for a schema, with internal links that make it easy to navigate from one construct to another. Lists of constructs of the various types can also be easily generated.

6 Conclusions

The structure of the dictionary we have shown here is being used in the tool under development. It supports a variety of activities in a model independent way. Beside the generation from the core, it supports reporting, as we illustrated, and generation of parametric formats for import and export of schemas. Overall, the availability of a dictionary with a visible structure is very useful in the development of translation rules and in the maintenance of the tool itself. In particular, changes to the model do not require changes to the tool's engine. These are consequences of the main novelty of our approach which offers an integrated and highly correlated representation of schemas and models, both in a model specific way and within a unified framework, the supermodel.

References

1. F. W. Allen, M. E. S. Loomis, and M. V. Mannino. The integrated dictionary/directory system. *ACM Comput. Surv.*, 14(2):245–286, 1982.
2. P. Atzeni, P. Cappellari, and P. A. Bernstein. Modelgen: Model independent schema translation. In *ICDE*, IEEE Computer Society, pages 1111-1112, 2005.

3. P. Atzeni, S. Ceri, S. Paraboschi, and R. Torlone. *Databases: concepts, languages and architectures.* McGraw-Hill, 1999.
4. P. Atzeni and R. Torlone. Management of multiple models in an extensible database design tool. *EDBT, Lecture Notes in Computer Science 1057*, Springer, pages 79–95, 1996.
5. P. Atzeni and R. Torlone. Mdm: a multiple-data-model tool for the management of heterogeneous database schemes. *SIGMOD*,ACM, pages 291–301, 1997.
6. C. Batini, G. D. Battista, and G. Santucci. Structuring primitives for a dictionary of entity relationship data schemas. *IEEE Trans. Software Eng.*, 19(4):344–365, 1993.
7. P. Bernstein, T. Bergstraesser, J. Carlson, S. Pal, P. Sanders, and D. Shutt. Microsoft repository version 2 and the open information model. *Information Systems*, 22(4):71–98, 1999.
8. P. A. Bernstein. Applying model management to classical meta data problems. *CIDR*, pages 209–220, 2003.
9. P. A. Bernstein, A. Y. Halevy, and R. Pottinger. A vision of management of complex models. *SIGMOD Record*, 29(4):55–63, 2000.
10. J. Bézivin, E. Breton, G. Dupé, and P. Valduriez. The atl transformation-based model management framework. Research Report Report 03.08, IRIN, Université de Nantes, 2003.
11. K. T. Claypool and E. A. Rundensteiner. Sangam: A framework for modeling heterogeneous database transformations. In *ICEIS (1)*, pages 219–224, 2003.
12. K. T. Claypool, E. A. Rundensteiner, X. Zhang, H. Su, H. A. Kuno, W.-C. Lee, and G. Mitchell. Sangam - a solution to support multiple data models, their mappings and maintenance. In *SIGMOD Conference*, 2001.
13. C. Hsu, M. Bouziane, L. Rattner, and L. Yee. Information resources management in heterogeneous, distributed environments: A metadatabase approach. *IEEE Trans. Software Eng.*, 17(6):604–625, 1991.
14. R. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, Sept. 1987.
15. B. K. Kahn and E. W. Lumsden. A user-oriented framework for data dictionary systems. *DATA BASE*, 15(1):28–36, 1983.
16. S. Melnik. *Generic Model Management: Concepts and Algorithms.* Springer-Verlag, 2004.
17. E. Rahm and H. Do. On metadata interoperability in data warehouses. Technical report, University of Leipzig, 2000.
18. R. Soley and the OMG Staff Strategy Group. Model driven architecture. White paper, draft 3.2, Object Management Group, November 2000.
19. G. Song, K. Zhang, and R. Wong. Model management though graph transformations. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 75–82, 2004.

A MOF-Compliant Approach to Software Quality Modeling

Xavier Burgués[1], Xavier Franch[1], and Josep M. Ribó[2]

[1] Universitat Politècnica de Catalunya (UPC) c/ Jordi Girona 1-3 (Campus Nord, C6) E-08034 Barcelona, Catalunya, Spain
{diafebus, franch}@lsi.upc.edu
[2] Universitat de Lleida (UdL), C. Jaume II, 69 E-25001 Lleida, Catalunya, Spain
josepma@eps.udl.es

Abstract. Software quality is a many-faceted concept that depends on the kind of artifact to be measured, the context where measurement takes place, the quality framework used, and others. Furthermore, there is a great deal of standards, white papers, and in general proposals of any kind related to software quality. Consequently, a unified software quality framework seems to be needed to compare, combine or select these proposals and to define new ones. In this paper we propose a MOF-compliant approach for structuring quality models in order to formalise software quality issues and deal with quality information modelling. We propose two types of models: a generic model, situated in the M2 MOF layer; and a hierarchy of reference models, defined in the M1 and M0 MOF layers. The generic model elements are derived from the UML metamodel by specialization. Then, we can instantiate them to get reference models that formalise (combinations of) existing proposals which may be further refined for defining quality frameworks to be used in different experiences. Each of these models is divided into three parts, namely fundamental concepts, metrics and context. We illustrate our proposal providing a multi-level reference model in the context of collection libraries quality evaluation.

1 Introduction

Quality assessment and management (QA&M) plays currently a crucial role in all the facets of software development. This means that not only the software process and the system-to-be are targets of QA&M, but also subprocesses such as specification, design and testing, and software-related artifacts such as system requirements, specifications and software architectures. As a result, we may find a great deal of proposals aiming at the study of QA&M issues in those contexts, so diverse in nature such as software process assessment and improvement [32], analysis of data models like UML class diagrams [18] or ER models [27], measurement of OO designs [6], and so on. Furthermore, the tendency seems not to converge into more compact, general-purpose frameworks but on the contrary, to provide new, specialized proposals.

All of these proposals share a core of common concepts, e.g. metrics, quality factor, etc., but it is not obvious to identify similarities and differences between them. This difficulty hampers the understanding of the quality frameworks, their further extension or evolution, and their comparison when it becomes necessary to choose one in a given context.

Several authors claim that ontologies, conceptual models or similar descriptions are needed in order to precisely define the concepts, processes, languages and tools related to software quality [24, 30]. The goal is the definition of a framework useful to analyse the variety of approaches, to define new ones and to adapt the existing ones to new contexts. As a result, it becomes necessary to work on the foundations, to obtain a set of widely accepted general concepts with a clear structure to be used as the basis of particular methods and tools.

In [5] we proposed a 3-level hierarchy of quality models. Each level was related to a different abstraction degree: the generic model provided a universal unified framework; reference models allowed the definition of operational frameworks, ready to be used as the concepts coming from the generic model were instantiated; domain models fit in the specificities of concrete QA&M experiences and set a comfortable way to deal with quality. This hierarchy was a good starting point for stating a quality framework but some serious problems were identified:

- Our proposal was *ad hoc*, without being integrated into any existing and consolidated metamodel, architecture or ontology. This was a serious drawback considering three different aspects: semantics of the models; reuse of existing concepts, methods and tools; and dissemination of the approach.
- The frontier among reference and domain models was too fuzzy and arbitrary. In fact, for some applications, we found that the last thing to refine was not the domain but other parts of the generic model, for instance the type of artifact itself or the metrics to be used.
- We just allowed one level of reference and domain models. This was in fact a serious limitation, since it was impossible to refine or combine models hampering thus quality knowledge structure and reuse.
- We included in the generic model a dimension (the language dimension) that is not present in all the approaches, unlike the other three dimensions that we consider.
- Our experiences were a few and then the proposal was still unstable. Once we acquired more knowledge we discovered some minor flaws, specially in the generic model.

In this paper we propose a conceptual framework for structuring quality models that overcomes these drawbacks. The framework is presented in section 2. It is integrated into the Meta Objects Facility (MOF) architecture [26] as an extension of the UML metamodel [36], it supports a hierarchical structure of reference models without imposing any particular refinement order, it removes the language dimension and it has been validated with a greater number of cases (i.e., proposals about quality available in the literature). The core of the proposal is presented in sections 4 and 5. Previously, section 3 provides a short summary about the MOF architecture and UML metamodel. Section 6 provides the conclusions.

2 A Hierarchy of Quality Models

We present a framework for dealing with software quality that consists of a hierarchy of two types of quality models:

- *Generic model*. The root of the hierarchy. It introduces the fundamental concepts that are present in every single approach to QA&M. "Quality model", "artifact" and "metrics" are some of these concepts. It is abstract enough to be used in several software engineering activities: specification, design, development, certification, selection, etc.
- *Reference models*. They provide particular interpretations of the generic model fundamental concepts in a particular setting. As an example, a reference model could be built up following the ideas of the ISO/IEC 9126 quality standard, part 1 [21] and incorporating metrics-related notions coming from the theory in [12, 37]. We may have different degrees of refinement which means that reference models can be structured in hierarchies until we obtain leaves, which stand for reference models that can be used in particular QA&M experiences. Also the hierarchy may contain models that are not obtained by refinement but by composition, because different models may focus on just one part of the generic model.

Although diverse, these two models are structured into three different parts:

- *Fundamental concepts*. It embraces the concepts and relationships that form the quality models and system requirements about quality. The concepts therein stem from general quality standards [20, 21] and widespread catalogues of quality factors and requirements [11, 25].
- *Metrics*. Here we define the types of metrics to be used to measure the items defined by the model and to state the satisfaction of requirements. Classic proposals [12, 37] and quality standards again [21] are the foundations of this part.
- *Context*. It has to do with the software domains which the quality models will be attached to; the structure of artifacts to be measured; and the environment in which they operate (a type of organisation, a particular one, a department, a project, etc.). Domains may be structured as a taxonomy as proposed in [19, 8]. The artifacts may be aggregations or compositions of others.

Figure 1 illustrates the evolution from the generic model to reference ones. It shows also the recommended use of our framework. From the generic model, we obtain virtually hundreds of reference models, one for each consolidated proposal that has been defined in the literature for the concepts present in the generic model (we depict just three in the figure). For instance, we get a reference model for the ISO/IEC 9126 standard, other for the software domains as defined in the INCOSE taxonomy, other for the Stevens set of scales [34], and so on. The important thing is that each of these reference models takes as few assumptions as possible, not compromising therefore its use unnecessarily; in fact, a great deal of these reference models just refine one of the three parts of the generic model sometimes even not completely.

These first-level reference models can then be refined to introduce details, as shown in fig. 1. This allows to structure quality proposals in such a way that details are introduced progressively, making understanding easier. A general strategy we have adopted is to use first-level reference models to represent the general structure of the approach, and second-level ones to indicate particular elements. Another strategy consists on using this refinement concept for distinguishing among normative or mandatory parts of a proposal (defined at the first-level) from optional or recommended parts (defined at lower levels). We present examples of both situations in section 5.

Once the target level of detail has been reached, we can combine the lower-level reference models to obtain new ones embracing all the aspects of quality. This combination can be made for several reasons: to put together widespread proposals for further use, as we will show in section 5; to construct *ad hoc* frameworks for particular experiences; or to create a collection of reusable frameworks that can be used in lots of quality-related experiences. Fig. 1 also illustrates this combination.

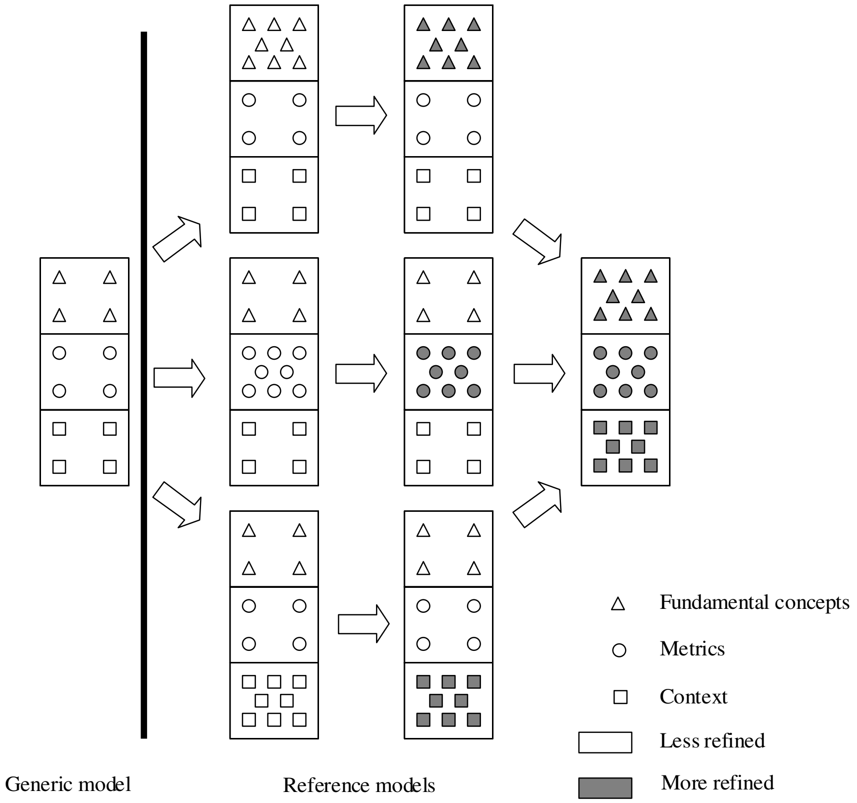


Fig. 1. A framework for constructing quality models

3 The MOF Architecture and the UML Metamodel

UML [36] is currently a *de facto* standard in object-oriented modeling. As it is a general purpose notation, it has to be tailored to specific contexts, in particular to different software domains. The extensions provide some clear advantages: the integration of the domain in a standard framework, a potential usage by the software engineering community and the existence of a large number of support tools. Moreover, integration to the widespread MOF metamodeling architecture [26] –adopted by OMG– is also provided.

Due to these advantages, many proposals to extend the UML metamodel in different domains have come out in the last few years. Consider, just to mention a few, SPEM [33], CWM [10], CORBA [9] and [35]. Our work is integrated in this UML extension approach and, hence, it benefits from the above mentioned advantages.

The MOF metamodeling architecture consists of four layers: M0, M1, M2 and M3, such that layer M_i contains instances of elements defined in M_{i+1}. M0 contains runtime objects, M1 is the model level and contains the classes for M0 objects; UML models are defined in M1. M2 is the metamodel level and contains elements used to build models in M1 (e.g., *Class*, *Association* and *Dependency*). The UML metamodel and other metamodels are defined in M2. Finally, M3 is the meta-metamodel level. MOF is the meta-metamodel shared by all the metamodels defined in the OMG framework. Its main contents are the essential elements of the UML metamodel (in particular, the infrastructure library of the UML 2.0 metamodel is reused in the MOF model definition [36]).

As mentioned in the introduction, in this paper we propose to extend the UML metamodel with concepts aimed at the generation of quality models of software-related artifacts. We want the result to be fully integrated in the MOF metamodeling architecture and coherent with the standard UML extension mechanism in order to end up with a UML-consistent result. There are two ways to tailor UML: (1) define a so-called *heavyweight extension*, which provides a first-class metamodel extension mechanism for extending the metamodel and (2) define a *UML profile*. While the first approach is more expressive and comprehensible, the second one provides compatibility with UML modeling tools. Therefore, in our work we have adopted both of them following the methodology defined in [13], which consists of two steps:

1. Proposal of a heavyweight extension of the UML metamodel, providing a properly built metamodel.
2. Transformation of the previous extension into a UML profile. Since the extension built in step 1 preserves the semantics of the original UML metamodel and since all its metaelements are based on already existing UML metaelements, this transformation can be done in a semi-automatic way using any of the methodologies presented in [23, 36, 13].

4 A Generic Model for Quality as an Extension of the UML Metamodel

As our generic model is an extension of the UML metamodel, its elements are metaclasses and metaassociations at layer M2. Our target is the construction of a quality metamodel which any reference model can be an instance of. Figure 2 shows the package layout of the integration of the generic model into the UML metamodel and figure 3 shows the model itself. Table 1 lists the UML metamodel elements from which the generic model elements derive by specialization.

The elements are presented next. We structure the presentation into the three parts enumerated in section 2. In fact, the three parts are also structured as packages, but since this structure is not essential for the goal of the paper, we do not address this issue.

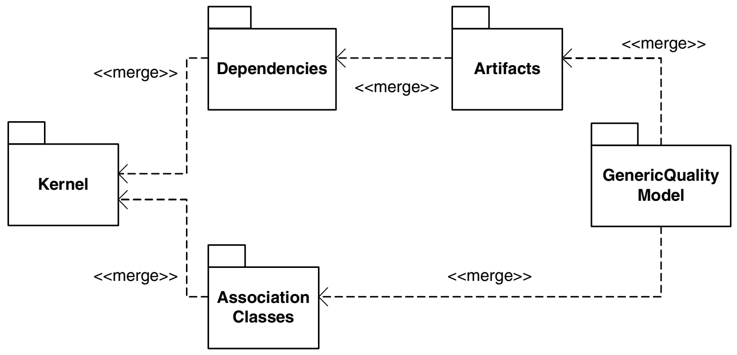


Fig. 2. Integration of the generic model into the UML metamodel

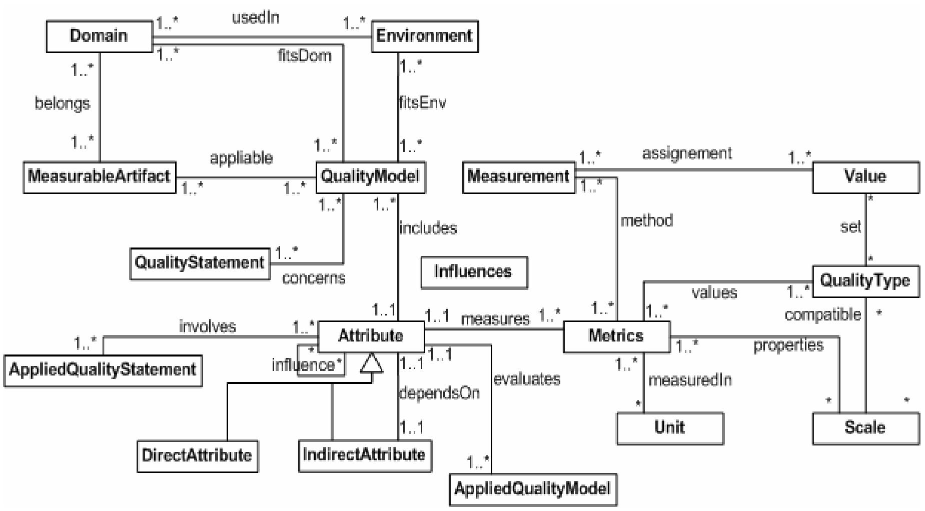


Fig. 3. The generic model

Table 1. Connection among the generic model and the UML metamodel

Generic model	UML metamodel
Domain, Environment, Attribute (and heirs), Metrics	Classifier
MeasurableArtifact	Artifact
Scale, Unit	Enumeration
QualityType	DataType
Value, QualityStatement, AppliedQualityStatement	ValueSpecification
QualityModel, AppliedQualityModel, AppliedQualityStatement, Measurement, Influences	AssociationClass

4.1 Context

Domain captures the knowledge about those aspects of *MeasurableArtifacts* for which a quality model is required. The concept of "artifact" is utterly comprehensive, embracing different things that can be as general as the notion of "office tool", as specific as a "sorting algorithm", or related not to the software itself but to software descriptions and documentations as models or specifications. It is important to remark that an artefact may *belong* to several domains. For instance, the quality of a geographical information system may be viewed from the point of view of information systems and from the point of view of geographical software.

The class *Environment* represents the circumstances in which the domains are *used*. It may define the kind of organization (for instance, a huge supermarket, a non-governmental organization, ...), a kind of project (like a CMM level 3 one), etc.

Taking into account that the three classes have been defined as UML classifiers, we can compose instances of these three elements and refine them, therefore we may construct taxonomies of domains and packages of artefacts, we can tailor quality concepts in each single department of an organization, etc. As a consequence, we may define in a reference model, for instance, that the reliability of a component is a combination of the reliability of its subcomponents.

4.2 Fundamental Concepts

A *QualityModel* will *fit* into a domain in a given environment (e.g., there will be a quality model for groupware systems in medium-size companies and a quality model for mail servers in ISPs). Thus, *QualityModel* classes of the reference models (instances of the *QualityModel* metaclass of the generic model) will turn into association classes between Domain and Environment classes. Please note that we are not arguing about the adequacy and validity of the *QualityModel*, but just focusing on the concepts which it stems from. As an artifact may belong to several domains and a domain may be used in several environments, several quality models may be *appliable* to each artifact. Quality models *include* quality *Attributes* (such as efficiency, reliability, etc.). As *Attribute* is an heir of *Classifier*, we may organize them as a hierarchy, which is the usual case in most proposals. Therefore, they can be *Direct* or *Indirect*; indirect attributes *depend on* others. Attributes are not mutually independent and so their *influence* on each other is also represented. *Influences* allows categorizing and grading these dependencies (providing kind, intensity, etc.). The class *QualityStatement* represents quality requirements, constraints, criteria and other elements that *concern* a quality model and that may be needed to use the model in an evaluation process.

A quality model is applied on an artefact to obtain an *AppliedQualityModel* in which its attributes will be *evaluated*. This concept is the cornerstone of a particular QA&M experience. It allows for instance to obtain a quality model for a particular mail server, ERP system, UML class diagram, or whatever. Quality statements over an applied quality model are called *AppliedQualityStatements* and are the refinement of the quality statements (e.g., "The page shall be downloaded in less than 5 secs.") of a particular quality model *involving* attributes from this model (e.g., *throughput* or *refreshing time*).

4.3 Metrics

The class *Metrics* represent possible *methods* to perform *measurements* of attributes to find the *value* to *assign*. Metrics are mainly characterised by the category of their *Scale* (which define their *properties*), their *Unit* of measurement and the *QualityType* of their *Values* (that of course shall be *compatible* with the scale).

Measurement appears as the *assignment* of values to the attributes of the artifacts under consideration by applying a particular *method* of measurement in the form of a metric that applies to the attribute. It is possible for a given attribute to be measured with different methods in different quality models. For this reason, it would have been more natural to model this measurement as a ternary relationship between quality models, artefacts and attributes, but since ternary associations are not supported by the MOF at level M2 [26], we have decomposed them into binaries.

5 An Example: A Reference Model for Collection Libraries Quality Framework

In this section we aim at building a reference model related to the evaluation of collection libraries like JCF [1], STL [28] and LEDA [29]. Such a model would be useful in some situations, remarkably when a library of this kind is necessary during the development of an application; this is a particular case of the general problem of Commercial Off-The-Shelf (COTS) component selection [14]. This reference model relies on the combination of some standards, catalogues and other proposals. It is important to remark that our goal in this section is to show how a realistic reference model, which integrates well-known proposals made in the field of software quality, can be defined as an instantiation of the generic model presented in the previous section. For this reason, we are not arguing that the resulting model is the best one for its intended purpose; this is beyond the scope of this paper and would deserve a thorough investigation in its own. Partly also for the same reason, and partly for lack of space, the resulting quality model is not fully complete, we have focused on the most relevant parts of its structure and not in the details, for instance how to express the experimental procedures for metrics and how to validate them.

According to the guidelines outlined in the introduction, we adopt a strategy such that reference models are fine-grained (i.e., each of them refines just those elements directly implied by the approach) making easier their reuse in many contexts. The general strategy is depicted at fig. 4. We decide to divide the experience into two parts. In the first part we build a reference model for the general problem of COTS selection, called *COTS selection framework* in the figure. This is built upon:

- The ISO/IEC 9126 standard, parts 1 (the quality model structure [21]) and 2 (proposal of external metrics for software artifacts [22]). This standard already sets the high level attributes (called characteristics and subcharacteristics in the standard) to be measured. Otherwise, they should be set as the result of an analysis process like GQM [2] or other similar goal-driven proposals.
- The NFR framework [7] for establishing different degrees of relationships among quality factors.

- Fenton's metrics framework [12] for presenting metrics fundamental concepts.
- A commercial classification of COTS components coming from the Gartner consulting group to define software domains of interest [15].

At the first level of the hierarchy, we provide separate refinements for each of the parts (fundamentals, metrics and domain) of the generic model focusing on the general layout of each proposal. The three resulting models stay at the M1 MOF layer, with their elements obtained as instantiations of the M2 generic model; in particular, each association in M2 induces an association or association class in M1.

In the second level we include concrete values for the concepts, i.e.: the quality factors provided by the ISO/IEC 9126-1 (efficiency, suitability, …); the NFR type of links (hurt, some, …); Fenton's scales (nominal, ordinal, …) with simple types; and the concrete elements of the Gartner hierarchy (CRM, ERP, DCM, …). The added part of these models are at M0, which means that their elements are instances of the former M1 reference models (which of course are preserved at this level).

In the third level we add more detail to the metrics part refining by means of the ISO/IEC 9126-3 (including types, units, etc.). We did not include this part in the former level again for enhancing reuse: we can propose other concrete metrics starting from the level-two metrics-related model. Also, at this third level, we merge the two level-2 reference models that refined the fundamental concepts part. Finally, at the forth level, we join the three reference models for fundamental concepts, metrics and context and we obtain the target COTS selection framework. This reference model may be used as starting point for building proposals for supporting COTS selection processes at any kind of software domain, such as ERP systems and collection libraries.

The reference model for collection libraries is based on two basic manipulations. On the one hand, the internal structure of the collection libraries shall be represented in the reference model. This is done at the first two levels, by declaring the artifacts as composable (M1) and then by defining a collection library as composed by collections and algorithms, the former composed of data structures and operations (M0). On the other hand, the COTS selection framework is specialized not only by including this new second-level model but also by modifying the instances in M0 for customizing to this domain. Once we have the model built, we evaluate particular collection libraries such as JCF, STL and LEDA.

Let's take next a closer look to the resulting models. We present directly the COTS selection framework at fig. 5 (M1) and 6 (M0). We may observe the following:

- The inheritance hierarchy classifying attributes as direct and indirect is combined in this particular model with the classification of quality factors in attributes[1], subcharacteristics and characteristics made by the standard ISO/IEC 9126. The refined model reflects ISO/IEC's directives concerning the hierarchy (see <<*dependsOn*>> associations): characteristics may be decomposed into subcharacteristics, subcharacteristics into other subcharacteristics and attributes and, finally, attributes may be decomposed into attributes.

[1] Please do not be get confused with the general term "attribute" that we have introduced in the generic model and the particular term "attribute" as defined in the ISO/IEC 9126-1 standard.

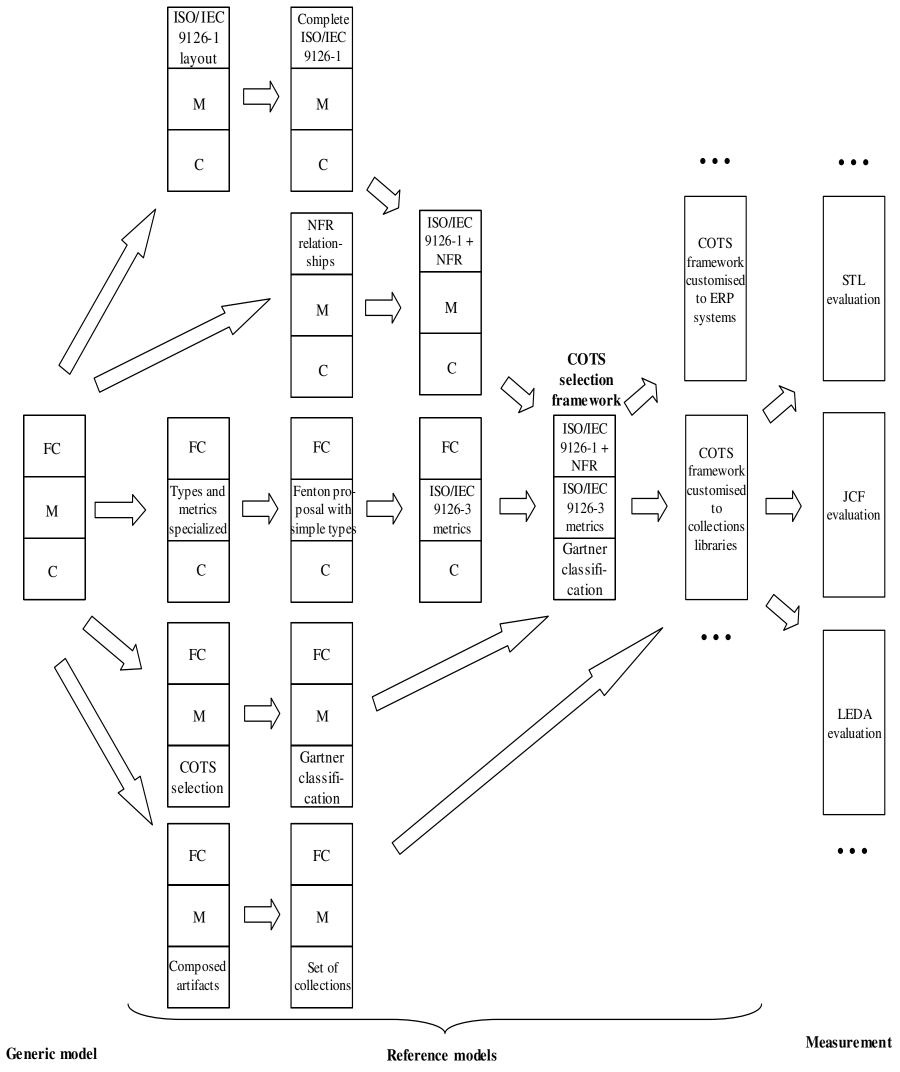


Fig. 4. General structure of the COTS selection reference model (FC: fundamental concepts; M: metrics; C: context)

- Metrics will be classified as *observations* and *formula* depending on the way to obtain attribute's values (measuring or calculating).
- Quality types may be *simple* or *enumerated*. Simple ones will include integers, strings, booleans and other frequently used types. There are also many situations in which we must assign a value taken from an enumeration (ranks, labels, …)
- There will exist an object representing each kind of influence as defined in the NFR framework (*break*, *hurt*, *unknown*, *help* and *make*). Analogously, each char-

acteristic and subcharacteristic as defined by the ISO/IEC standard will be represented by an object of the appropiate class. Links corresponding to the extension of the association *dependsOn* will reflect the decomposition of characteristics into subcharacteristics defined by the standard. On the other hand, the extension of *Influences* will take into account how the quality factors interact with each other. An excerpt of these objects and links is shown at the top of fig. 6.

- The rest of fig. 6 shows how a similar use of objects and links is going to represent software domains and categories defined by the Gartner group together with their hierarchical relations and scales, types and their compatibility.

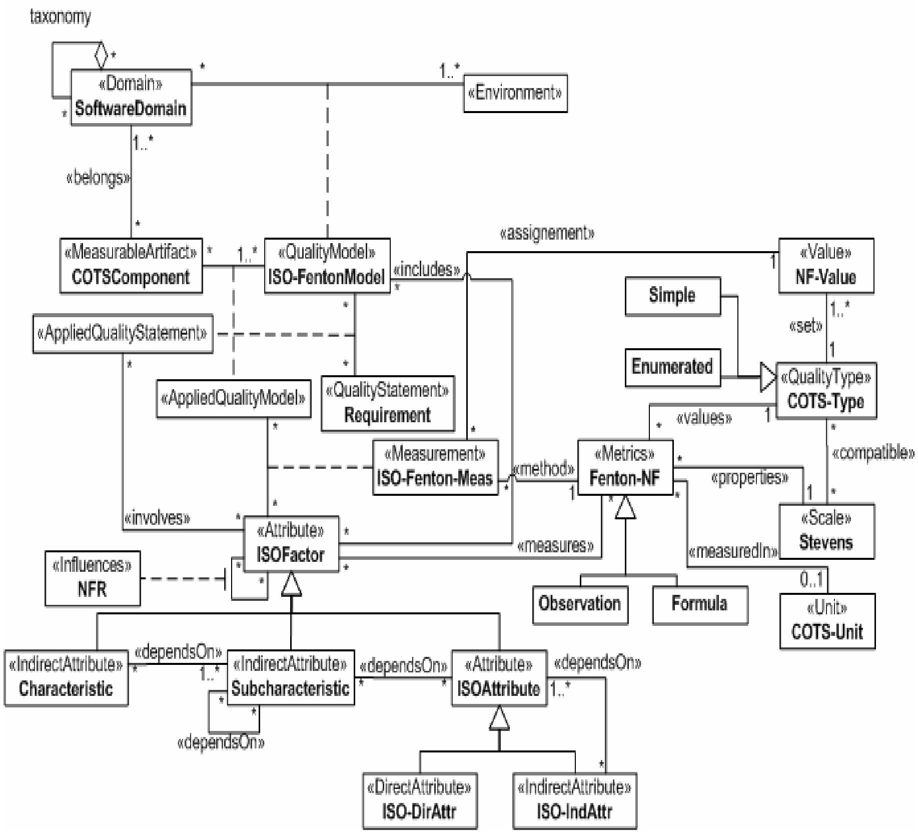


Fig. 5. Reference model for the COTS selection framework, M1 layer

Concerning the customization for the collections library case, fig. 7 summarizes the changes in both levels.

At level M1, it is just required to declare a recursive association over *Artifact* to represent the notion of artifact composition. At level M0:

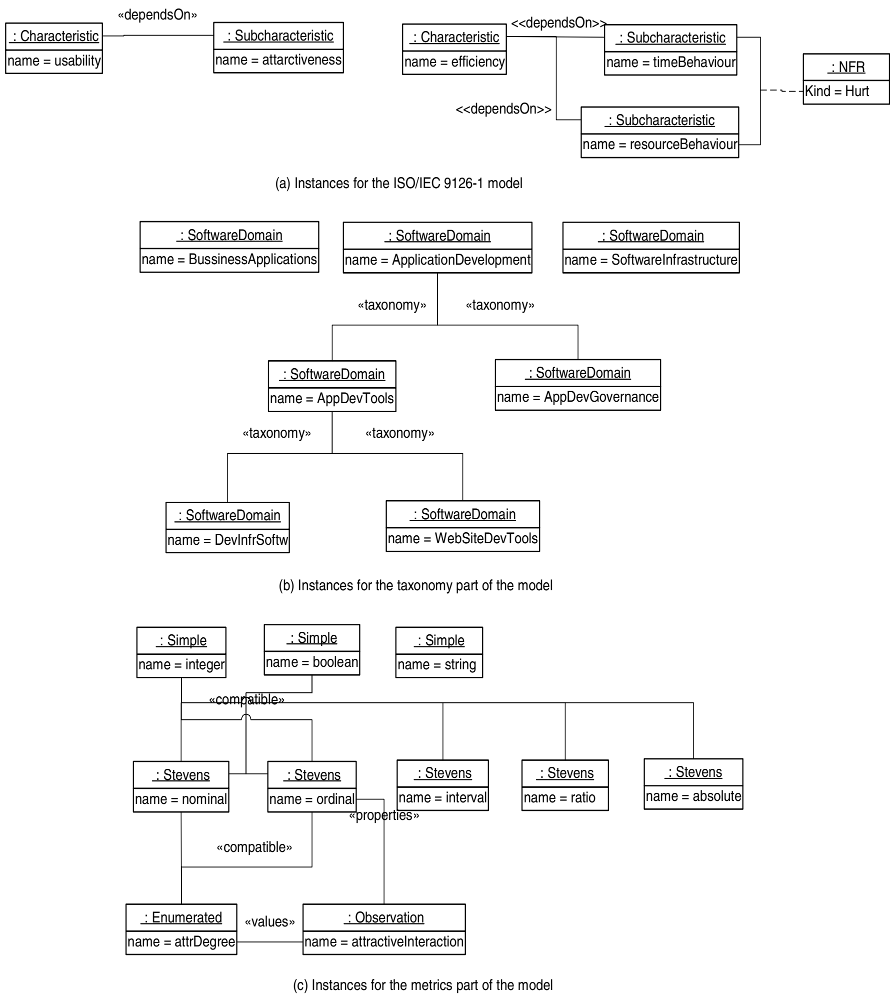


Fig. 6. The reference model for the COTS selection framework, level M0

- A new type of value is introduced for measuring the efficiency of data structures and algorithms using the big-Oh notation [3]. Correspondingly, the values for this type are also introduced: O(1) or constant; O(n) or lineal; and so on.
- The structure of collection libraries is depicted at three levels: library; collections and algorithms; data structures and operations.
- The ISO/IEC 9126-1 model is customized for this particular domain. For instance, we show how the subcharacteristic *Attractiveness* disappears, because it does not apply in this context according to its definition. The same happens with others.
- The metrics for measuring efficiency with the big-Oh notation also appear at M0.

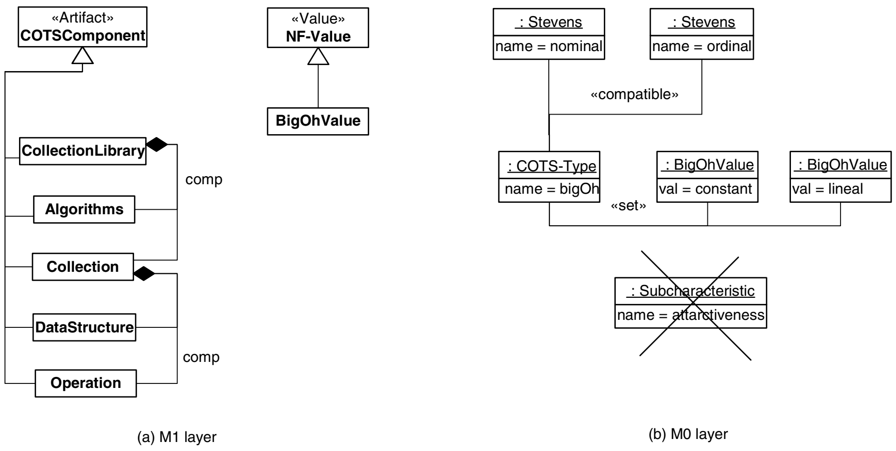


Fig. 7. The COTS selection framework customised to the collection libraries domain

6 Conclusions

The work presented here models the domain of software quality (one of the six key areas appearing at level 2 of CMM [32]) in a MOF-compliant way and overcoming other limitations of our previous work [5] mentioned in the introduction. We have collected the fundamental concepts that exist in this field considering the main issued proposals and we have built a metamodel embracing this knowledge. Concrete proposals may be derived from this metamodel to conform reference models to be used. The main contributions of the proposal are:

- *Theoretical framework*. The most important aspects of software quality are integrated into a single framework. This characteristic helps in providing a common baseline for analyzing, situating and comparing quality-related frameworks.
- *Integration with standards*. The framework is compliant with the MOF architecture and defined as an extension of the UML metamodel. This is a spread way of defining metamodels as in [33, 10, 9, 35]. It allows the reuse of concepts (e.g., classifier), methods (e.g., metamodel extension methodologies) and tools (e.g., for defining profiles) that are well know in the community. Dissemination of the proposal is also ameliorated. As a drawback, the semantics of UML has not been formalized yet. However, several groups are working in such formalization: (e.g., UML 2.0 Semantics Project - IBM, Technical University of Munich and Queen's University -, the precise UML group - www.puml.org -).
- *Applicability*. The proposal is highly structured, with a hierarchy of reference models that allow to define quality proposals in fine-grained chunks and to put them together as desired. In fact, we aim at constructing a catalogue of such chunks to be able to define quality frameworks in an easy and flexible way. We are planning to define the chunks as patterns making explicit the context of application, the prob-

lems forces they reconcile and the solution they propose. These patterns would be helpful for dealing with quality in real development projects in an effective manner.

- *Methodological guidance*. We have identified some methodological guides in the construction of quality frameworks. The scenario presented in section 2 and illustrated in section 5 is very common and the definition of levels in the hierarchy as done in the example applies to the general case.
- *Tool support*. The generic model may be the root of a hierarchy of stepwise refined models which may be developped following the above mentioned (or others) methodological guides. It is possible to build a tool to support this developing process taking care of the relationships between models and performing some correctness tests over new models with respect to the concept of correct instantiation. As the generic model is an extension of the UML metamodel this tool may be built on top of existing software able to deal with metamodeling.

Some other proposals have been proposed for dealing with software quality by means of generic frameworks. An important approach is the work by Kitchenham *et al.* [24] which defines a data model and uses it for storing experimental data. The model is not intended to be a general framework but nevertheless it is an attempt to gather quality information in a single model. More recently, there has been a joint effort in the Spanish and south-american community to produce an ontology of software quality ([16, 30]). However, this work is mainly focused on measurement. Our proposal is more comprehensive than these approaches since it is aimed at embracing the huge variety of concepts behind quality. In addition, these proposals do not provide integration with the MOF architecture or UML metamodels and do not provide this notion of stepwise refinement for building quality frameworks presented in this paper. Other proposals that do rely on UML use it as a working notation in which to develop their goal (e.g. defining metrics for databases or COTS components [4, 17] or evaluation processes [31]), not with the aim of providing ontological knowledge.

References

1. K. Arnold, J. Gosling, D. Holmes. *The Java Programming Language*. Addison-Wesley, 3rd edition, 2000.
2. V. Basili. “Goal-Question-Metric Paradigm”. In *Enciclopedya of Software Engineering”*, John Wiley, 1994.
3. G. Brassard, P. Bratley. *Fundamentals of Algorithmics*. Prentice-Hall, 1996.
4. A. L. Baroni, C. Calero, M. Piattini, F. Brito. “A Formal Definition for Object-Relational Database Metrics”. In *Proceedings of the 7th International Conference on Enterprise Information Systems*, Miami (USA), may 2005.
5. X. Burgués, X. Franch. “Formalising Software Quality using a Hierarchy of Quality Models”. In *Proceedings of the 15th International Conference on Database and Expert Systems Applications* (DEXA’04), LNCS 3180, Zaragoza (Spain), Sept. 2004.
6. A. Burton-Jones, P. Meso. “How good are these UML diagrams? An empirical test of the Wand and Weber good decomposition model”. In *Proceedings 23rd International Conference on Information Systems (ICIS’02),* 2002.
7. L. Chung, B. Nixon, E. Yu, J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Kluwer Academic Publishers, 2000.

8. J.P. Carvallo, X. Franch, C. Quer, M. Torchiano. "Characterization of a Taxonomy for Business Applications and the Relationships Among Them". In *Proceedings 3rd International Conference on COTS-Based Software Systems* (ICCBSS'04), LNCS 2959, Redondo Beach (CA, USA), Feb. 2004.
9. UML Profile for CORBA. OMG document formal/02-04-01. Available at omg.org
10. Common Warehouse Metamodel Specification. OMG document formal/2003-03-02.
11. D. G. Firesmith. "Using Quality Models to Engineer Quality Requirements". *Journal of Object Technology,* 2(5), 2003.
12. N. Fenton, S. Pfleeger. *Software Metrics:A Rigorous Practical Approach*. PWS, 1998.
13. X. Franch, J.M. Ribó. "A two-tiered Methodology for Metamodel Extension Applied to UML 1.4". Technical Report LSI-04-51-R, LSI-UPC, November 2004.
14. A. Finkelstein, G. Spanoudakis, M. Ryan. "Software Package Requirements and Procurement". In *Proceedings of the 8th IEEE International Workshop on Software Specification and Design* (IWSSD), 1996.
15. Gartner Consulting. http://www4.gartner.com/Init. Last accessed November 2004.
16. F. García, F. Ruíz, M.F. Bertoa, C. Calero, M. Genero, L. Olsina, M. Martín, C. Quer, N. Tondori, S. Abrahao, A. Vallecillo, M. Piattini. "Una Ontología de la Medición del Software". Informe Técnico UCLM DIAB-04-04-2, Febrero 2004.
17. M. Goulao, F. Brito. "Formalising Metrics for COTS". In *Proceedings of the 1st* International *Workshop of Models and Processes for the Evaluation of COTS components* (MPEC), held jointly with ICSE 2004.
18. M. Genero, G. Poels, M. Piattini. "Defining and Validating Measures for Conceptual Data Model Quality". In *Proceedings 14th International Conference on Advanced Information Systems Engineering* (CAiSE'04), LNCS 2348, Riga (Latvia), June 2002.
19. R. Glass, I. Vessey. "Contemporary Application Domain Taxonomies". *IEEE Software*, 12(4), 1995.
20. IEEE Standard 1061-1992. *Standard for a software quality metrics methodology*. 1992.
21. ISO/IEC Standard 9126-1 Software Engineering – *Product Quality – Part 1*, 2001.
22. ISO/IEC Standard 9126-2 Software Engineering – *Product Quality – Part 2*, 2003.
23. Y. Jiang, W. Shao, L. Zhang, Z. Ma, X. Meng, H. Ma. "On the Classification of UML's Meta Model Extension". In *Proceedings of the 7th UML International Conference*, LNCS 3273, 2004.
24. B. Kitchenham, R. Hugues, S.G. Linkman. "Modeling Software Measurement Data". *IEEE Transactions on Software Engineering*, 27(9), 2001.
25. S. Keller, L. Kahn, R. Panara. "Specifying Software Quality Requirements with Metrics". *System and Software Requirements Engineering* – IEEE Computer Society, 1990.
26. *MOF 2.0 Core Final Adopted Specification*. Document ptc/03-10-04.
27. D.L. Moody. "Metrics for Evaluating the Quality of Entity Relationship Models". In *Proceedings of the 17th International Conference on Conceptual Modelling* (ER'98), Singapore, LNCS 1507, November 1998.
28. D.R. Musser, A. Saini. *STL Tutorial and Reference Guide*. Addison-Wesley, 1996.
29. K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, 1999.
30. L. Olsina, M.A. Martín. "Ontology for Software Metrics and Indicators: Building Process and Decisions Taken". In *Proceedings of Web Engineering - 4th International Conference* (ICWE), LNCS 3140, 2004.
31. M. Saeki. "Embeding metrics into Information Systems Development Methods: an Application of Method Engineering Technique". In *Proceedings International Conference on Advanced Information Systems Engineering* (CaiSE03), LNCS 2681.

32. Software Engineering Institute (CMU). *The Capability Maturity Model: Guidelines for Improving the Software Process*. Addison-Wesley, 1995.
33. Software Process Engineering Metamodel (SPEM).OMG doc. formal/2005-01-06
34. S.S. Stevens. "On the theory of scale types and measurement". Science 103, 1946.
35. UML testing profile. OMG document ptc/04-04-02.
36. *UML 2.0 Infrastructure Final Adopted Specification*, document ptc/03-09-15 and *UML 2.0 Superstructure Final Adopted Specification*, document ptc/03-08-02, available at http://www.uml.org/, last accessed March 2005.
37. H. Zuse. *Framework of Software Measurement*. De Gruyter, 1998.

Conceptual Modeling Based on Transformation Linguistic Patterns

Isabel Díaz[1,2], Juan Sánchez[2], and Alfredo Matteo[1]

[1] Universidad Central de Venezuela – Laboratorio TOOLS – Escuela de Computación, Ciudad Universitaria, Facultad de Ciencias, Caracas 1051, Venezuela
{idiaz, amatteo}@kuaimare.ciens.ucv.ve
[2] Universidad Politécnica de Valencia – Dpto. de Sistemas Informáticos y Computación, Camino de Vera s/n, E-46022 Valencia, España
{idiaz, jsanchez}@dsic.upv.es

Abstract. Many object-oriented development approaches specify the system functional requirements by means of use cases. During the requirements analysis, these approaches generally agree on expressing the system behaviour using two representations: the Object Model and the Interaction Model. The Interaction Model development is subordinated to the Object Model construction, which provides the former with the basic information of the object classes. The Interaction Model contributes to identifying the class operations. Although these models are closely linked and their development is supposed to be iterative, in practice, there are no established mechanisms that guarantee the consistency of both models throughout their construction. The Interaction Model information is also under-used or underestimated in the Object Model. Persistent connections between the analysis models and the Use Case Model are not recognized. A framework to simultaneously construct the Object Model and the Interaction Model from use cases is described in this paper. Its main purpose is to strengthen the information exchange between the models so that it is complementary and consistent. To fulfil this purpose, the framework is centered on a transformation model based on linguistic patterns.

1 Introduction

The specification and analysis of software behaviour are two of the most important activities in system development [1,2]. In the first phases of its life cycle, these activities are determinant in understanding the system functionality. The behaviour specification is a description of the functional requirements that shows the information exchange established between the system and the elements in its surroundings. Nowadays, most object-oriented software development approaches specify system behaviour by means of the Use Case Model [3,4]. The analysis of the behaviour specification describes the components that integrate the system, their relationships and restrictions, and how these components exchange information to provide the system with the specified functionality in the use cases. Generally, the object-oriented software development approaches express the output of this activity by means of the Object Model and the Interaction Model. Neither of these models is sufficient by itself; they are complementary and must be used together for system design. However, in practice, the Interaction Model construction is a frequently ignored activity. While the structural model is considered to be fundamental for the system development, the dynamic model is often considered to be optional [5,6,7,8]. We believe that this situation originated from the high level of difficulty of these models, especially for inexperienced modellers. Therefore, interaction modelling must be promoted in order to take advantage of its potential during system development.

A specific framework, Metamorphosis, has been defined to achieve this objective. The main purpose of Metamorphosis is to facilitate the Interaction Model construction without playing down the Object Model. It places the dynamic representation at the same level of importance as the system structural representation. The framework assumes that the analysis models can be constructed simultaneously and iteratively, supplementing each other. This is possible if the Interaction Model construction does not depend on the Object Model information. Metamorphosis uses the Use Case Model, the Object Model, and the Interaction Model as the principal artefacts to study system behaviour. The construction of these models relies on the semantic and syntax of the Unified Modeling Language (UML) [4]. Using the extension mechanisms that this language provides, the UML metamodel of these artefacts has been enriched with the information needed to establish connections between their elements.

The main goal of this paper is to describe how the Metamorphosis framework identifies abstractions in the texts of the use cases and deduces elements of the analysis models by means of transformation patterns. These patterns are based on the linguistic information about the sentences of the use cases. Specifically, the action transformation patterns describe how these sentences are converted into parts of an Interaction Model and into parts of an Object Model. These patterns have been specified to be automated and integrated with automatic software production software. The paper has seven sections. Section 2 presents how the Metamorphosis framework has been conceptualized and its transformation strategy. Section 3 describes the structure of the action transformation patterns and the function of its elements. Section 4 shows the application process of these patterns. Section 5 presents related works. The last two sections present our conclusions and the references.

2 Metamorphosis Transformation Strategy

Metamorphosis is a framework that has been developed to facilitate the conceptual modelling of an object-oriented system in an automatic software production environment [9]. The conceptual model is obtained by means of the use case transformation. This section describes the models that participate in this transformation and the strategy that is followed.

2.1 The Use Case Model

In Metamorphosis, the Use Case Model is the fundamental input to deduce the Interaction and the Object Model. Each *use case* is considered as a behaviour unit that is described by a text written in natural language. A use case shows the complete and organized sequence of actions that the system must perform when interacting with the actors in order to fulfil a certain goal. Metamorphosis assumes the UML UseCases

Package concepts [4]. To guarantee the recognition of the linguistic properties of the use case text, elements in this package have been extended and described in the so-called Metamorphosis Use Case Linguistic Profile [9,10]. This profile is based on the action concept, which is a fundamental element in the description of use cases. An *action* can express: (i) a *communication* that is established between the actor and the system to exchange information; (ii) an internal *behaviour* of the system, which responds to the communication established with the actor. *Special* actions can also be distinguished to allow conditioning, adding, or repeating actions or groups of them.

From the *syntactic or grammatical perspective*, an action is a use case sentence. A use case is perceived as a text that consists of a sequence of sentences [11]. The sentences may be simple or special. A *simple sentence* represents a communication or behaviour action that can be described as a set of words, each of the which has a syntactic category (i.e., adjective, verb, etc.). These words can also form groups, according to the grammatical function that they fulfil in the sentence, configuring phrase structures (i.e., noun phrase, prepositional phrase, etc.). *Special sentences* are distinguished by having a predefined format that uses key words (i.e. 'INCLUDE' and 'REPEAT'). The simple sentences can be atomic or complex. Unlike the atomic sentences, the complex sentences can be decomposed into clauses. Each clause is a simple sentence that keeps the same object and the same verb as the original sentence.

In order to achieve a reasonable balance between documentation and expressiveness, Metamorphosis assumes the application of only those style and content guidelines that are mandatory to guarantee that the use cases fulfil their purpose in the system development [12]. These guidelines are the following: (i) sentences must be declarative, affirmative, and active; (ii) each sentence must have a single subject and a single main verb; (iii) the main verb of the sentence must be transitive, ensuring the presence of the direct object; and (iv) the sentence terminology must be normalized, controlling the use of synonyms.

From the *semantic perspective*, a use case action can be expressed as a relationship established between one or more *semantic roles* [13,14]. A sentence can be univocally characterized by the semantic roles that it contains. Each role denotes an abstract function performed by an element that participates in an action. This function is defined independently from the syntactic structure of the sentence. These role properties allow us to specify generic patterns, independent from the language that is used to write the use cases. The roles used by Metamorphosis can be: (i) *primary roles,* if they always participate in a transitive sentence such as *agent* (what/who performs an action) and *object* (what/who undergoes an action); and (ii) *secondary roles,* if they are not required by the verbal action and always appear linked to a primary role. Some secondary roles are: *destination* (receiver/beneficiary of the action), *owner* (what/who possesses), *owned* (possessed entity), *state* (entity status), *location* (where an action takes place); and *time* (when the action occurs).

2.2 The Conceptual Model

The Metamorphosis Conceptual Model is expressed by means of an Object Model and an Interaction Model [4]. The *Object Model* represents the system internal components (objects/classes) and their relationships and restrictions. The purpose of the *Interaction Model* is to show how these components exchange information to achieve the described functionality in the use cases. Interactions are the basic elements to express the analysis of the specified behaviour. An *interaction* describes the information exchange established between two or more instances to communicate with each other [4]. In Metamorphosis, these instances are considered to be *class objects* that describe the system internal composition. The classification proposed by Jacobson is assumed (border, entity, and control classes) [3]. The information exchange or messages among these instances represent *operations* of these classes.

The elements used by Metamorphosis to express the conceptual model have been specified in two profiles: the Structure Profile and the Interaction Profile [15]. These profiles are extensions of the UML Class Package and the UML Interaction Package, respectively [4]. The notation is also UML 2.0 compliant. Specifically, sequence diagrams are used to represent the interaction fragments.

If the system's behaviour analysis is based on an action, the result is a structure fragment and an interaction fragment. A *structure fragment* is a part of the Object Model. It is formed by at least one class. This fragment is deduced from a use case action. An *interaction fragment* is also an interaction that forms part of another interaction [4]. It can be formed by one or more instances that interchange one or more messages. An interaction fragment is derived from a use case action. The integration all structure fragments and of all interaction fragments that are obtained from all the use cases, allow the construction of the system Conceptual Model.

2.3 The Action Transformation Strategy

The most important activity of the Metamorphosis framework is the transformation of actions. The complete transformation of a use case depends on the transformation of each action that belongs to the use case. The action transformation strategy applied by Metamorphosis is based on patterns. The purpose of the transformation patterns is to capture transformation knowledge and reuse it suitably. Patterns specify the way that actions are transformed. The pattern specification shows how an action is converted into a structural fragment and into an interaction fragment [15]. The roles are used to describe both the action and the fragments. The specification of a transformation pattern is generic, domain-independent, and implementation-independent. The patterns are also independent of the language used to write the sentences and their syntactic structures. To obtain the desired structural and interaction fragments, a transformation pattern must be applied. Thus, a sentence is transformed into specific fragments. This requires knowing the syntactic structure associated to the roles that participate in the sentence.

The action transformation patterns allow the identification of certain elements of the system Conceptual Model (i.e. lifelines, classes, and messages). To identify other elements and to verify the consistency of the Conceptual Model groups of actions must be analyzed. Thus, the information generated for each transformed action must be integrated. In addition, the information that can only be deduced through either a partial or complete analysis of the use case must be incorporated into the obtained fragment. For example, two transformation activities performed from the use case text are: (i) to combine the interaction fragments that are deduced from each sentence of a use case until the interaction is completed; and (ii) to deduce the candidate parameters of each message. Finally, the integration of the fragments deduced for each use case allows the system Conceptual Model to be constructed. This task must resolve the possible conflicts that are generated when all partial representations are combined.

3 Specification of an Action Transformation Pattern

An action transformation pattern is described in four parts. The first part is the pattern *heading* which contains its identification and the action description that will be transformed. The second part is the pattern *body* that is composed of the transformation rules and the results obtained by its application. The third part is dedicated to the *role description* used by the pattern. Finally, the fourth part is the pattern *foot*, which contains observations about the pattern (this part won't be described in this paper because it is not relevant). The elements that specify a pattern are explained in this section and an example is presented.

3.1 The Pattern Heading

The pattern heading is composed of the following elements: the pattern name, the action description, and the application context description of the pattern. Table 1 shows the simplified version of the heading of a Metamorphosis pattern.

(i) Name. It is the identification of a pattern that distinguishes it from the others. The `Going Downstairs` name illustrates the action type and the transformation solution that describes this pattern (Table 1).

(ii) Action description. It is a concise explanation about an action that must be transformed by means of the specified pattern. Language natural is used to express this description in order to facilitate its initial comprehension. The action is described from a conceptual and semantic perspective.

(iii) Action context. It describes the canonical form of an action in terms of semantic roles. The context indicates the roles that must be identified in an action so that the transformation rules can be applied. It also establishes the frequency, restrictions, and conditions of these roles. An action context is specified by means of:

- A *name* that distinguishes an action context from the others.
- A *formula* that allows determining if an action fulfils a context. An action context (AC) is specified using a logic formula AC=<α, μ, ψ>, where α is a set of variables that represent roles, μ is a set of auxiliary constants, and ψ is a set of functions that are applicable to the AC terms. The formula of the `Owning Chain` context uses the following action roles: `agent`, `object`, `destination`, `state`, `owner` and `owned` (Table 1). `BasicAction`, `Ownership` and `SetState` are Boolean functions. The application of the `NumberOf` function produces an integer value. The use of the symbol "_" indicates the value absence of a variable.
- A *graphic representation* of the action context that describes its roles and the relationships established among them. Its main purpose is to facilitate comprehension of the action context. This graph shows the roles that participate in

the context and its restrictions by means of a UML class diagram. To indicate that the classes of this diagram represent roles, the `<<role>>` stereotype was introduced. The graphic expression of this stereotype is a straight slash placed in front of the role name ("/`roleName`"), as is suggested in [16]. The `Owning Chain` diagram indicates that an only `agent` instance participates in the action context (Table 1). However, in this context, two or more `Ownership` must be established. The information of an action context diagram can be complemented with other types of specifications (i.e. restrictions and properties).

- A *description* in natural language about the action context.

3.2 The Pattern Body

The pattern body supplies information about the expected solution. This solution is the outcome of the action transformation into analysis model fragments. These fragments express the action analysis from the dynamic and static perspectives. Table 2 shows the pattern body whose heading is specified in Table 1 (the `Going Downstairs` pattern). The elements that describe the pattern body are the following:

(i) Transformation Rules. They describe how the participants of an action context are turned into elements of a fragment through the use of roles. These rules generate both the dynamic fragment and the static fragment at the same time. The left side of a rule corresponds to the roles used to specify the interaction fragment as well as the structural fragment; i.e. `initiator`, `bridge` and `script` are fragment roles and are localized on the left side of the rule (see Table 2). The right side of each rule shows how to identify the fragment elements. To recognize these elements, a formula that applies functions to action context roles is used.

For example, in order to identify the i^{th} `bridge` of a fragment, the `Kernel` function is applied to the j^{th} `owner` in the action context. This function extracts the most important constituent of the `owner` (in syntactic terms, this constituent is the noun head used to `owner`). The result obtained when the `Kernel` function is applied must be normalized. The `Norm` function expresses this result in its canonical form. Another rule in Table 2 uses the difference function (~) that takes all the components from the `owner` that does not belong to its `Kernel`. Thus, the i^{th} `attach` is obtained from the j^{th} `owner` constituents. The signature of the `script` element is deduced by a `Sequence` function. This function constructs a label with the `verbalAction` and the previously normalized `object`.

(ii) Dynamic Model Fragment. When an action is transformed, an interaction fragment is obtained. In Metamorphosis, the interaction fragments are specified by means of a generic structure. This structure uses roles to represent several interaction types and to add semantics to improve its comprehension [16]. The interaction fragment generated from an action that satisfies the pattern context is described using: a *name* that distinguishes one interaction fragment from the others; a *graphic representation* that shows the element layout that participate in the interaction pattern. These elements are represented by means of roles and a *description* in natural language about the interaction fragment.

Table 1. The heading of an action transformation pattern

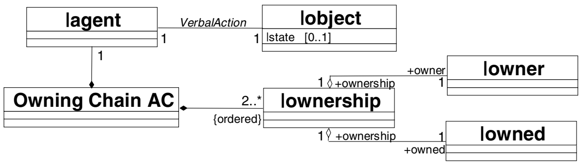
NAME	GoingDownstairs	
ACTION DESCRIPTION	This pattern can be applied to an action that represents an internal behaviour of the system. The action indicates a state change of a single system component. In order to carry out this change, this component or entity is found by means of other entities that are its owners. Each owner entity belongs to another owner entity until the relevant entity is recognized. The action explicitly expresses the ownership relation between the entity that will be changed and its owner entities.	
ACTION CONTEXT	**Name**	Owning Chain
	Formula	$\forall$V,A,O,St,Or,Od: $\exists$n>1 / NumberOf(Ownership)=n $\wedge$ BasicAction(verbalAction:V,agent:A,object:O,destination:_) $\wedge$ (SetState(object:O,state:St) $\vee$ SetState(object:O,state:_)) $\wedge$ ($\forall$i=1..n $\text{Ownership}_i(\text{owner}_i$:Or,$\text{owned}_i$:Od) $\wedge$ owned_1:O $\wedge$ $\text{owned}_i:\text{owner}_{i-1}$);
	Graphic Representation	 Iagent VerbalAction 1 1 Iobject Istate [0..1] 1 +owner Iowner 1 +ownership 1 Owning Chain AC 2..* {ordered} Iownership 1 +ownership 1 Iowned +owned
	Description	This context describes actions that have an active entity that initiates or controls the action (the agent) and a passive entity on which such action falls (the object). The action does not have an entity that fulfils the destination role (this is indicated by the symbol "_"). The state change undergone by the object (as a consequence of the action performed) may or may not have been explicitly expressed. The action must have more than one Ownership relationship between an owner entity and an owned entity. The NumberOf function is used to determine the number of Ownership relationships that are present in the action. The first of these relationships is established between the object and an owner entity. The remaining Ownership relationships are established among entities such that an owner entity in an Ownership is an owned entity in the following Ownership relationship. The agent and object roles must participate only once in the action so that the pattern can be applied. The state role is an attribute of the object role which may or may not be present in the action. It must have two or more Ownership relationships established in the action.

Table 2 describes the Domino Effect Interaction Fragment. This representation can be obtained using the specified transformation rules. The information contained in this representation is enriched with the semantics provided by the UML elements [4]. For example, it can be observed that the messages are synchronous. The execution of each message activates the execution of another message; the execution of a message is not completed until the activated message has finished. Other transformation patterns apply UML operators to specify message concurrency, choice of alternative traces, or weak/strict sequencing of the messages.

(iii) Structural Model Fragment. It is a part of the system Object Model that is deduced from an action. This part is described by means of a generic structure. Roles are used to strengthen the semantics of this structure [16,17]. A structural fragment has: a *name* that distinguishes it from the others; a *graphic representation* that shows the layout of the elements that participate in the interaction pattern and a *description* in natural language about the interaction fragment.

A structural fragment can be obtained using the transformation rules specified by the pattern. However, the deduction of a structural fragment is independent of the generation of the interaction fragment corresponding to the same action. These fragments can be obtained simultaneously or each one can be obtained separately at different times. Although the structural and interaction fragments share information that is obtained using the same rules, their deduction process can be carried out independently. The consistency of the structural fragments and the interaction fragments is guaranteed by the application of the same action transformation rules.

Because the structural fragment is generated in the context of an action, the information presented can be incomplete. The fragment representation must be complemented with the information obtained from other actions. To express this limitation, the information that can be changed is introduced inside brackets. This can happen with the multiplicity and the parameters of a structural fragment. Table 2 shows the `Initiator-Performer Bridge Structural Fragment`. The information presented is enriched with UML elements [4]. Thus, the `{optional}` constraint applied to the `initiator` class indicates that this class may or may not participate in the fragment. This depends on the information that is required in the model.

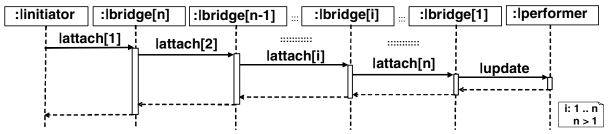
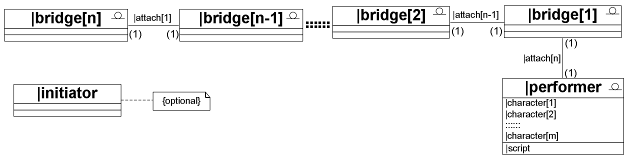
3.3 Role Description

In order to facilitate the comprehension of the transformation patterns, a role model is also presented [16,17]. In this model, a role defines a subtype of a metamodel class or a base metaclass. The roles specify the properties that a model element must have in order to be part of a transformation pattern. The specialization with roles is fulfilled from the Metamorphosis Interaction and Structure Profiles. This is represented using a class diagram, but other specifications can be used to complement this information.

Figure 1 presents a part of the role model corresponding to the `Going Downstairs` pattern. The role model of the `Initiator-Performer Bridge Structural Fragment` has the following characteristics. The `bridge` and `performer` represent entity classes in the system domain. There are one or more `bridge` classes but only a `performer` class can be contained in the structural fragment. `initiator` represents a border class or a control class. If there is an `initiator` class in the structural fragment, there is only one. The fragment has one or more `attach` relationships. The `character` represents a class attribute. A class can have zero o more explicitly specified attributes. Only a `script` operation participates in the structural fragment.

The same role can represent several metaclasses in both the interaction fragment and in the structural fragment. For example, the `attach` role is a type of association relationship in `Initiator-Performer Bridge Structural Fragment`. However, this role is a type of message in `Domino Effect Interaction Fragment`. In addition, the same role can be used by several patterns, but it must maintain its semantics when the role participates in the same fragment type.

Table 2. The body of the Going Downstairs pattern

TRANSFORMATION RULES	initiator	← agent
	bridge[i]	← <Kernel(owner$_k$)> **NORM** ∀i=1..(n-1) ∧ ∀k=(n-1)..1
	performer	← <Kernel(owner$_1$)> **NORM**
	attach[i]	← Owner$_k$~Kernel(owner$_k$) ∀i=1..(n-1) ∧ ∀k=n..1
	update	← Sequence(verbalAction,<object>^{NORM},state)
	character[i]	← <Kernel(object)^{NORM}>$_k$ ∀k=1..m ∧ GetState(object)<>"_"
	script	← Sequence(<verbalAction>^{NORM},<Kernel(object)>^{NORM})
DYNAMIC MODEL FRAGMENT	**Name**	Domino Effect Interaction Fragment
	Graphic Representation	
	Description	Three lifeline roles are used: (i) initiator is played by an instance of an initiator class; (ii) bridge[i] is played by the ith bridge instance in the set of bridges; and (iii) performer is played by an instance of a performer class. The messages are synchronous. An initial message is sent by an initiator instance to a bridge instance. The message attach relates a sender instance to a receiver instance. This message induces the receiving instance to send another message to another bridge instance and so on, until a bridge instance sends a message to a performer instance. This message activates the update operation that changes the performer instance state.
STRUCTURAL MODEL FRAGMENT	**Name**	Initiator-Performer Bridge Structural Fragment
	Graphic Representation	
	Description	The following roles denote three different types of object classes: initiator, bridge and performer. These classes are entity classes except for the initiator class. In the action context, one or more attributes can be assigned to the performer class. character plays the role of these attributes. In addition, the performer class has at least one operation. The script represents this operation, which can have parameters. attach is a type of association relationship established among the classes. Some or all these relationships can be aggregations and/or compositions. In the action context, the multiplicity of the class relationships is 1 to 1.

Twenty-five transformation patterns have been specified so far. The actions of these patterns correspond to simple sentences (atomic and complex) or special sentences (see Section 2.1). The dynamic aspect of these patterns was validated empirically. A validation strategy was designed in order to determine whether the patterns generated the interaction fragments expected by expert analysts. This experience also allowed us to both identify new patterns and to improve the specified

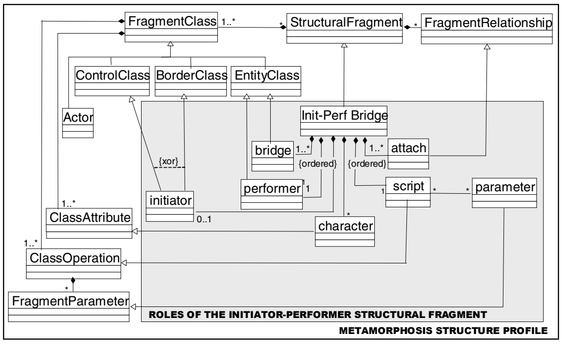


Fig. 1. The role description of the `Going Downstairs` pattern (partial view)

patterns. A grammar that describes syntactic information of sentences written in Spanish was defined to validate the patterns. In addition, a tool to automatically obtain the interaction fragment from each sentence was developed. This tool was integrated into a software automatic production environment [9]. Although the obtained results have been positive, the validation strategy must be reinforced and extended in order to prove the structural fragments of the specified patterns.

4 Application of an Action Transformation Pattern

To apply a transformation pattern to a particular sentence, this pattern must be instantiated. This consists of assigning a grammatical expression to the semantic roles used to specify each pattern. For example, the `agent` role may be linked to a noun phrase and the `owner` role to a prepositional phrase that contains an "`of`" preposition. At the instantiation level, the patterns are dependent on the language used to write the use cases. A grammar can be used to specify this match.

The application of a transformation pattern assumes the action is a sentence that fulfils the style and content guidelines indicated in Section 2.1. The sentence must also contain the corresponding syntactic and semantic information (constituents/ phrases and semantic roles). This implies that: (i) the roles of the sentence have been identified; and (ii) each role identified is linked with a phrase. Thus, the action can be analyzed as a sentence from the syntactic perspective. The application of a pattern obtains the structural and interaction fragments that correspond to a particular sentence. This process is described in an example later on. The way how the resultant fragments are integrated with other fragments previously obtained is also shown. The sentence of the Table 3 will be used as an example. The information contained in this sentence is presented also in Table 3.

Basically, the application process of any pattern can be performed in two steps.

The first step: to recognize the pattern that must be applied. It implies determining which action context fulfils a sentence. The contexts specified by each pattern are used for this purpose. A sentence can only fulfill only one context. There is only one action context for each transformation pattern. The example sentence satisfies the `Owning Chain Context` (see Table 1). Table 4 shows the formula evaluation of this context. Because the `Owning Chain Context` belongs to the `Going Downstairs` pattern, this pattern must be applied to the example sentence.

Table 3. Example sentence

"*The system verifies the deadline of the rental contract of each lessee of the real estate agency*"

{ *The* (*system*)$_{\text{head}}$ }$_{\texttt{noun-phrase/subject}\textbf{(agent)}}$
{ *verifies* }$_{\texttt{main-verb}}$
{ [*the* (*deadline*)$_{\text{head}}$]$_{\texttt{noun-phrase/direct-object}\textbf{(object/owner)}}$
[(*of*)((*the*)(*rental contract*)$_{\text{head}}$)$_{\texttt{noun-phrase}}$]$_{\texttt{of-prepositional-phrase}\textbf{(owned/owner)}}$ }
{ [*of* (*each* (*lessee*) $_{\text{head}}$)$_{\texttt{noun-phrase}}$]$_{\texttt{of-prepositional-phrase}\textbf{(owned/owner)}}$
[(*of*)((*the*) (*real estate agency*) $_{\text{head}}$)$_{\texttt{noun-phrase}}$]$_{\texttt{of-prepositional-phrase}\textbf{(owned/owner)}}$ }

Table 4. Example of sentence context evaluation

"*The system verifies the deadline of the rental contract of each lessee of the real estate agency*"

`NumberOf(Ownership)=3` ∧ `SetState(object:`'the deadline'`,state:'_')` ∧
`BasicAction(verbalAction:` 'verifies'`,agent:` 'The system',
`object:` 'the deadline'`,destination:'_')` ∧
Ownership$_1$(owner$_1$: 'of the rental contract', owned$_1$: 'the deadline') ∧
Ownership$_2$(owner$_2$: 'of each lessee', owned$_2$: 'of the rental contract') ∧
Ownership$_3$(owner$_3$: 'of the real estate agency', owned$_3$: 'of each lessee') ∧
owned$_1$: 'the deadline' ∧ owner$_1$:owned$_2$ ∧ owner$_2$:owned$_3$;

The second step: to obtain the structural and interaction fragments. This consists of applying the transformation rules of the identified pattern. When the transformation rules of the `Downstairs` pattern are applied to the example sentence, the followings fragments are obtained: `Domino Effect Interaction Fragment` and `Initiator-Performer Bridge Structural Fragment` (see Table 2). Figures 2 and 3 show the graphic representation of these fragments. The elements of these fragments were obtained as follows:

(i) A border/control lifeline/class (*initiator*) whose name is extracted from the *noun-phrase* head of the *agent* (*System*).
(ii) Two lifelines/classes (*bridge roles*): *Real Estate Agency* and *Lessee*. These elements were recognized from the *noun-phrase* head of *of-prepositional-phrases* (*owner roles*). The canonical form of each head was verified.
(iii) A lifeline/class (*performer role*): *Rental Contract* obtained from the *noun-phrase* head of an *of-prepositional-phrase* (*owner role*). The head canonical form was determined.
(iv) Two synchronous-messages/association-relationships (*attach roles*): *of the* and *of each*. They are obtained using *of-prepositional-phrases* (*owner roles*) as follows: the difference function (~) takes all the components from the `Owner` that does not belong to its `kernel`.

(v) A synchronous-message (*update role*): *Verifies the dealine*. It is deduced by a `Sequence` function. This function constructs a label using information from the *verbalAction, object*, and *state roles*.

(vi) An operation (*script role*): *Verify deadline*. It is deduced by a `Sequence` function. This function constructs a label using information from the *verbalAction* and *object roles*.

(vii) No attribute was identified (*character role*) because the `object state` was not explicitly expressed in the example sentence (*state role*).

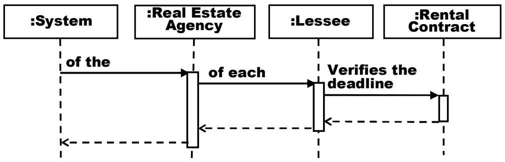


Fig. 2. An example of an Interaction Fragment

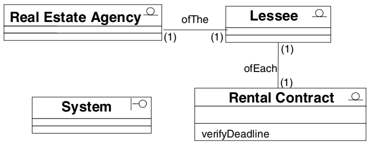


Fig. 3. An example of a Structural Fragment

In order to deduce the conceptual model whose scope is a use case, the fragments of each of their actions must be integrated. The integration process consists of combining the fragments of an action with the partial representation obtained up to the previous action. The fragment integration is carried out "action by action" in the same order established by the action sequence of the use case. A partial representation is the result of combining: (i) the fragments obtained from two actions; or (ii) a fragment with a partial representation. The integration process must guarantee the consistency (resolution of conflicts) and the completeness of the conceptual model.

Next, we are going to show how the fragments obtained from example sentence (Figures 2 and 3) are integrated with the fragments deduced from another sentence that belongs to the same use case. Table 5 presents a partial view of the action transformation pattern applicable to sentences such as the following: "*The system registers the personal details of the lessee and the address of the property in the rental contract*". Figure 4 and Figure 5 present the fragments obtained when this pattern is applied to the sentence.

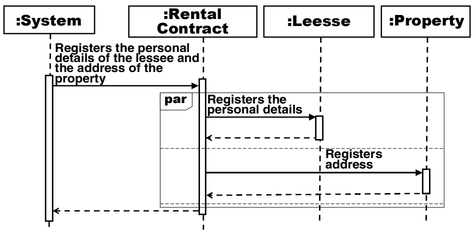

:System
:Rental Contract
:Leesse
:Property
Registers the personal details of the lessee and the address of the property
par
Registers the personal details
Registers address

Fig. 4. Another example of an Interaction Fragment

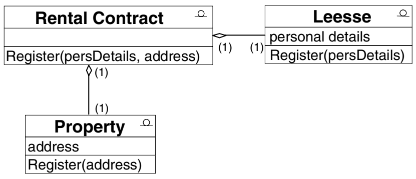
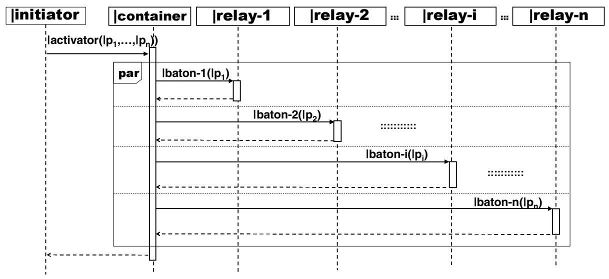
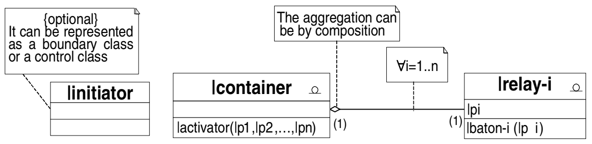


Fig. 5. Another example of a Structural Fragment

In order to obtain the partial representation of the conceptual model, two basic rules are applied: (i) the fragment elements to be integrated that are not in the partial

Table 5. Another action transformation pattern (partial view)

Pattern Name	Concurrent Relief by the Right	
Description	This pattern can be applied to actions that represent an internal behaviour. The action produces a change or makes a consultation of the properties of a system entity. This entity sends concurrent messages to other entities in order to change or to consult their properties. Grammatically, the action is a sentence that can be decomposed into two or more clauses. Each clause keeps the `agent`, the `verbalAction` and the `destination` of the `object` that has the complete action. The clause `object` can be equal or different to the `object` of the other clause. Each clause always has a single `ownership` relationship established between the `object` (the `owned`) and an `owner` entity. The `object` of each clause may or may not be linked to a `state` explicitly.	
Action Context	**Name**	Distributed Action by the Right
	Formula	$\forall$V,A,O,V,Or,Od,S: $\exists$n>1 / NumberOf(Clauses)=n $\wedge$ ($\forall$i=1..n, Clauses$_i$(verbalAction:V,agent:A, objet$_i$:O$_i$,destination:D) $\wedge$ Ownership$_i$(owner$_i$:Or$_i$,owned$_i$:Od$_i$) $\wedge$ (SetState$_i$(objet$_i$:O$_i$,state:S$_i$) $\vee$ SetState$_i$(objet$_i$:O$_i$,state:_)));
Transformation Rules	initiator	$\leftarrow$ agent
	container	$\leftarrow$ <Kernel(destination)> $^{\text{NORM}}$
	executor[i]	$\leftarrow$ <Kernel(owner$_i$)> $^{\text{NORM}}$ $\forall$i=1..n
	activator	$\leftarrow$ Sequence(<verbalAction>$^{\text{NORM}}$, Ownership<object$_i$,owner$_i$>) $\forall$i=1..n
	update[i]	$\leftarrow$ Sequence(<verbalAction>NORM,<object$_i$>$^{\text{NORM}}$) $\forall$i=1..n
	p[i]	$\leftarrow$ <Kernel(object$_i$)$^{\text{NORM}}$> $\forall$i=1..n
Interaction Fragment	**Name**	Pass on Batons
	Graphic Representation	 Iinitiator Icontainer Irelay-1 Irelay-2 Irelay-i Irelay-n Iactivator(Ip$_1$,...,Ip$_n$) par Ibaton-1(Ip$_1$) Ibaton-2(Ip$_2$) Ibaton-i(Ip$_i$) Ibaton-n(Ip$_n$)
Structural Fragment	**Name**	Container of Batons
	Graphic Representation	 {optional} It can be represented as a boundary class or a control class Iinitiator The aggregation can be by composition ∀i=1..n Icontainer Iactivator(Ip1,Ip2,...,Ipn) (1) (1) Irelay-i Ipi Ibaton-i (Ip i)

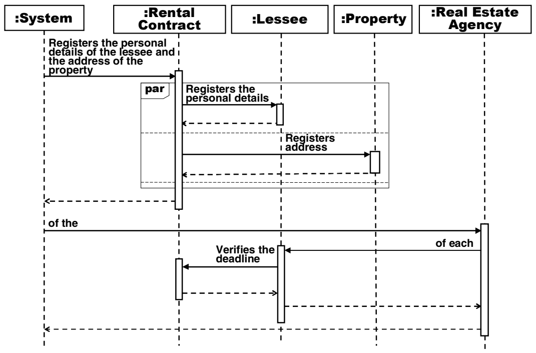


Fig. 6. Interaction Model

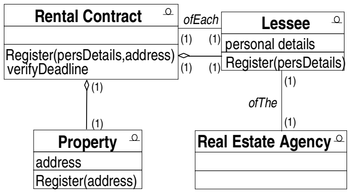


Fig. 7. Object Model

representation obtained up to that point are added to the partial representation with all their properties (i.e. lifelines, association relationships, etc.); (ii) the fragment elements to be integrated that are in the partial representation must be checked to guarantee that the properties of these elements are kept. An integration conflict is generated when the properties of the same element take a different value both in the fragment to be integrated and in the partial representation. Depending on the type of property, some rules are applied to resolve the conflict. For example, if the multiplicity of a relationship established between classes is different, then the greatest multiplicity is taken into account. This rule determines whether the element must be modified or kept in the partial representation after the fragment integration. Some conflicts can only be resolved by carrying out a thorough analysis of the overall use case or system. This is the case of some class attributes and message or operation parameters. The study of the integration conflicts are beyond the scope of this paper.

The result of integrating the fragments obtained from the sentences used as examples are presented in Figures 6 and 7. To illustrate the integration process, it was assumed that the action "*The system verifies the deadline of the rental contract of each lessee of the real estate agency*" (Figures 2 and 3) is located in the use case after the action "*The system registers the personal details of the lessee and the address of the property, in the rental contract*" (Figures 4 and 5). Figure 6 shows the partial representation obtained when the lifeline `Real Estate Agency` and the messages described in Figure 2 are introduced. Figure 7 incorporates an association relationship (`ofEach`) between the classes `Rental Contract` and `Lessee` with the fragment represented in Figure 5. The class `Real Estate Agency` was also incorporated as well as an association relationship (`ofThe`) between this class and the class `Lessee`. The complete conceptual model of a use case can be achieved by applying a similar strategy.

5 Related Works

In this paper, we have proposed a framework to simultaneously construct the Object Model and the Interaction Model from textual descriptions of use cases. To the best of our knowledge, there is no other approach that proposes such a construction process. However, there are some partially related approaches. The linguistic-oriented approaches are based on the application of Natural Language Processing techniques (i.e., parsers, grammars, semantic roles, etc.). These approaches are very effectives to identify some primitives of the conceptual model [18,19,20]. However, the models can not be obtained directly (intermediates models are required) and the participation of the analyst is very necessary to complete them. The heuristic-oriented approaches are based on the application of "step to step" procedures to construct a complete and robust conceptual model [3,5,8]. The weaknesses of these approaches are the strategies proposed to recognize the model primitives. Other approaches try to achieve a balance between the linguistic and heuristic approaches [21,22]. These approaches generate interaction models or structural models independently without to establish links between them.

6 Conclusions and Further Work

This paper describes a framework created to obtain conceptual model fragments from the sentences of a use case. This framework defines an automatic transformation strategy based on patterns that use the linguistic information of use cases. This strategy establishes persistent links among the use cases and the corresponding conceptual fragments. The patterns register the knowledge regarding the action transformation of use cases into conceptual model fragments. This knowledge is captured as reusable and generic solutions. The patterns facilitate the study of the correctness, completeness, and consistency of the designed transformations. They also contribute to the documentation, creation, understanding, application, and maintenance of this knowledge. In addition, the patterns are based on roles to provide special meaning to the model elements and facilitate their recognition. The roles allow the patterns to be independent from the language that is used to write the use cases.

The Metamorphosis transformation strategy can simultaneously generate both the Object Model fragment and the Interaction Model fragment from each sentence of a use case. However, the construction of a fragment is independent of the construction of the other fragment. The consistency of the structural fragments and the interaction fragments is guaranteed by the application of the same action transformation rules to obtain the elements that are shared by both types of fragments.

Our work is currently oriented towards establishing a permanent validation strategy of the transformation patterns for the purpose of guaranteeing its timely customization and expansion. This involves identifying the new action contexts, designing their corresponding fragments, and verifying that such correspondence is valid. The definition of an evolution and integration strategy of the fragments is also required. It seeks to guarantee the consistency when these models undergo changes as well as guarantee the traceability between the different elements.

Acknowledgments

This work has been developed with the support of: (i) the project Destino Nº TIN2004-03534, MEC, Spain; and (ii) the Council of Humanistic and Scientific Development of the Central University of Venezuela (CDCH/UCV).

References

1. Van Lamsweerde A.: Requirements Engineering in the Year 2000: A Research Perspective, Proc. of the 22nd Conference on Software Engineering (ICSE'00), pp. 5-19, ACM Press.
2. Nuseibeh B., Easterbrook S.: Requirements Engineering: A Roadmap, in Proceedings of the 22nd Conference on Software Engineering (ICSE'00), Conference on The Future of Software Engineering, pp. 37-46, ACM Press.
3. Jacobson I., Christerson M., Jonsson P., Övergaard G.: Object-Oriented Software Engineering. A Use Case Driven Approach, Addison-Wesley, 1992.
4. Object Management Group: Unified Modeling Language: Superstructure Specification, Version 2.0, August 2003, http://www.omg.org/uml.
5. Larman C.: Applying UML and Patterns: an Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd ed), Prentice-Hall, USA 2004.
6. Rosenberg D., Scott K.: Use Case Driven Object Modelling with UML: a Practical Approach, Addison-Wesley Longman, Inc., USA, 1999.
7. Song I.-Y.: Developing Sequence Diagrams in UML, in A.R. Tawil, N.J. Fiddian & W.A. Gray (Eds.), Proceedings of the 20th International Conference on Conceptual Modeling: (ER'01), p.p. 368-382, Springer-Verlag, Berlin, 2001.
8. Hilsbos M., Song I.-Y.: Use of Tabular Analysis Method to Construct UML Sequence Diagrams, in Proceedings of the 23th International Conference on Conceptual Modeling (ER'04), pp. 740-752, Springer-Verlag, Berlin, 2004
9. Díaz I., Moreno L., Fuentes I., Pastor O.: Integrating Natural Language Techniques in OO-Method, in Proceedings of the Sixth International Conference on Intelligent Text Processing and Computational Linguistics (CICLing'05), LNCS 3406, 560-571, Springer-Verlag, 2005.
10. Rolland C., Ben-Achour C.: Guiding the Construction of Textual Use Case Specifications. Data & Knowledge Engineering 25(1998), 125-160. Elsevier Science B.V.
11. Díaz I., Losavio F., Matteo A., Pastor O.: A Specification Pattern for Use Cases, Information & Management, Vol. 41/8 (2004), pp. 961-975, Elsevier Science B.V.
12. Ben Achour C., Rolland C., Maiden N.A.M., Souveyet C.: Guiding Use Case Authoring: Results of an Empirical Study, in Proceedings of the Fourth IEEE International Symposium on Requirements Engineering (RE'99), pp. 36-43.
13. Gildea D., Jurafsky D.: Automatic Labeling of Semantic Roles, Computational Linguistics, 28(3): 245-280, 2002.
14. Fillmore Ch.: The Case for Case. In Universals in Linguistic Theory, ed. By Bach & Harms. New York: Holt, Rinehart & Winston. 1968.
15. Díaz I., Moreno L., Pastor O., Matteo A.: Interaction Transformation Patterns based on Semantic Roles, in Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'05), LNCS 3513, pp. 239-250, Springer-Verlag, Junio 2005.
16. France R., Kim D., Ghosh S., Song E.: A UML-Based Pattern Specification Technique, IEEE Transactions on Software Engineering, Vol. 30, Nº 3, pp. 193- 206, March 2004,.
17. France R., Ghosh S., Song E., Kim D.: A Metamodeling Approach to Pattern-Based Model Refactoring, IEEE Software, Special Issue on Model Driven Development, Vol. 20, Nº 5, pp. 52- 58, September-October 2003.
18. Burg, J.F.M., van de Riet, R.P.: Analyzing informal requirements specifications: a first step towards conceptual modeling. *Proceedings on Applications of Natural Language to Information Systems (NLDB'96).* Amsterdam, The Netherlands, IOS Press. 1996.

19. Flield, G., Kop, C., Mayerthaler, W., Mayr H., Winkler C., Weber Ch., Salbrechter A.: Semantic Tagging and Chunk Parsing in Dynamic Modeling. Natural Language Processing and Information Systems. LNCS 3136, pp. 421-426, Springer-Verlag, 2004.
20. Juristo, N., Moreno, A.M., López, M.: How to use linguistic instruments for object-oriented analysis. *IEEE Software*, 17, 3, 80-89 (2000).
21. Feijs, L.M.G.: Natural language and Message Sequence Chart representation of use cases. *Information and Software Technology*, 42, 633-647, 2000.
22. Mencl V.: Converting Textual Use Cases into Behaviour Specifications, Technical Report 2004/05. Charles University, Department of Software Engineering. Czech Republic.

Applying Modular Method Engineering to Validate and Extend the RESCUE Requirements Process

Jolita Ralyté[1], Neil Maiden[2], Colette Rolland[3], and Rébecca Deneckère[3]

[1] CUI, University of Geneva, Rue de Général Dufour, 24, CH-1211 Genève 4, Switzerland
ralyte@cui.unige.ch
[2] Centre for HCI Design, City University, Northampton Square, London EC1V OHB, UK
N.A.M.Maiden@city.ac.uk
[3] CRI, University of Paris 1 – Sorbonne, 90 Rue de Tolbiac, 75013 Paris, France
{rolland, denecker}@univ-paris1.fr

Abstract. Configuring and applying complex requirements processes in organisations remains a challenging problem. This paper reports the application of the Map-driven Modular Method Re-engineering approach (MMMR) to a research-based requirements process called RESCUE. RESCUE had evolved in the light of research findings and client requests. The MMMR approach was applied to model the RESCUE process, identify omissions and weaknesses, and to reason about improvements to RESCUE that are currently being implemented. Results have implications for both the scalability and effectiveness of the MMMR approach and for innovative requirements processes such as RESCUE.

1 Introduction

Establishing the requirements for software-based socio-technical systems remains a challenge for many organisations. One reason for this is the increasing complexity of the processes needed to establish such requirements effectively. Although some robust processes are emerging, such as REVEAL [4], KAOS [16] and RUP [6], we still lack tried-and-tested techniques for manipulating and adapting these requirements processes so that they meet the needs and constraints of client organisations. This paper reports the results of a collaboration between method engineering and requirements engineering researchers to apply one formalism – the MAP formalism [14] – to model and extend the RESCUE requirements process [10].

Our objectives for this work were two-fold. The RESCUE team wanted to validate and extend the RESCUE process and improve its effectiveness in future requirements engineering projects. The authors of the MAP formalism wanted to test the utility of the map-driven method re-engineering (MMMR) approach [12] for verifying, extending, customising and integrating a full-scale requirements process. RESCUE is a complex and multi-disciplinary process that has been used to specify requirements for several air traffic management systems [9; 10]. In spite of these successes the lack of a formal representation of the process led to concerns about the completeness and effectiveness of RESCUE. Therefore the MMMR approach was applied to achieve three goals. Firstly, it was applied to verify the RESCUE process to discover gaps and inconsistencies in the process. In the MMMR approach this was achieved by discovering missing and single strategies for achieving process intentions. Secondly it was used to extend RESCUE by adding new strategies based on reported good practice and academic research for scenario-based requirements processes. Thirdly, the maps were used to enable local customization of RESCUE to meet client process needs and constraints.

The remainder of this paper is in 5 parts. Section 2 describes the MMMR approach. Section 3 describes the RESCUE process. Section 4 describes how we re-engineered RESCUE using MMMR. Section 5 reports how RESCUE was extended using this re-engineering work. The paper ends with a review of this work, and outlines future work.

2 Map-Driven Modular Method Re-engineering (MMMR)

Our approach for method re-engineering uses the MAP formalism [14]. This section briefly introduces this formalism and describes the Map-driven Modular Method Re-engineering (MMMR) approach.

2.1 The Map Formalism

The MAP formalism provides a process representation system based on a non-deterministic ordering of *intentions* and *strategies*. An *intention* I_i is a goal to be achieved by the performance of an activity whereas a *strategy* S_{ij} is an approach, a manner to achieve an intention. Following the Map formalism, several strategies can be provided by the process model to achieve each intention.

Another key element of a map is a triplet $<I_i, I_j, S_{ij}>$ named a *section*. A *section* represents a way to achieve the target intention I_j from the source intention I_i following the strategy S_{ij}. Each section of the map captures the condition to achieve an intention and the specific manner in which the task associated with the target intention can be performed. This manner is called an *Intention Achievement Guideline (IAG).*

The arrangement of the sections in a map forms a labelled directed graph with intentions as nodes and strategies as edges. The directed nature of the graph shows which intentions can follow each other. Two types of progression guidelines, *Intention Selection Guideline (ISG)* and *Strategy Selection Guideline (SSG)*, help to select the next intention and the next section respectively.

The process model represented in the form of a map has a modular structure; each of its IAGs represents a more or less autonomous guideline which can be *simple*, *tactical* or *strategic* with regard to its content, formality, granularity, etc. A *simple guideline* may have an informal content and advise on how to proceed to handle the situation in a narrative form. A *tactical guideline* is a complex guideline, which uses a tree structure to relate its sub-guidelines. This guideline follows the *NATURE* process modelling formalism [5], which proposes two different structures: the *choice* and the *plan*. Each of its sub-guidelines belongs to one of these types of guideline. Finally, the s*trategic guideline* is a complex guideline using the MAP formalism. Therefore, the map allows to represent methods in different levels of abstraction. An IAG associated to one map section can also be represented by a map at a lower level of abstraction.

2.2 The Process Model for Map-Driven Modular Method Re-engineering

We represent the process model of every method as a map with its associated guidelines. As mentioned above, the map structure offers the re-engineered method a high degree of modularity and provides means to evaluate this method, to decompose it into method chunks, to enhance it by adding new strategies to achieve its intentions, etc. As shown in Fig. 1, our MMMR process model is also represented as a map.

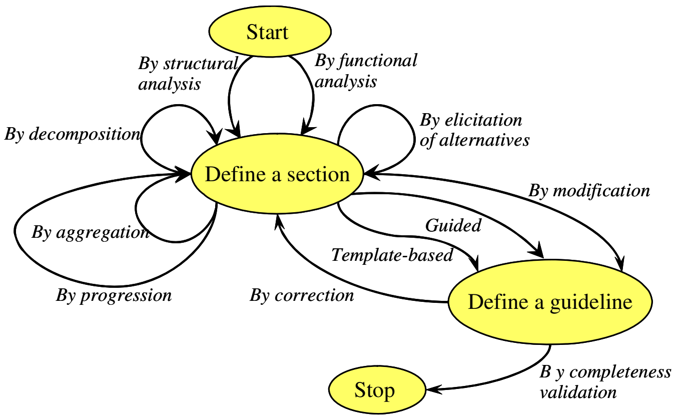


Fig. 1. Process Model for Map-driven Method Re-engineering

According to this map structure, re-engineering the process model of a method requires us first to redefine it in terms of map sections and their guidelines. For this reason, our process seeks to achieve two core intentions: *Define a section* and *Define a guideline* and proposes a set of strategies to satisfy them. For example, there are two strategies *By structural analysis* and *By functional analysis* to achieve the intention *Define a section*. The *Structural analysis* strategy is recommended when the re-engineered method does not provide the method engineer with a formally defined process model but with a simple description of the product to construct. This strategy uses a glossary of generic process intentions to support the discovery of method intentions. On the other hand, the *Functional analysis* strategy should be used if the method has a defined process model that is expressed in the form of steps and recommended actions. This strategy helps to identify the method map sections from these actions, and steps.

When a section is defined, the method engineer can either define the guidelines associated to this section (to progress to the intention *Define a guideline*) or define new sections (to repeat the intention *Define a section*).

The definition of the section guidelines consists of describing the IAG associated with each section, the ISG associated to a set of sections having the same source intention and different target intentions and the SSG associated to every set of parallel sections. The definition of these guidelines is supported by two strategies: the *Template based* strategy and the *Guided* strategy. The former provides a template for every type of guideline and provides advice to experts whereas the latter helps novices by providing more detailed recommendations.

The definition of new sections based on the existing ones (Fig. 1) may be achieved in four different ways, or manners: *By decomposition* of an existing section into several ones, *By aggregation of* a set of sections into a new one, *By elicitation of alternative* sections to a given one, i.e. having an alternative strategy or an alternative source or target intention, and *By progression* strategy which helps to define a new section allowing to progress in the method map from the existing one.

Modifications of the sections (decomposition, aggregation) imply the revision of the associated guidelines if already defined. The *Modification* strategy guides the method engineer to accomplish these transformations. In a similar manner, the process of guidelines definition may imply the transformation of existing sections. For example, the decomposition of an intention achievement guideline could lead to decomposition of the corresponding section. Such transformations can be accomplished following the *Correction* strategy.

The method re-engineering process ends with the *Completeness validation* strategy. This strategy helps to verify if all of the guidelines associated to the map sections have been defined. Due to space limitation we cannot present all of these guidelines. However some of them will be further explained in section 4 when used to re-engineer the RESCUE approach that we introduce in the next section.

3 Introduction to RESCUE

The RESCUE (Requirements Engineering with Scenarios for User-Centred Engineering) process [9] supports a concurrent engineering process in which different modelling and analysis processes take place in parallel. The concurrent processes are structured into 4 streams shown in Fig. 2. Each stream has a unique and specific purpose in the specification of a socio-technical system:

- Human activity modelling provides an understanding of how people work, in order to baseline possible changes to it [17];
- System goal modelling enables the team to model the future system boundaries, actor dependencies and most important system goals [18];
- Use case modelling and scenario-driven walkthroughs enable the team to communicate more effectively with stakeholders and acquire complete, precise and testable requirements from them [15];
- Requirements management enables the team to handle the outcomes of the other 3 streams effectively as well as impose quality checks on all aspects of the requirements document [13].

Sub-processes during these 4 streams (shown in bubbles in Fig. 2) are co-ordinated using 5 synchronisation stages that provide the project team with different perspectives with which to analyse system boundaries, goals and scenarios. These 4 streams are supplemented with 2 additional processes. Acquiring requirements from stakeholders is guided using ACRE [7], a framework for selecting the right acquisition techniques in different situations.

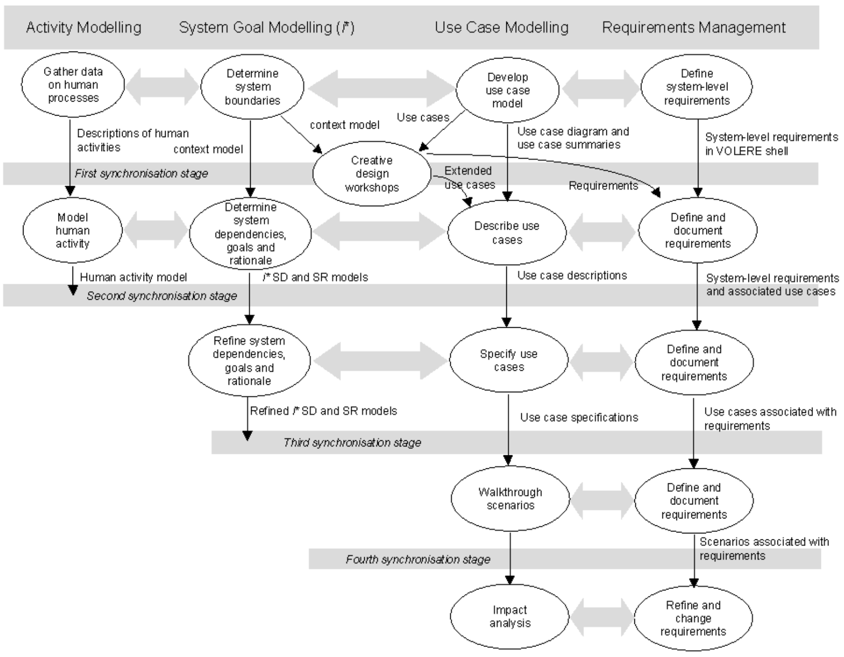

Activity Modelling
System Goal Modelling (i*)
Use Case Modelling
Requirements Management
Gather data on human processes
Determine system boundaries
Develop use case model
Define system-level requirements
Descriptions of human activities
context model
context model
Use cases
Use case diagram and use case summaries
System-level requirements in VOLERE shell
Creative design workshops
First synchronisation stage
Extended use cases
Requirements
Model human activity
Determine system dependencies, goals and rationale
Describe use cases
Define and document requirements
Human activity model
i* SD and SR models
Use case descriptions
System-level requirements and associated use cases
Second synchronisation stage
Refine system dependencies, goals and rationale
Specify use cases
Define and document requirements
Refined i* SD and SR models
Use case specifications
Use cases associated with requirements
Third synchronisation stage
Walkthrough scenarios
Define and document requirements
Scenarios associated with requirements
Fourth synchronisation stage
Impact analysis
Refine and change requirements

Fig. 2. The RESCUE process structure

Creativity workshops normally take place after the first synchronization stage, to discover and surface requirements and design ideas that are essential for *i** system modelling and use case specification during stage 2. Stage 1 inputs to the workshops include the system context model from the system goal modelling stream and use case diagrams from the use case modelling stream, both shown in Fig. 2.

Scenarios walkthroughs to discover more complete requirements take place during stage 4. Scenarios are generated and walked through using ART-SCENE, a web-based scenario environment that was designed using one cognitive principle often exploited during prototyping – that people recognise items, for example scenario events generated by ART-SCENE, better than they recall them from memory [2]. For each generated normal event and alternative course the facilitator guides stakeholders to recognize, discover and document requirements.

Work and deliverables from RESCUE's 4 streams are coordinated at 5 key synchronisation points at the end of the 5 stages shown in Fig. 2, implemented as one or more workshops with deliverables to be signed off by stakeholder representatives:

1. The *boundaries* point, where the team establishes first-cut system boundaries and undertakes creative thinking to investigate these boundaries;
2. The *work allocation* point, where the team allocates functions between actors according to boundaries, and describe interaction and dependencies between these actors;
3. The *generation* point, where required actor goals, tasks and resources are elaborated and modelled, and scenarios are generated;

4. The *coverage* point, where stakeholders have walked through scenarios to discover and express all requirements so that they are testable;
5. The *consequences* point, where stakeholders undertake walkthroughs of the scenarios and system models to explore impacts of implementing the system as specified on its environment.

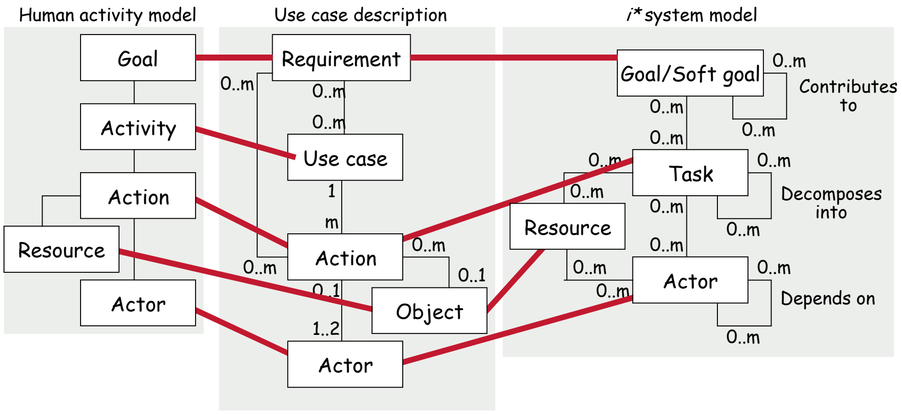


Fig. 3. The RESCUE concept meta-model as a UML class diagram showing mappings between constructs in the 3 model types

The synchronisation checks applied at these 5 points are designed using a RESCUE meta-model of human activity, use case and *i** modelling concepts constructed specifically to design the synchronisation checks. It is shown in simplified form in Fig. 3 – the darker horizontal lines define the baseline concept mappings across the different models used in RESCUE. In simple terms, the meta-model maps actor goals in human activity models to requirements in use case descriptions and *i** goals and soft goals. Likewise, human activities map to use cases, and human actions to use case actions that involve human actors in use cases and tasks undertaken by human actors in *i** models. Human activity resources map to *i** resources and objects manipulated in use case actions, and actors in all 3 types of model are mapped. The complete meta-model is more refined. Types and attributes are applied to constrain possible mappings, for example use case descriptions and *i** models describe system actors, however only human actors in these models can be mapped to actors in human activity models.

RESCUE was originally developed to support the scenario-driven specification of requirements using ART-SCENE [11]. Streams such as use case modelling were developed to provide direct inputs into ART-SCENE's scenario generation tool, and other streams such as activity modelling and goal modelled were added to improve the completeness and correctness of the use case specifications. Other changes were made in response to client requests as the process was rolled out on different projects. At no time was RESCUE re-engineered systematically to improve its completeness, to enable it to be customized to meet the needs of different clients or to support effective integration with other processes with the RUP. Therefore, in the summer of 2004, a collaborative exercise to model and re-engineer RESCUE using MMMR was undertaken.

4 Re-engineering RESCUE

The complexity of RESCUE meant that its process should be represented at different levels of abstraction. The re-engineering activity started by defining the map at the higher level of abstraction, then by detailing the IAG associated with each of its sections as lower level maps.

4.1 Defining First Level RESCUE Map

As RESCUE is process-oriented, we apply the *Functional analysis* strategy (Fig. 1) to re-engineer its process into a map. This strategy recommends to identify first the main method intentions and the strategies proposed by the method to satisfy these intentions, and finally, to order these intentions and strategies in the map.

Defining RESCUE map sections. The guideline associated to the *Functional analysis* strategy recommends analysing the process steps to identify the key product parts that are target products of these steps, and to couple them with some of the generic intentions provided in our method base glossary representing the objective of each step. Therefore, each method step is defined by one or more intentions. As shown in Fig. 2, RESCUE is divided into five main stages. Based on these stages the core intentions of the RESCUE map were identified as follows:

- The objective of the first RESCUE stage is to identify the boundaries of the system under consideration and to approve them. We called this intention *Agree on System Boundaries*. The RESCUE approach uses different models, such as human activity model, context model and use case model, to achieve this objective. As a consequence, we named the strategy that achieves this intention the *Multi-perspective modelling* strategy.
- The second RESCUE stage is called work allocation. It is intended to deliver use case specifications for each actor of the system. We called the corresponding intention *Specify Use Cases*. The achievement of this stage is mainly based on organisation of creativity workshops. Therefore we named the strategy *Creativity workshop driven*.
- The third RESCUE stage results in the automatic scenario generation from the use cases using ART-SCENE. Therefore, the name of the intention is *Generate Scenarios* and the corresponding strategy is called *With ART-SCENE.*
- During the fourth RESCUE stage the stakeholders are invited to walk through the generated scenarios to discover and express requirements so that they are testable. As a result, the main intention of this stage is *Specify Requirements* and the strategy is called *With Scenario Walkthrough*.
- Finally, the fifth RESCUE stage deals with requirement validation by analysing the impact of scenario execution and requirements correction, and new requirements acquisition and specification if necessary. Consequently, we could define two main intentions: (1) *Validate Requirements*, which can be achieved by following the *Impact Scenario Analysis* strategy and (2) *Specify Requirements*, which is achieved by following the *Feedback* strategy.
- The RESCUE process ends by delivering the complete set of requirements specifications. We called this strategy the *Delivery* strategy.

The next step recommended by the guideline consists in ordering the identified intentions and strategies in a process map. For every intention and one associated strategy we have to identify the pre-conditions that should be satisfied in order to reach the intention following this strategy. That is, we need to identify the product necessary to achieve this intention (the required input product) and then to identify which intention produces this product. For example, the achievement of the intention *Specify Use Cases* using the *Creativity workshop driven* strategy requires as input products the models that are obtained during the first process stage, that is by achieving the intention *Agree on System Boundaries*. The intention *Specify Requirements* requires as input product scenarios that are obtained by achieving the intention *Generate Scenarios*. Furthermore, the intention *Specify Requirements* was identified twice (in stages four and five), but it is evident that we put this intention in the map only once. In a similar manner we arranged the identified intentions and strategies in the map and we obtained the first version of the RESCUE map shown in Fig. 4.

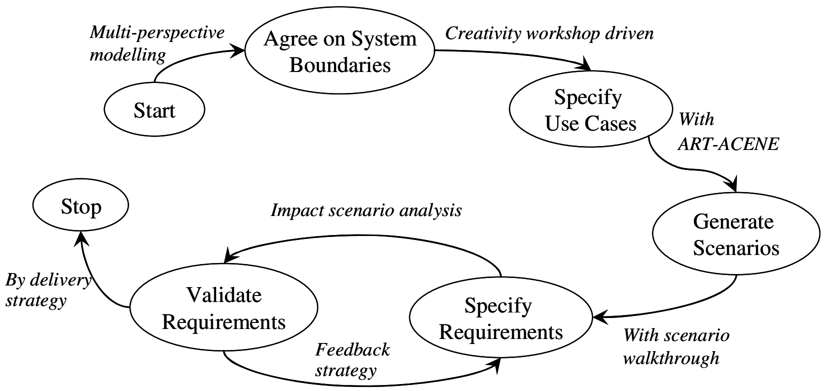


Fig. 4. First version of the RESCUE map

Following Fig. 1, the next step is to refine the obtained map sections by applying different strategies or to define different guidelines associated to this map. Let us refine the map first.

Refining the RESCUE map. Each intention in the RESCUE map should be modelled at the same level of abstraction. The intentions described in the first level RESCUE map represent the main products that are obtained by applying RESCUE. However, the scenarios produced by achieving the intention *Generate Scenarios* are only used as a means to specify requirements in a specification that is the main achievement from RESCUE. Therefore, we merged the sections <*Specify Use Cases, Generate Scenarios, With ART-SCENE*> and <*Generate Scenarios, Specify Requirements, With scenario walkthrough*> by applying the *Aggregation* strategy (Fig. 1) to obtain a new section <*Specify Use Cases, Specify Complete Requirements, With generated scenario walkthrough*>.

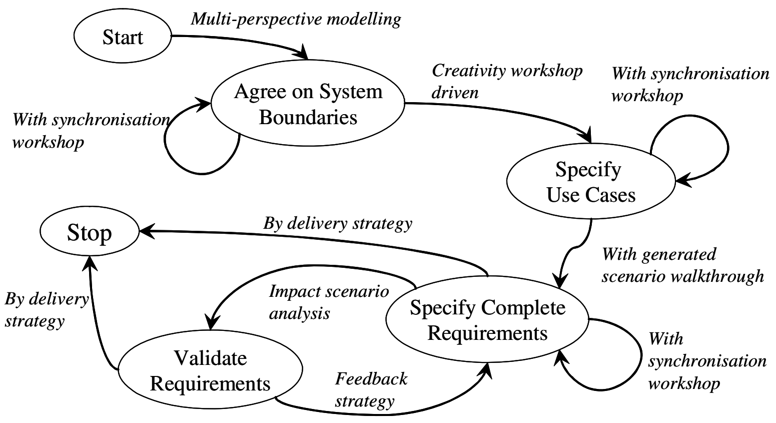

Start
Multi-perspective modelling
Agree on System Boundaries
With synchronisation workshop
Creativity workshop driven
With synchronisation workshop
Specify Use Cases
Stop
By delivery strategy
With generated scenario walkthrough
Impact scenario analysis
Specify Complete Requirements
By delivery strategy
Validate Requirements
Feedback strategy
With synchronisation workshop

Fig. 5. The RESCUE map

The *Progression* strategy (Fig. 1) allowed us to add new sections to the RESCUE map. Each RESCUE stage ends by synchronising the results of that stage. To describe these synchronisations we added three new sections to the RESCUE map: (1) <*Agree on System Boundaries, Agree on System Boundaries, With synchronisation workshop*>, (2) <*Specify Use Cases, Specify Use Cases, With synchronisation workshop*> and (3) < *Specify Complete Requirements, Specify Complete Requirements, With synchronisation workshop*>.

In a similar manner we added a new section <*Specify Complete Requirements, Stop, By delivery strategy* > that ends the RESCUE process after achieving the intention *Specify Complete Requirements*. Fig. 5 depicts the obtained first-level RESCUE map.

4.2 Defining Second Level Maps

Each IAG associated to the RESCUE map can also be defined as a map. Therefore, we re-applied the map definition process as defined in our MMMR (Fig. 1). Because of the lack of space, we do not describe how all of the second level maps were developed. Fig. 6 illustrates the map representing the IAG associated to the section <*Start, Agree on System Boundaries, Multi-perspective modelling strategy*> of the RESCUE map (Fig. 5). According to this map, the requirements engineer has to work with four artefacts – the human activity, context and use case models, and the system requirements documentation, to define system boundaries. The RESCUE approach provides several different ways to achieve the four corresponding intentions. For example, there are two strategies, *With light weight ethnography techniques* and *With DFD techniques*, to *Model Human Activity*. The *Cross checking* strategies allow to validate the correctness and coherence of the obtained models.

Fig. 7 shows another example of the second level map, the IAG associated to the RESCUE map section <*Specify Use Cases, Specify Complete Requirements, With generated scenario walkthrough*>. The RESCUE team generates scenarios and walking through them using ART-SCENE to discover requirements by using different walkthrough techniques. Requirements are documented using the VOLERE shell.

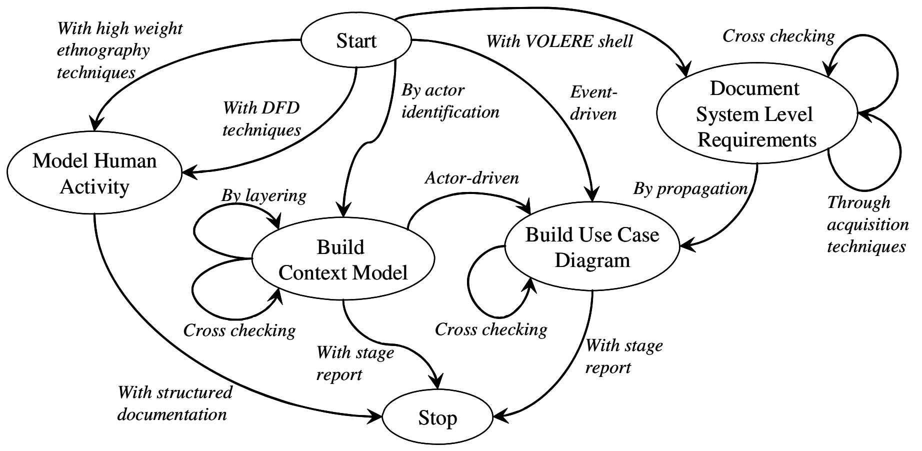


Fig. 6. Second level RESCUE map: the IAG associated to the section <*Start, Agree on System Boundaries, Multi-perspective modelling strategy*>

Whilst the first level map represents a more or less linear process, the second level maps are richer and often provide several strategies to achieve each intention. The progression guidelines are important in the second level maps. Given that, each map describes multiple manners, or ways, to achieve an intention, it needs to provide as much guidance as possible for selecting the right intention for each situation in RESCUE. An SSG provides this guidance for each set of parallel sections, whilst an ISG has to help the selection of the next intention to attain.

The definition of selection arguments plays an important role in strategies selection. In order to better define strategy selection arguments we propose a set of predefined attributes such as time, amount of resources, required domain knowledge, user involvement, difficulty of management, etc., that are specialised according to the nature of the strategies to compare. These attributes allow us to evaluate different aspects of the corresponding strategies and to compare them. Table 1 illustrates the comparison of four strategies allowing to attain the intention *Discover Requirements* from the intention *Produce Agreed Scenario* (Fig. 7).

Table 1. Comparison of four strategies to achieve the intention *Discover Requirements* from the intention *Produce Agreed Scenario*

Strategy selection attributes	Distributed workshop	Individual walkthrough	Facilitated workshop	Mobile walkthrough
Elapsed time to discover requirements/per scenario	5	0.5	0.5	0.3
Amount of analyst resource needed	0	0	2	1
Level of domain knowledge required	High	High	Low	Medium
Level of user involvement needed	High	High	High	Low
Level of management commitment needed	Low	Medium	High	Low
Capability to handle complex systems	Medium	Low	High	Medium
Capability to handle innovative systems	Medium	Low	High	Low
Capability to handle unstable requirements	High	High	High	High
Requirements discovery rate/hour	<8	<8	8-10	8-10
Number of VOLERE attributes discovered	5	5	5	5

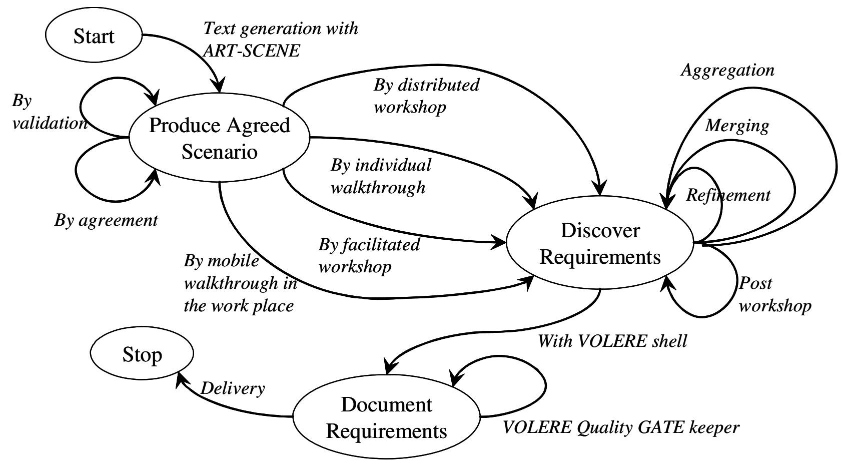


Fig. 7. IAG associated to the section <*Specify Use Cases, Specify Complete Requirements, With generated scenario walkthrough*>

5 Validation and Extension of RESCUE

In order to validate and extend the RESCUE process we explored all its second level maps, including some not shown in this paper. In particular we sought to overcome the weakness of single strategy intentions of RESCUE map and create and develop new strategies to achieve intentions that only have one strategy in the current version

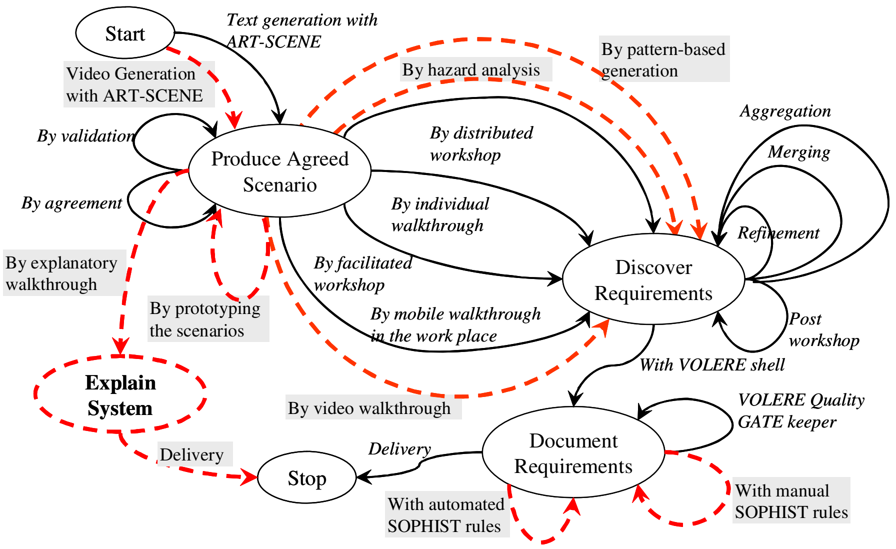

Start
Text generation with ART-SCENE
Video Generation with ART-SCENE
By hazard analysis
By pattern-based generation
By validation
Produce Agreed Scenario
By agreement
By distributed workshop
By individual walkthrough
By facilitated workshop
By explanatory walkthrough
By prototyping the scenarios
By mobile walkthrough in the work place
Aggregation
Merging
Refinement
Discover Requirements
Post workshop
With VOLERE shell
Explain System
By video walkthrough
VOLERE Quality GATE keeper
Delivery
Stop
Delivery
Document Requirements
With automated SOPHIST rules
With manual SOPHIST rules

Fig. 8. The guideline IAG <*(Use case specifications), Specify Complete Requirements with generated scenario walkthrough* > enhanced with new sections

of RESCUE. For example, we considered possible new strategies to enhance the complete requirements specification process captured in the map of Fig. 7.

Fig. 8 shows the new process map for requirements specification with new sections and strategies shown in dashed edges. By systematically reviewing and walking through the process map, we were able to consider each intention in turn, and brainstorm new strategies for each intention. As a consequence, 9 new strategies and one new intention were identified and modelled. The new intention, *Explain System*, was generated and added to the map in response to questions about how the process ended. Not all instances of the process result in documented requirements. Scenarios can also be used as effective communication and explanation devices for a new system, independent of their use to discover requirements. Hence the new intention and an associated new strategy, *By explanatory walkthrough*, added to the process map. An explanatory walkthrough exploits the narrative structure of a scenario to describe and explain the future system's behaviour, and other types of requirement linked to that behaviour. It is an important component of a requirements review or read through activity.

The remainder of this section is two parts. The first outlines new strategies added to the *Complete Requirements Specification* map as a result of the process modelling exercise. The second describes strategy selection arguments in tabular form to demonstrate how to select between these new strategies.

5.1 New Strategies for Specifying Complete Requirements

Two new strategies, *Video Generation with ART-SCENE* and *Discover Requirements, by Video Walkthrough* were designed to produce the agreed scenario and discover requirements. Currently ART-SCENE scenarios are text-based. A text-based use case specification is input to ART-SCENE to generate an interactive and structured scenario that describes normal and alternative course events in text form. However, recent extensions to ART-SCENE to support multi-media representation of scenarios [19] have revealed new opportunities for video-based scenario walkthroughs. Initial trials reveal that multi-media scenario representations provide more cues from which stakeholders can discover and document requirements [19]. Therefore, the use case specification will be extended with a video sequence that describes the normal behaviour of actors to achieve their goals, and use case normal course events are linked to episodes, such as *an air traffic controller communicating with a pilot*, in the digital video. The ART-SCENE algorithm will still be used to generate alternative courses for each normal course event that are now linked directly to one or more digital video episodes, thus producing an agreed scenario in a video form. To discover requirements, an enhanced version of ART-SCENE will enable stakeholders to control and play a digital video of the normal course behaviour. Then, during the playing of the video, stakeholders are prompted with alternative course questions in text form, such as what if the pilot misunderstands the air traffic controller, in response to which they can document new requirements using existing ART-SCENE functions. We hypothesise that richer scenario representations will lead to more complete requirements discovery.

The brainstorming session also surfaced 3 other strategies for discovering requirements. One strategy, *By pattern-based generation*, recalled earlier research undertaken by the RESCUE team that is not currently implemented in ART-SCENE. Alexander's original ideas of a pattern [1] focus on the interactions between the physical form of the built environment and how this form inhibits or facilitates various sorts of individual and social behaviour in it. The emphasis is on the characteristics of the environment that might facilitate or inhibit action. A pattern captures the essentials of a 'good design' that maximises characteristics that facilitate desirable actions over those that inhibit these actions. Applied to socio-technical system design with ART-SCENE, a pattern must capture the essential elements of the software system (expressed as functional and non-functional requirements), and how the form of this system facilitates and inhibits desirable individual or social behaviour (expressed using the scenario). It captures good designs that have been shown to facilitate desirable behaviour expressed in the scenario [8].

Based on the discovery of this strategy, we will extend ART-SCENE with 2 types of pattern that guide the discovery and documentation of system requirements. Firstly, we will develop and implement patterns that describe classes of solutions, expressed as generic requirement statements, to classes of abnormal behaviour and state in the environment, expressed as alternative courses that instantiate these classes. Implementation of these patterns in ART-SCENE is tractable because scenario alternative courses are generated automatically using classes of abnormal behaviour and state. During a scenario walkthrough, the pattern is applied to recommend generic requirements statements that describe what a system shall do avoid or mitigate against the effects of a selected alternative course [15]. For example, *an expected event not occurring* can be handled by the system in different manners – *by re-requesting the event*, *by undertaking some default action*, or *by assuming that the event has taken place*.

Secondly, we will develop and implement socio-technical system design patterns that link sequences of events and actions that describe desirable future system use in the environment, expressed as scenario normal courses, to system requirements facilitate the desirable and inhibit undesirable behaviour. Consider the *collect-first-objective-last* pattern reported in [8]. A person who interacts with a system using a personal item should not leave the personal item behind. One design to achieve this is to make the user reclaim the item before achieving their goal. This design can be found in ATMs, metro barriers and secure access systems, and can be specified computationally as a pattern to match in a scenario normal course. Again, we hypothesise that implementing these 2 strategies for *By pattern-based generation* will lead to the discovery of more complete and correct requirements.

Another discovered strategy for discovering requirements from an agreed scenario was *By hazard analysis*. In simple terms, hazard analysis applies simple techniques, such as checklists, to discover hazards associated with a new system. ART-SCENE's automatic generation of scenario alternative courses can also identify potential hazards associated with a specified system. To implement a full hazard analysis strategy within ART-SCENE we will extend its model of abnormal behaviour and state to include a more complete set of hazard classes, then introduce generation settings that will allow a requirements engineer to generate scenarios that are tailored for more rigorous hazard analysis.

Finally, we introduced two new strategies based on techniques from the SOPHIST group with which to document requirements. Goetz & Rupp [3] report 25 authoring rules from psychotherapy that assist in the analysis and quality assurance of requirements expressed in text form. Examples of these rules include *(6) Clarify the modal operators of imperative (e.g. the use of should, shall, must etc)* and *(12) Question nouns without references (e.g. reference to all users, or just certain user groups or individuals).* In RESCUE we can supplement its use of the VOLERE requirements shell [13] with manual and automatic application of these 25 rules. The manual strategy is now implemented through engineer training and guidelines in ART-SCENE that advice on how to describe textual requirements. The automatic strategy will be implemented using a new tool that will parse and invite re-writes of the entered requirements specification to check each requirement against each of the 25 requirements authoring rules. Again, we hypothesise that these 2 strategies will result in more correct and consistent documentation of requirements.

5.2 Strategy Selection

Adding new strategies enriches RESCUE but also makes it more difficult to implement. Additional selection guidelines are needed to combine and/or select between strategies to achieve one intention. To guide selection we have developed new strategy comparison tables that define the predicted cost and benefit of adopting one strategy over another according to strategy selection attributes. Table 2 compares 3 of the defined strategies for achieving the intention *Discover Requirements* from the intention *Product Agreed Scenario*.

Table 2. Comparison of three new strategies to achieve the intention *Discover Requirements* from the intention *Produce Agreed Scenario*

Strategy selection attributes	Pattern-based generation	Hazard analysis	Video walkthrough
Elapsed time to discover requirements/per scenario	0	0.5	0.5
Amount of analyst resource needed	0 (min)	2	0
Level of domain knowledge required	Low	High	Medium
Level of user involvement needed	None	Low	High
Level of management commitment needed	Low	Medium	High
Capability to handle complex systems	Low	Medium	Medium
Capability to handle innovative systems	Low	Low	Low
Capability to handle dependencies on other system (or inter-system dependencies)	Low	Low	N/A
Capability to handle unstable requirements	Low	N/A	High
Requirements discovery rate/hour	Unknown	N/A	<12
Number of VOLERE attributes discovered/requirement	2	N/A	5
Use case action specification rate/hour	High		Low

6 Conclusion

This paper reports a research-driven investigation of the MMMR approach to re-engineer the RESCUE requirements process. Findings were relevant for RESCUE and MMMR. Development of the process models revealed important omissions and single strategy intentions in RESCUE that we resolved by adding new intentions and strategies to the process models. This led us to re-investigate existing literature about scenario-driven requirements processes, and to undertake cost-benefit analyses of RESCUE strategies that we will investigate through future RESCUE rollouts.

Existing process representations of RESCUE did not afford such analysis. The MMMR process maps also gave the authors confidence that changes to RESCUE were consistent with the existing process. The result was an agenda of improvements to RESCUE and its software tools that we are currently implementing.

The paper also demonstrates the effectiveness of MMMR for modelling large-scale requirements processes. Modelling intentions and strategies, rather than processes and artefacts was tractable and cost-effective whilst still allowing the discovery of missing or weak elements of the process. Moreover, thanks to the MAP formalism the RESCUE process was transformed into a modular method: each RESCUE map section represents a more or less autonomous process module. These modules can be combined in different manners and reused in the construction of situation-specific requirements engineering processes in order to meet the needs of client organisations.

The next stage of our collaboration will model RUP's requirement processes [6] as a basis for integrating RESCUE into RUP. Once RUP process maps have been developed, we will merge intentions shared by RUP and RESCUE, add RESCUE intentions to RUP process maps, and introduce RESCUE strategies for achieving these shared intentions.

References

1. Alexander, C. (1979), 'The Timeless Way of Building', New York: Oxford University Press.
2. Baddeley, A.D. (1990), 'Human memory: Theory and practice', Lawrence Erlbaum Associates, Hove.
3. Goetz, R. & Rupp, C. (2003), 'Psychotherapy for Systems Requirements', Proceedings 2nd IEEE International Conference on Cognitive Informatics, IEEE CS Press, p. 75-80.
4. Hammond, J., Rawlings, R. & Hall, A. (2001), 'Will It Work?', Proceedings 5th IEEE International Symposium Requirements Engineering, IEEE CS Press, p. 102-109.
5. Jarke, M., Rolland, C., Sutcliffe, A. & Domges, R. (1999), 'The NATURE requirements Engineering', Shaker Verlag, Aachen.
6. Leffingwell, D. & Widrig, D. (2000), 'Managing Software Requirements: A Unified Approach', Addison-Wesley-Longman.
7. Maiden, N.A.M. & Rugg, G. (1996), 'ACRE: Selecting Methods For Requirements Acquisition', Software Engineering Journal 11(3), p. 183-192.
8. Maiden, N.A.M., Cisse, M., Perez, H. & Manuel, D. (1998), 'CREWS Validation Frames: Patterns for Validating System Requirements'', Proceedings REFSQ98 Workshop.
9. Maiden, N.A.M., Jones, S.V. & Flynn M. (2003), ''Innovative Requirements Engineering Applied to ATM', Proceedings ATM (Air Traffic Management), Budapest, June 23-27.
10. Maiden, N.A.M., Jones, S.V., Manning, S., Greenwood, J. & Renou, L. (2004), 'Model-Driven Requirements Engineering: Synchronising Models in an Air Traffic Management Case Study', Proceedings CAISE'04, Springer-Verlag LNCS 3084, p. 368-383.
11. Mavin, A. & Maiden, N.A.M. (2003), 'Determining Socio-Technical Systems Requirements: Experiences with Generating and Walking Through Scenarios'', Proceedings 11th International Conference on Requirements Engineering, IEEE CS Press, p. 213-222.

12. Ralyté, J. & Rolland, C. (2001), 'An Approach for Method Re-engineering'. Proceedings of the 20th International Conference on Conceptual Modeling (ER2001), LNCS 2224, Springer, p. 471-484.
13. Robertson, S. & Robertson, J. (1999), 'Mastering the Requirements Process', Addison-Wesley-Longman.
14. Rolland, C., Prakash, N. & Benjamen, A. (1999), 'A multi-model view of process modelling'. Requirements Engineering Journal, p. 169-187.
15. Sutcliffe, A.G., Maiden, N.A.M., Minocha, S. & Manuel, D. (1998), 'Supporting Scenario-Based Requirements Engineering', IEEE Transactions on Software Engineering, 24(12), p. 1072-1088.
16. van Lamsweerde, A. (2004), 'Goal-Oriented Requirements Engineering: A Roundtrip from Research to Practice', Proceedings 12th IEEE International Conference on Requirements Engineering, IEEE CS Press, p. 4-7.
17. Vicente, K. (1999), 'Cognitive work analysis', Lawrence Erlbaum Associates.
18. Yu, E. & Mylopoulos, J.M. (1994), 'Understanding "Why" in Software Process Modelling, Analysis and Design', Proceedings, 16th International Conference on Software Engineering, IEEE CS Press, p. 159-168.
19. Zachos, K. & Maiden, N.A.M. (2004), 'ART-SCENE: Enhancing Scenario Walkthroughs with Multi-Media Scenarios', Proceedings 12th IEEE International Conference on Requirements Engineering, IEEE CS Press, p. 360-361.

Security Patterns Meet Agent Oriented Software Engineering: A Complementary Solution for Developing Secure Information Systems

Haralambos Mouratidis[1], Michael Weiss[2], and Paolo Giorgini[3]

[1] School of Computing and Technology, University of East London, England
h.mouratidis@uel.ac.uk
[2] Dept. of Computer Science, Carleton University, Ottawa, Canada
weiss@scs.carleton.ca
[3] Dept. of Information and Communication Technology, University of Trento, Italy
paolo.giorgini@dit.unitn.it

Abstract. Agent Oriented Software Engineering and security patterns have been proposed as suitable paradigms for the development of secure information systems. However, so far, the proposed solutions are focused on one of these paradigms. In this paper we propose an agent oriented security pattern language and we discuss how it can be used together with the Tropos methodology to develop secure information systems. We also present a formalisation of our pattern language using Formal Tropos. This allows us to gain a deeper understanding of the patterns and their relationships, and thus to assess the completeness of the language.

1 Introduction

Information systems security is definitely not a new topic, since its history starts in the sixties [12]. Nevertheless, only recently more importance has been given to information security, and it is considered now one of the main issues during information systems development. This situation is the result of two main factors: (1) the wide usage of information systems by institutions, companies and individuals, and, therefore, the storage of important information; and (2) the increasing number of information systems security criminals such as hackers and attackers. Research on the security of information systems has mainly focused on the definition of security protocols, security mechanisms and other technical solutions. Yet, it has been widely argued over the last few years that security is not simply a technical issue, and that security solutions cannot be blindly inserted into information systems, but security considerations need to be tightly integrated with the development of information systems [14,4,10].

Following this argument, two software engineering paradigms, namely agent oriented software engineering and security patterns, have been proposed (e.g., in [8] and [13]) as promising paradigms for the development of secure information systems. On the one hand, it has been argued [10] that agent oriented software engineering is one of the most natural ways of characterising security issues in information systems, since characteristics, such as autonomy, intentionality and sociality, provided by the use of agent orientation, allow developers first to model the security requirements in high-level, and then incrementally transform these requirements to security mechanisms. On the other hand, security patterns capture design experience and proven solutions to security-related problems in such a way that can be applied by non-security experts [13]. In addition, security patterns introduce several layers of abstraction and thus help to close the gap between security specialists and software engineers.

So far, the solutions proposed by these two paradigms are divided; that is, they only consider either an agent oriented or a security pattern solution. We believe that the integration of agent oriented software engineering and security patterns represents an effective solution for the consideration of security issues during the development stages of information systems. This is mainly due to the appropriateness of an agent oriented philosophy for dealing with the security issues that exist in a computer system and the appropriateness of patterns to transfer security related knowledge to non-security specialists.

Secure Tropos [10] extends the agent oriented software engineering methodology Tropos [3] by providing a set of security-related concepts and processes to allow developers to consider security issues throughout the development stages. Secure Tropos supports three development stages: the early requirements analysis stage, in which the social issues related to the security of the system are identified and analysed; the late requirements analysis stage, in which the technical issues related to the security of the system are identified and analysed; and the architectural design stage, in which the architectural style of the system is defined with respect to the system's security requirements and the requirements are transformed into a design. However the latter could be a very difficult task, especially for a developer without knowledge of security, possibly resulting in the development of a non-secure system. For this reason, we propose to complement secure Tropos with the use of security patterns. Security patterns capture existing proven experience about how to deal with security problems during the software development and they help to promote best design practices.

Building on our previous work [9,15] we introduce an approach for modelling security issues in information systems using agent-oriented software engineering and security patterns. Section 2 introduces the proposed security pattern language. Section 3 discusses the formalisation of the pattern language. Section 4 describes how it can be applied, and Section 5 concludes this paper.

2 Security Pattern Language

Patterns by themselves are only point solutions, and they are usually organised into pattern languages. A *pattern language* is a set of closely related patterns that guide the developer through the process of designing a system [1]. As the patterns from a pattern language are applied, each pattern suggests new patterns to be applied that further refine the design, until no more patterns can be applied.

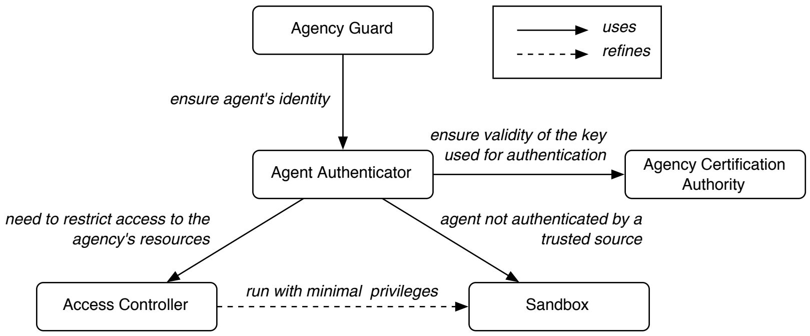

Agency Guard
uses
refines
ensure agent's identity
ensure validity of the key used for authentication
Agent Authenticator
Agency Certification Authority
need to restrict access to the agency's resources
agent not authenticated by a trusted source
Access Controller
run with minimal privileges
Sandbox

Fig. 1. Roadmap of the security pattern language

Since our aim is to integrate agent oriented software engineering and security patterns, the pattern language should employ agent-oriented concepts, such as intentionality, autonomy, sociality, and identity. Therefore, the structure of a pattern should be described not only in terms of collaborations and the message exchange between the agents, but also in terms of their social dependencies and intentional attributes, such as goals and tasks. This allows for a complete understanding of the pattern's social and intentional dimensions.

We use the so-called Alexandrian format for organising each pattern [1]. The sections of a pattern are *context*, *problem*, *solution*, and *consequences*. Brief descriptions of the problem and solution are put in boldface, followed by more detailed discussions. The consequences are organised into *benefits*, *liabilities*, and *related patterns*. Figure 1 provides a roadmap of our pattern language. The directed links show dependencies between patterns, and point from a pattern to the patterns that developers may want to consult next. It should also be stressed that the patterns of the language have been identified from real implementations of agent-based systems, and their initial versions have been workshopped at a patterns conference [9] in order to validate and improve them.

2.1 Agency Guard

... a number of agencies exist in a network. Agents from different agencies must communicate with each other, or exchange information. This involves the movement of some agents from one agency to another, or requests from agents belonging to one agency for resources belonging to another agency.

***

A malicious agent that gains unauthorised access to the agency can disclose, alter or generally destroy data residing in the agency.

Many malicious agents will try to gain access to agencies that they are not allowed to access. Depending on the level of access the malicious agent gains, it might be able to shut down the agency, or exhaust the agency's computational resources, and thus deny services to authorised agents. The problem becomes the more severe the more backdoors there are to an agency, enabling potential

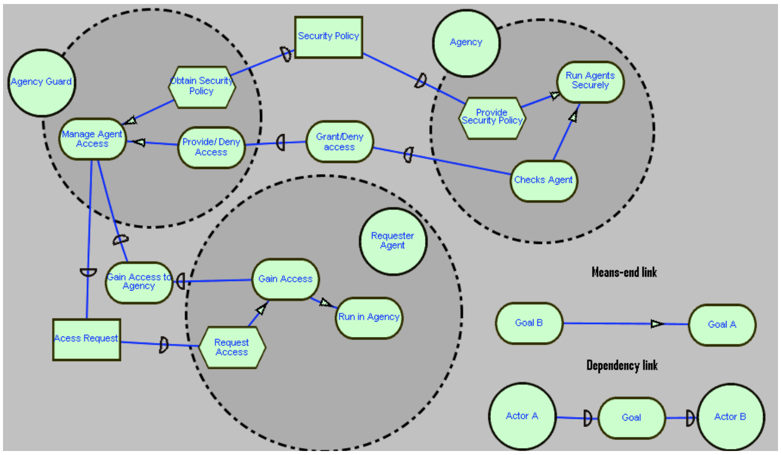


Fig. 2. Structure of the *Agency Guard* pattern

malicious agents to attack the agency from many places. On the other hand, not all agents trying to gain access to the agency must be treated as malicious, but rather access should be granted based on the security policy of the agency.

Therefore:

Ensure that there is only a single point of access to the agency.

When a Requester Agent wishes to gain access to an Agency (either to access resources or move to this agency) its requests must be directed to an Agency Guard, which grants or denies access requests according to the agency's security policy. The Agency Guard is the only point of access to the Agency, and cannot be bypassed, meaning all access requests must go through it. In traditional terms, the concept of an Agency Guard is referred to as a monitor [2].

The structure of the pattern in terms of the actors involved and their social dependencies is shown in Figure 2 using the Tropos notation. Each circle represents an actor, and each dependency link between two actors indicates that one actor depends on the other for some goal (oval), task (hexagon) or resource (rectangle) to be achieved. Moreover, actors are analysed internally (internal analysis is indicated within the dashed line circles) with the aid of means-end links, which are used to indicate (alternative) means (goals/tasks) for reaching a goal. For example, the Agency depends on the Agency Guard to Grant/Deny Access to the Agency according to the Agency's security policy.

***

Benefits:

– It is easier to secure a single point of access, rather than many backdoors.
– Only the Agency Guard needs to be aware of the security policy, and it is the only entity that must be notified if the security policy changes.
– Being the single point of access, only the Agency Guard must be tested for correct enforcement of the agency's security policy.

Liabilities:

- Requester Agents need to present all their credentials (including identity), although they may only be required for some operations on the agency.
- A malicious Requester Agent may masquerade its identity.
- A single point of access to the agency can degrade the performance of the agency (that is, its response time for handling access requests).
- The Agency Guard is a single point of failure. If the Agency Guard fails, the security of the agency as a whole is at risk.
- We cannot prevent Requester Agents from attempting to circumvent the Agency Guard. We, therefore, also need to log access requests.

Related patterns:

- *Agent Authenticator* – ensures the identity of the Requester Agent.

2.2 Agent Authenticator

... you are using *Agency Guard* to protect access to an agency or its resources. To be allowed access, agents must be authenticated, that is, they must provide information about the identity of their owners.

***

Many malicious agents will try to masquerade their identity when requesting access to an agency.

If such an agent is granted access to the agency, it might try to breach the agency's security. In addition, even if the malicious agent fails to cause problems in the security of the agency, the agency under attack will no longer trust the agent impersonated by the malicious agent.

Therefore:

Authenticate agents as they enter the agency.

Requester Agents must be authenticated by the Agency. By authenticating the agent, the Agency Guard ensures the agent comes from an owner trusted by the Agency. Each Requester Agent's owner and Agency have a pair of public/private keys. The Agent Authenticator can authenticate the Requester Agent in two ways: the agent can be digitally signed with its owner's private key, or with the private key of the Agency in which the agent resides. In order for the second approach to work, mutual trust must be established between the sending and receiving agencies (each Agency can be set up so it has a list of "trusted" agencies). If the Agent Authenticator does not trust the Agency from which the agent originates, it can reject the agent, or accept it with minimal execution privileges.

The structure of the pattern is shown in Figure 3.

***

Benefits:

- Since authentication concerns are dealt with in a single location, it is not necessary to provide each agent with its own authentication mechanism.

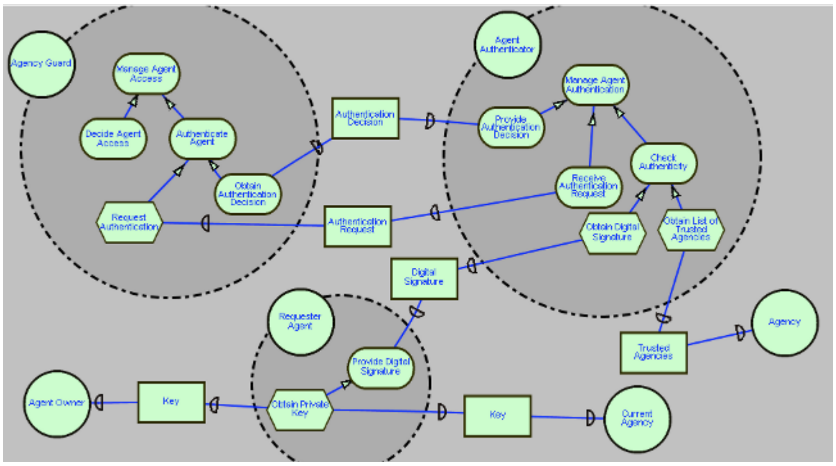


Fig. 3. Structure of the *Agent Authenticator* pattern

– The use of an Agent Authenticator ensures that Requester Agents are authenticated, before they can request a resource from the agency.
– When implementing the system, only the Agent Authenticator must be verified for correct enforcement of the agency's authentication policies.

Liabilities:

– The Agent Authenticator is a single point of failure. If it fails, the security of the agency as a whole is at risk.
– The public key used to authenticate the Requester Agent may be invalid.
– This solution may be too restrictive, as it prevents agents that provide services that the agency cannot provide itself, but cannot be authenticated, from executing.

Related patterns:

– *Access Controller* – restricts access to the agency's resources.
– *Access Certification Authority* – ensures validity of the public key used to authenticate the Requester Agent.
– *Sandbox* – allows running an unauthenticated agent with minimal privileges.

2.3 Sandbox

... you are using *Agent Authenticator* to ensure the requester agent's identity, but the requester agent cannot be properly authenticated. This can be the case either when the agent could not be authenticated, or if it has been authenticated by an agency that the receiving agency does not trust.

***

An agency is most likely exposed to a large number of malicious agents that will try to gain unauthorized access to it.

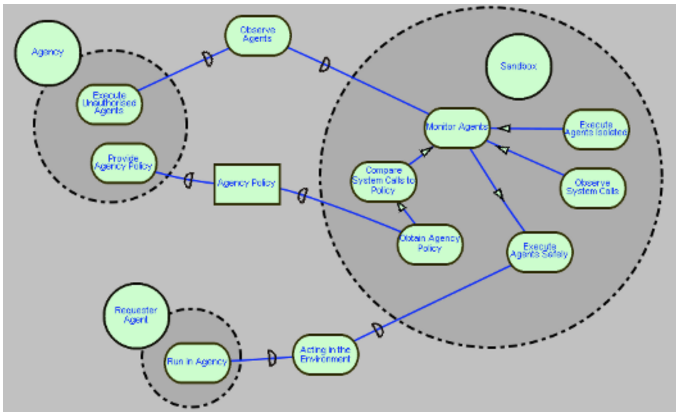


Fig. 4. Structure of the *Sandbox* pattern

Although the agency will try to prevent access to those agents, it is possible that some of them might be able to gain access to the agency's resources. Thus, it is necessary for the agency to operate in a manner that will minimise the damage which can be caused by unauthorized agents gaining access. In addition, some unauthorized agents might be allowed access by the agency in order to provide services the agency's agents cannot provide. Thus, the agency must be cautious to accept such unauthorized agents without putting its security at risk.

Therefore:

Execute the agent in an isolated environment that has full control over the agent's ingoing and outgoing messages.

This solution prevents malicious agents from performing unauthorised operations. The agent is allowed to destroy anything within a restricted environment (a Sandbox), but cannot touch anything outside. The concept is similar to the Java security model, and the chroot environment in Unix. The Sandbox observes all system calls made by the agent, and compares them to the agency's policy. If any violations occur, the Agency can shut down the suspicious agent.

The structure of the pattern is shown in Figure 4.

***

Benefits:

– Agents not authorised but, nonetheless, valuable for the agency can be executed without compromising its security.
– The agency can identify possible attacks (by observing the actions of the agents in the Sandbox), and prevent them from occurring.

Liabilities:

– Some computational resources of the agency might be diverted to non-useful actions, if non-useful agents are sandboxed.
– The use of a Sandbox introduces an extra layer of complexity.

No related patterns.

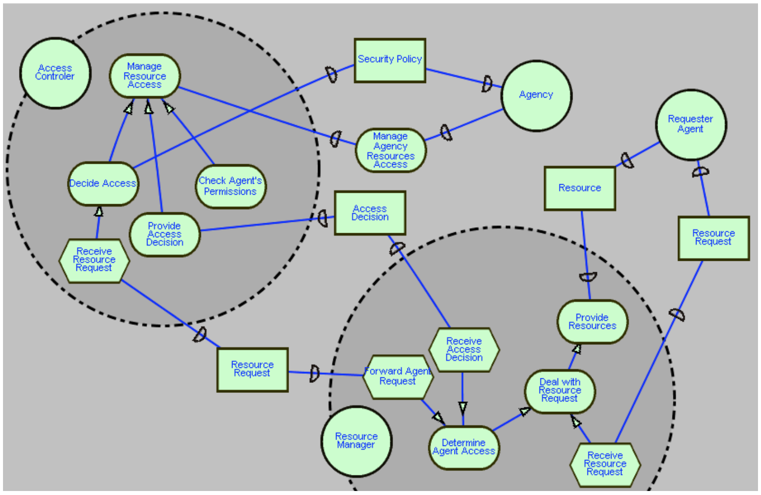


Fig. 5. Structure of the *Access Controller* pattern

2.4 Access Controller

. . . you are using *Agent Authenticator* to ensure the requester agent's identity. Now you need to restrict access to the agency's resources. Many different agents can exist in an agency, which require access to the agency's resources in order to achieve their operational goals. However, they should be able to access only specific resources.

***

Agents belonging to an agency might try to access resources that they are not allowed to access.

Allowing this to happen might lead to serious problems such as the disclosure of private information, or the alteration of sensitive data. In addition, different security privileges will be applied to different agents. The agency should take into account its security policy, and consider each access request individually.

Therefore:

Intercept all requests for the agency's resources.

The agency uses an Access Controller to restrict access to each of its resources. When a Requester Agent requests access to a resource, the request is directed to the Access Controller, which then checks the security policy and decides whether the access request should be approved or rejected. The access decision is then forwarded to the corresponding Resource Manager.

The structure of the pattern is shown in Figure 5.

***

Benefits:

– The agency's resources are used only by agents allowed to access them.
– Different policies can be used for accessing different resources.

Liabilities:

– There is a single point of attack. If the Access Controller is compromised, the system's access control system fails.

No related patterns.

2.5 Agent Certification Authority

... you are using *Agent Authenticator* to authenticate agents coming to the agency. Each agent has a certificate, which contains a public key. The agent proves its identity by signing with its private key, which might come from the agent's owner or the agency that it currently resides. However, such a signature is only valid, if the corresponding public key has been certified.

∗∗∗

Malicious agents wishing to access the agency might try to authenticate using invalid keys or keys that are not certified.

The agency should not authenticate such agents since they might endanger the security of the agency.

Therefore:

Authenticate agents only if their public key has been certified by a trusted certificate authority.

An Agent Certification Authority is used to certify the agent's public key. The Requester Agent sends a certification request to the Agent Certification Authority. When the Agent Certification Authority receives the request, it verifies its validity by checking when the request was generated, that the signed message can be verified using the Requester Agent's public key, and that the public key of the agent is not already in use. If the request is valid, the Agent Certification Authority generates a signed certificate that binds the public key to the agent. Before actually sending the certificate to the agent, the Agent Certification Authority creates an entry in its certification database. When the agent receives the certificate, it can prove its identity by signing with its private key.

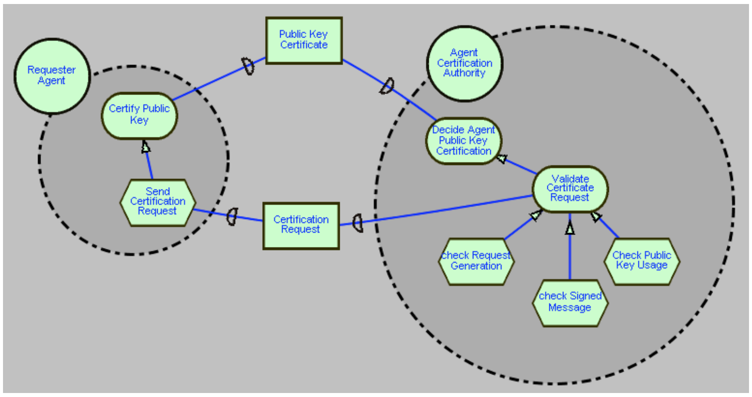


Fig. 6. Structure of the *Agent Certification Authority* pattern

The structure of the pattern is shown in Figure 6.

***

Benefits:

- The validity of the Requester Agent's public key is verified
- The Requester Agent's claim that a public key belongs to its owner or to its originating agency is verified.

Liabilities:

- Scalability is limited when too many Requester Agents try to obtain a verification of their public keys at once. This can be solved by having more than one Agent Certification Authority. Different Agent Certification Authorities are aware of each other and can route certificates between them if necessary.

No related patterns.

3 Formalising the Pattern Language

An important consideration in the development of a pattern language is to assess its completeness. For this reason, in addition to the graphical representation, we have developed a formalisation of the language. Patterns are formalized in terms of the problems they address, and the solutions they offer. We consider a pattern language to be *complete*, if the solutions proposed by the patterns contained in the language address all the problems raised. It is important to note that completeness can only be assessed with regard to a stated set of problems. As new security problems are identified, the pattern language needs to be extended.

As the roadmap in Figure 1 illustrates, the patterns in a pattern language are interconnected. The links indicate uses and refines relationships as defined in [11]. Intuitively, uses can be interpreted in either one of two ways. Either a particular problem is not addressed by a pattern, and the problem has to be resolved by another pattern in the pattern language; or the application of a pattern raises new problems that must be addressed by further patterns in the pattern language. The basic idea underlying our formalization is to follow the uses links between the patterns, and to record the problems addressed by each pattern, as well as the new problems they raise. From this we can either conclude that the application of our patterns helps establish security (that is, that all security problems raised are resolved), or that we need to add more patterns to our language in order to resolve the open problems. For a formal definition of the uses relationship, and a proof that this approach allows us to show the completeness of a pattern language see [13].

Our formalization of patterns is only a partial formalization. A full formalization may not even be desirable, since patterns are meant to be human-readable artifacts, and applying a pattern often requires adapting the pattern to the specific needs of a given context [1]. We, therefore, focus on the following sections of a pattern: problem (addressed by this pattern), solution (how the problem is addressed), and consequences (new problems raised). The solution is included in the formalization, since the application of a pattern results in elements being added to the model, which other problem statements may refer to.

We use Formal Tropos (FT) [7] to describe problems and solutions. A FT specification describes the relevant elements (actors, goals, dependencies, etc.) of a domain and their relationships. The description of each of the elements is structured into an outer and an inner layer. The outer layer is similar to a class declaration. It associates a set of attributes with each element that define its structure. There is also a set of predefined special attributes such as **depender** and **dependee**. The inner layer expresses constraints on the lifetime of the objects, given in a typed first-order linear-time temporal logic.

In passing, it should be noted that a solution that establishes security does not necessarily imply that it is the best solution in terms of other system qualities. Not included in our formalization are non-security softgoals such as complexity. The contributions to non-security softgoals could be used to compare alternative selections of patterns in terms of the quality of the overall solution (i.e., the combined result of applying the patterns). We will incorporate the formalization of non-security softgoals in our future work.

For reasons of space, we present only the formalization of the patterns related to authentication and their relationships. However, the same principles can be used to formalise the rest of the patterns. We present the problem addressed, solution, and new problems introduced by each pattern. The formalization of problems appear where they are first raised, and are referenced in later patterns. Such an approach also proved helpful in ensuring that the description of problems does not make use of any of the new model elements introduced by the solution, but which were not part of the model before the pattern was applied.

To represent problems and solutions in FT we express them using global assertions. These assertions are first-order predicate expressions over model components. Problems are thus statements about the current model. If the assertion holds true, the pattern is applicable. After applying the pattern, the solution statement is asserted, possibly introducing new model elements, and the assertion for the problem no longer holds. The new problems raised by a pattern are assertions that enable the application of further patterns, until no more new problems are raised. The three patterns whose formalization we will present are: *Agency Guard*, *Agent Authenticator*, and *Agent Certification Authority.*

3.1 Agency Guard

This pattern states that RequesterAgents can access the agency from multiple places via the GainAccessToAgency goal dependency. The formalization of the problem (P1) specifies that a RequesterAgent can gain access to the agency by exploiting multiple GainAccessToAgency dependencies in which it participates. Solution S1 resolves this problem, as specified in the last clause of the assertion. Problem P2 also introduces the notion of the owner of a RequesterAgent. In essence, the formalization of P2 indicates that just ensuring that agents can only access the agency through a single point does not ensure that the agents are who they claim to be. This problem needs to be addressed separately.

Problem: */* P1: A malicious agent can gain unauthorised access to the agency from multiple places, not all of which provide the same level of security. */*

$\exists$ ra : RequesterAgent ($\exists$ ga1, ga2 : GainAccessToAgency (ga1.**depender** = ra $\wedge$
ga2.**depender** = ra $\wedge$ ga1.**dependee** $\neq$ ga2.**dependee**))

Solution: */* S1: Ensure that there is only a single point of access to the agency. */*

$\forall$ ra : RequesterAgent ($\forall$ ga1, ga2 : GainAccessToAgency (ga1.**depender** = ra $\wedge$
ga2.**depender** = ra $\rightarrow$ ga1.**dependee** = ga2.**dependee**))

Consequences: */* P2: Agents can enter the agency by posing as another agent. */*

$\forall$ ar : AccessRequest ($\exists$ ra : RequesterAgent (ar.**dependee** = ra $\wedge$
ar.**dependee**.owner $\neq$ ra.owner))

3.2 Agent Authenticator

The pattern resolves problem P2. Solution S2 states that RequesterAgents signed with the private keys of their owners (the DigitalSignature) can be authenticated via the corresponding public keys. Thus, they can no longer masquerade as another agent. However, this solution hinges on the fact that the agency knows the valid public key of the RequesterAgent's owner. But this is generally not the case, as described by problem P3. In fact, a malicious agent may claim that its owner is ra.owner = ao1, whereas, it is ar.**dependee**.owner = ao2. The formalization introduces two new attributes: the key attribute of a RequesterAgent, and the privateKey attribute to be associated with RequesterAgent owners.

Problem: */* P2: Agents can enter the agency by posing as another agent. */*

Solution: */* S2: Agents must prove their identity. Agents are authenticated via their own or their originating agency's public keys. */*

```
∀ ar : AccessRequest (∀ ra : RequesterAgent (ar.dependee = ra ∧
    ∀ ao : AgentOwner (ra.owner = ao ∧
        ∀ ds : DigitalSignature (ds.dependee = ra ∧
            ra.key = ao.privateKey → ar.dependee.owner = ra.owner))))
```

Consequences: */* P3: The agent's public key may not be valid or certified. A malicious agent can exploit this by signing with its own private key. */*

```
∃ ar : AccessRequest (∃ ra : RequesterAgent (ar.dependee = ra ∧
    ∃ ao1, ao2 : AgentOwner (ra.owner = ao1 ∧
        ∃ ds : DigitalSignature (ds.dependee = ra ∧
            ra.key = ao2.privateKey ∧ ao1 ≠ao2 ∧ar.dependee.owner = ao2))))
```

3.3 Agent Certification Authority

It is important to note that problem P3 is only stated in terms of the concepts used in the *Agent Authenticator* pattern. *Agent Certification Authority* adds to these the concept of a PublicKeyCertificate. A PublicKeyCertificate is signed by a trusted AgentCertificationAuthority. This proves that the publicKey of an agent is, in fact, that of the agent's owner. This publicKey can now be used to detect invalid digital signatures of other agents masquerading as the same owner. The application of this pattern does not introduce new security problems.

Problem: */* P3: The agent's public key may not be valid or certified. A malicious can exploit this by signing with its own private key. */*

Solution: */* S3: Authenticate agents only if their public key is certified. */*

$$\begin{array}{l} \forall\ \mathsf{ra} : \mathsf{RequesterAgent}\ (\forall\ \mathsf{ao} : \mathsf{AgentOwner}\ (\mathsf{ra.owner} = \mathsf{ao}\ \wedge \\ \quad \forall\ \mathsf{aca} : \mathsf{AgentCertificationAuthority}\ (\forall\ \mathsf{pkc} : \mathsf{PublicKeyCertificate}\ (\\ \qquad \mathsf{pkc.}\mathbf{dependee} = \mathsf{aca}\ \wedge \mathsf{pkc.}\mathbf{depender} = \mathsf{ra}\ \wedge \\ \qquad \mathsf{pkc.publicKey} = \mathsf{ao.publicKey} \rightarrow \mathsf{ra.publicKey} = \mathsf{ao.publicKey})))) \end{array}$$

3.4 Practical Value of the Formalisation

Although developers do not need to be aware of the formalisation when employing the proposed pattern language, its practical value cannot be underestimated. It allows us to assess the completeness of our pattern language with regard to its ability to establish security. We can observe that:

- Using the formalisation we show how the application of a given pattern results in assertions being added to the model. These allow us to formally reason about the security problems resolved by a given security solution.
- Formalisation leads to a deeper understanding of the patterns. We were able to discover non-obvious problems with a given security solution and to detect that there were patterns missing from the language to resolve them.

As an example of the former, consider the assertion made by solution S2 that the apparent initiator of an AccessRequest must equal the owner of the RequesterAgent, if the request has been signed with the initiator's private key. This eliminates the possibility of one agent masquerading as another, and is formalized as problem P2. As an example of the latter, an earlier version of the pattern language did not include *Agent Certification Authority.* This pattern was added as a means of dealing with invalid public keys, and problem P3 provides a formal justification for this extension of the pattern language.

4 Applying the Language

To make it easier to understand the practical application of the pattern language, we consider how the language was applied to the electronic Single Assessment (eSAP) system case study first introduced in [10]. The eSAP case study involves the development of an information system to support an integrated assessment of the health and social care needs of older people in England. Due to lack of space we cannot present the complete analysis here. The main secure goals of the eSAP system are: Ensure System Privacy, Ensure Data Integrity, and Ensure Data Availability. These have been further decomposed [10] into secure tasks such as Check Access Control, Check Authentication, and Check Information Flow.

According to secure Tropos, transforming security requirements to design is not an easy task, and it becomes more difficult if attempted by developers without much knowledge of security, which should be considered the norm rather than the exception. For example, from the analysis of the eSAP system, it is concluded that authentication and access control checks (amongst others) must be performed in order for the system to satisfy the system's secure goal Ensure Data Privacy. The system should be able to authenticate any agents that send a request to access information of the system, and the system must control access to its resources. Therefore, the developer must identify the appropriate actors (and their dependencies) to fulfil the above-mentioned security goals.

The proposed security pattern language can greatly help with the identification of these actors without endangering the security of the system. *Agency Guard* suggests a way of managing access to the eSAP system. *Agent Authenticator* can be used to enforce the agency's security policy. *Agent Certification Authority* describes how to certify the public key of a requester agent. *Access Controller* can be applied to perform access control checks. *Sandbox* is not applicable to the eSAP system. Not only does application of the patterns satisfy the fulfilment of the goals, but it also guarantees the validity of the solution. To apply a pattern, the developer must carefully consider the problem to be solved, and the consequences that the application of each particular pattern will have on the system. These consequences may introduce new problems that need to be resolved by other patterns until no problems remain. Figure 7 shows a possible use of the above-mentioned patterns in the eSAP system with respect to the Obtain Care Plan Information goal of the Older Person.

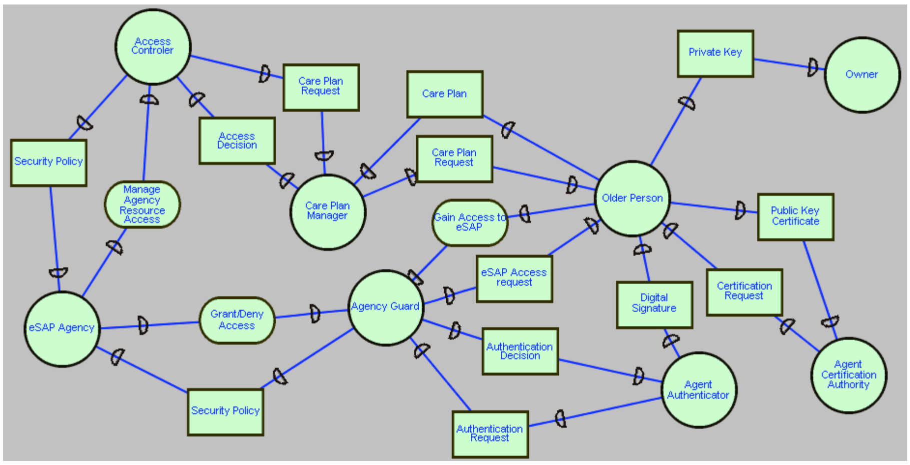


Fig. 7. Application of the patterns

We start by applying the *Agency Guard* pattern, which restricts access to the agency to a single point. As shown in Figure 7, the Older Person becomes the Requester Agent, the eSAP Agency corresponds to the Agency, and a new actor, the eSAP Guard, is introduced to assume the role of the Agency Guard. Next we apply the *Agent Authenticator* pattern to ensure the identity of the Older Person agent (the Check Authentication subgoal of Ensure Data Privacy), and the *Agent Certification Authority* pattern to ensure that the public key of the Older Person is certified. In addition, the Access Controller pattern is applied to restrict the Older Person's access only to their resources, i.e., to their own medical records. In this scenario, we assume that the Older Person should only be allowed to execute as an authorised user, and as such the *Sandbox* pattern is not applicable.

5 Conclusions

In this paper we propose an approach for the development of secure information systems that merges two important software engineering paradigms: agent oriented software engineering and security patterns. We believe this represents a suitable approach because agent orientation provides concepts such as autonomy, sociality and trust suitable for modelling security issues in information systems, whereas patterns complement agent orientation by transferring security knowledge to non security application experts in an efficient manner.

Approaches similar to ours have presented in literature. Liu et al. [8] have presented work to identify security requirements using agent oriented concepts. Jürgens proposes UMLsec [4], an extension of the Unified Modelling Language (UML), to include modelling of security related features, such as confidentiality and access control. The concept of an obstacle is introduced in the KAOS framework [5] to capture undesirable properties of a system, and to define and relate security requirements to other system requirements.

These approaches provide a first step towards the integration of security and software engineering and have been found helpful in modelling security requirements. However, they only guide the developer through how security can be handled within a certain stage of the development process. On the other hand, the area of security patterns is also very active. For example, Schumacher [13] applies the pattern approach to the security problem by proposing a set of patterns, called security patterns, which contribute to the overall process of security engineering; and Yoder and Barcalow [16] define architectural patterns for enabling application security. Fernandez and Pan [6] describe patterns for the most common security models. The main problem of these existing pattern languages is the lack of a framework to support the analysis of the security requirements and determine precisely the context in which a pattern can be applied.

By contrast, our approach merges the advantages of both the agent oriented and security patterns paradigms, by allowing developers to integrate a security pattern language within the development stages of an agent oriented software engineering methodology. This, in turn, allows developers to first analyse using agent oriented concepts the security issues related to the environment of the system, and the system itself, identify a set of security requirements needed by the system, and transform these requirements to a design that satisfies them with the aid of security patterns. However, much more work is required, and we plan to extend our pattern language to include more patterns to address security-related issues such as the privacy of the agents' information.

References

1. C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language: Towns, Buildings, Constructions*, Oxford University Press, 1977.
2. E. Amoroso. *Fundamentals of Computer Security Technology*, Prentice-Hall, 1994.
3. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos and A Perini. TROPOS: An Agent Oriented Software Development Methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, Kluwer, 8(3), 203–236, 2004.
4. J. Jürjens, UMLsec: Extending UML for Secure Systems Development, *UML 2002*, LNCS 2460, 412-425, Springer, 2002.
5. A. Dardenne, A. van Lamsweerde and S. Fickas. Goal-directed Requirements Acquisition, *Science of Computer Programming*, Special issue on the 6th International Workshop of Software Specification and Design, 1991.
6. E. Fernandez and R. Pan. A Pattern Language for Security Models, *Conference on Patterns Languages of Programs* (PLoP), 2001.
7. A. Fuxman, *Formal Analysis of Early Requirements Specifications*, MSc thesis, University of Toronto, Canada, 2001.
8. L. Liu, E. Yu and J. Mylopoulos. Analyzing Security Requirements as Relationships Among Strategic Actors, *Symposium on Requirements Engineering for Information Security* (SREIS), 2002.
9. H. Mouratidis, P. Giorgini and M. Weiss. Integrating Patterns and Agent-Oriented Methodologies to Provide Better Solutions for the Development of Secure Agent Systems, Hot Topic on the Expressiveness of Pattern Languages, *ChiliPloP*, 2003.
10. H. Mouratidis, P. Giorgini and G. Manson. When Security meets Software Engineering: A Case of Modelling Secure Information Systems. *Information Systems* (in press).
11. J. Noble. Classifying Relationships between Object-Oriented Design Patterns, *Australian Software Engineering Conference* (ASWEC), 1998.
12. J. Saltzer and M. Schroeder. The Protection of Information in Computer Systems. *Proceedings of the IEEE*, 63(9), 1278-1308, September 1975.
13. M. Schumacher. *Security Engineering with Patterns.* LNCS 2754, Springer, 2003.
14. T. Tryfonas, E. Kiountouzis and A. Poulymenakou. Embedding Security Practices in Contemporary Information Systems Development Approaches, *Information Management & Computer Security*, 9(4), 183–197, 2001.
15. M. Weiss. Pattern Driven Design of Agent Systems: Approach and Case Study. *Conference on Advanced Information Systems Engineering* (CAiSE), LNCS 2681, Springer, 2003.
16. J. Yoder, J. Barcalow, Architectural Patterns for Enabling Application Security, *Conference on Pattern Languages of Programs* (PLoP), 1997.

Kuaba Ontology: Design Rationale Representation and Reuse in Model-Based Designs

Adriana Pereira de Medeiros, Daniel Schwabe, and Bruno Feijó

Dept. of Informatics, PUC-Rio, Rua Marquês de São Vicente 225,
22453-900, Rio de Janeiro - RJ, Brasil
{adri, dschwabe, bruno}@inf.puc-rio.br

Abstract. This paper presents the Kuaba Ontology, a knowledge representation model for Design Rationale described in an ontology definition language. The representation of this model in a specific ontologies specification language, such as OWL or F-Logic, allows attributing semantics to recorded Design Rationale content, and defining rules that enable performing computable operations to support the use of Design Rationale in the design process of new artifacts. In addition, we propose to support the software design process through the use of the semantic descriptions defined by formal models of the artifacts. Representing Design Rationale using an ontology definition language and the artifacts formal model, enables a type of software reuse at the highest abstraction level, where rationales are re-employed in designing a new artifact. This kind of reuse is possible in knowledge domains where there are formal models describing the artifacts, in particular, in the Software Design domain.

1 Introduction

Designing a software artifact typically involves understanding the problem being addressed, identifying possible solution alternatives, analyzing them, and deciding which solutions will be used to construct the final artifact. The final products of this process, the artifact and its specifications, represent but a fraction of the knowledge employed by designers during the design process. They represent the final solution chosen for the particular design problem, but do not represent, for instance, the reasons that led the designers to choose that one among the other available alternatives, and why the others were discarded. In other words, they do not capture the Design Rationale (DR) that led to the artifact in question.

DR is the reasons behind design decisions. However a more complete definition is proposed by J. Lee [1]: design rationales include not only the reasons behind a design decision but also the justification for it, the other alternatives considered, the tradeoffs evaluated, and the argumentation that led to the decision. In most cases the DR is not adequately documented, which leads to requiring a high degree of verbal communication among persons that must work with an artifact, in order to understand the reasoning followed by the designer. For instance, this is fundamental when maintaining a software artifact designed by another person, or when trying to reuse it in the context of a new design. This is true even in the case of a single person, since in many cases, over a longer period of time, the designer himself may not recall all the rationale he himself used in the design of a particular artifact. Therefore, recording the DR during the design process is critical to allow its reuse.

There are several proposals in the literature for representing DR, such as IBIS [2], PHI [3], QOC [4] and DRL [5]. Most of them are incomplete or informal, not enabling machine-processable computations over the represented DR. Consequently, it is not possible to guarantee that the representation is consistent and even that it does actually provide some sort of explanation about the captured design. Furthermore, when applying them to formally defined artifacts (such as software), their informality prevents automatically taking into consideration alternatives prescribed by the design methods, as well as incorporating their restrictions. In other words, it is not possible to leverage the semantics of the artifact provided by the formal model that describes it.

For many knowledge domains, particularly in software design, there are formal models that describe the artifacts and present semantic descriptions, which allow reasoning over the artifacts being produced. In this paper, this special type of design domain is called "model-based design". An example of such a formal model is the UML specification language [6] used to describe a class diagram. A formally defined DR representation may allow integrating the formal semantics of the artifacts being designed, and allows automated computations over such representations. When such representations are available in a distributed environment, it is possible to envisage the collaboration between designers with semi-automated support, where DR representations can be searched for, recovered and integrated during the process of designing a new artifact. Such availability can therefore be the basis for collaborative (and even participatory) design, among designers working with a given artifact.

The integration of different DRs is possible only if the following conditions are satisfied: the artifacts are built from the same type of formal model (e.g. two UML class diagrams); the DRs are used to represent the same domain of application (e.g. a CD catalogue) and the DR of the artifacts is represented using the same (or compatible) representation scheme(s) - e.g. an ontology vocabulary.

A semi-automated computational environment is being built to support designers through processing of formal DR representations. This environment uses the formal model for the artifact being designed to suggest design options at each step in the design, and records the corresponding choices made by the designer, using a special purpose description language that will be described later. Depending on the richness of the formal model of the artifact being designed, the system may suggest new alternatives, and also check the consistency of decisions made by the designer. Theoretically, when formal semantics for the artifacts are available, fully automated systems could be constructed to automatically synthesize artifacts, but this is neither the approach nor the focus taken in this paper. We explicitly require human intervention in defining design steps or operations in producing the final design.

Consider the following motivating example shown in Fig.1. This example shows three design options defined by different designers to model the "*Genre*" information item in an UML class diagram modeling a CD catalogue. In Fig.1-a, the designer decided to model "*Genre*" as an *attribute* with multiplicity one or more. In Fig.1-b, the designer decided to model "Genre" as a *class* that has an association with a *CD* class, and in Fig.1-c another designer decided to model a "*Category*" information item instead of "*Genre*" to represent the same kind of information. This designer decided to model *Category* as a *class* with a self-relation of type aggregation to represent the subcategory concept.

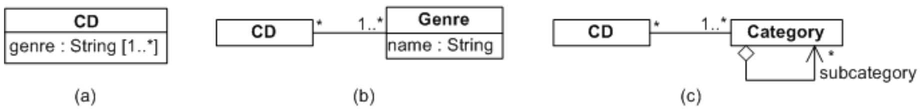


Fig. 1. Design options to model Genre information item

Since these artifacts are described in the same formal model (UML), and refer to the same domain (CD catalogues), a fourth designer could retrieve the DR representations of these artifacts (for instance, in a distributed environment), and integrate both rationales to design a new class diagram for this domain, reusing existing (partial) solutions.

In this particular case, the fourth designer could consider the "*Genre*" and "*Category*" information items to be really the same, and thus integrate the DR representations for each. For instance, she may consider modeling "*Category*" as an aggregation, but also taking the idea of allowing multiplicity one or greater, taken from the other modeling alternative. Thus, this designer could incorporate into her design the reasoning (arguments for and against each alternative considered) used by the other designers, and add her own reasons as well, finally making her own decisions.

This DR enables a type of reuse at the highest abstraction level, where rationales are re-employed in designing a new artifact. Starting with an existing DR, the designer can review and extend it, adding new alternatives or making different choices with respect to already defined alternatives, generating a new DR. From this point of view, both software maintenance and evolution can be considered as simply a continuation of a previous design process, captured in a given DR.

This paper is a significant extension and expansion of [7], in which we approach and exemplify the different uses of the Kuaba[1] ontology to representing and reusing DR. In the remainder of this paper, we first present a DR representation model for software designs, using the Kuaba ontology. Next, we address the issue of how the formal semantics of the artifacts being described can be integrated with the design process, which seen as an instantiation process of the formal model for the artifact. From this point of view, the richer the semantics of this formal model of the artifact, the greater the degree of support automation that could be achieved. Next, we present some scenarios of use for these DR representations, and the operations needed to support their reuse when designing new artifacts. We conclude by discussing related work and pointing out further work, and drawing some conclusions.

2 Kuaba: The Design Rationale Ontology

As previously mentioned, it is desirable to represent DR in a formally precise and computable way. Ontologies are good candidates for this, since, as defined in [8], they represent "an explicit specification of a conceptualization". In other words, they are knowledge representations, where a set of objects and their relationships are described through a defined vocabulary.

[1] "Kuaba" means "knowledge" in Tupy-guarany, the language of one of the native peoples in Brazil.

The Kuaba ontology describes a set of elements (classes, properties, relations and constraints) that express the DR domain. Our objective in proposing this ontology is to provide a vocabulary for DR described in an ontology definition language that allows attributing semantics to recorded DR content, and defining rules that enable performing computable operations and inferences on this content. The first version of the ontology was created using the Web Ontology Language (OWL) [9]. The current version is described in F-Logic [10] due to the availability of free inference engines.

The vocabulary described by Kuaba extends the argumentation structure of the Issue Based Information System (IBIS), whose approach for DR is to register the *issues* raised during design, the *positions* that address these issues and the *arguments* against or in favor of these positions. The extension enriches this argumentation structure by explicating the representation of the decisions made during design and their justifications, and the relations between the argumentation elements and generated artifacts. It consists also of integrating this argumentation structure with descriptions of the produced artifacts, and with information about the design history (when decisions were made, who made them, what design method was used, etc). Fig. 2 shows the elements of the vocabulary defined by the Kuaba ontology, using the UML notation to help visualization. Notice that such object oriented model is used only as a suggestion of illustration of ontology vocabulary; some relations and constraints were hidden to simplify the presentation.

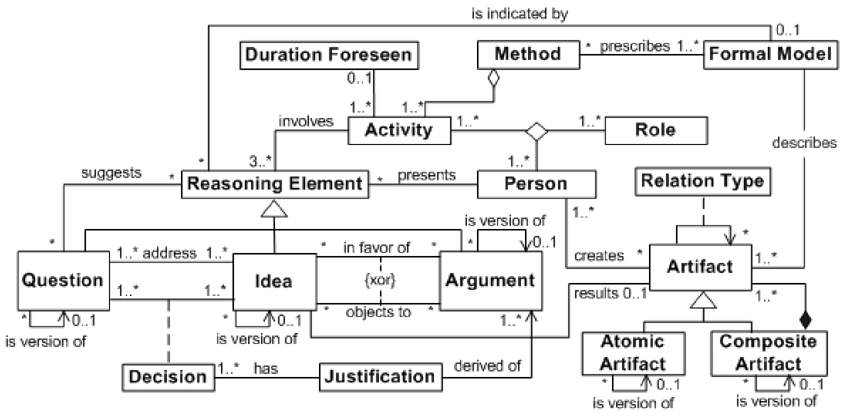


Fig. 2. The elements of the Kuaba Ontology Vocabulary

Briefly described, the Kuaba ontology vocabulary represents the people involved in a design activity and their respective roles. After defining the design method and the activities that will be undertaken, the people involved in the design activity use reasoning elements for organizing and recording their solution ideas about the artifact that is being constructed. Similarly to IBIS, these reasoning elements represent the design problems (questions) that the designer should deal with, the possible solution ideas for these problems and the arguments against or in favor of the presented ideas. In Kuaba some of these elements are described according to the formal model of the artifact prescribed by the design method used. People involved in the artifact design make decisions about the acceptance or rejection of the solution ideas presented. Each decision must have a justification that explains the "*why*" it was made. Justification is always derived from one or more arguments presented during the design. The ideas accepted during the design process originate artifacts that can be either atomic artifacts or composite artifacts. All reasoning elements (*Question*, *Idea* and *Argument*) and artifacts have a "*is-version-of*" relation, representing the fact that any one of them may be based on an existing element. This element may be either part of a previous version of this same artifact, and therefore the design is actually evolving it, or part of a different design that is being reused in a new context.

Below we show a portion of the Kuaba ontology vocabulary shown in Fig. 2 expressed using F-Logic.

```
// CONCEPTS --------------------
question::reasoning_element.
idea::reasoning_element.
reasoning_element[hasText->STRING; hasCreationDate->STRING;
                  isInvolved->activity; suggests->>question;
                  isPresentedBy->person;isIndicatedBy->formal_model].
question[hasType->STRING; isAddressedBy->>idea; hasDecision->>decision;
         isSuggestedBy->>reasoning_element; isVersionOf->question].
idea[address->>question; results->artifact;
     isConcludedBy->decision; isVersionOf->idea].
decision[isAccepted->BOOLEAN; hasDate->STRING; isMadeBy->>person
         concludes->idea; hasJustification->justification].
// ALGEBRAIC PROPERTIES OF RELATIONS (INVERSE) --------------------
FORALL X,Y X[address->>Y]  <-> Y[isAddressedBy->>X].
FORALL X,Y X[concludes->>Y] <-> Y[isConcludedBy->>X].
```

2.1 Representing Reasoning Elements and Decisions

Normally, the first activity done by the designer in designing a software artifact is the choice of design method or process that will be used to achieve the design. When the designer chooses a design method or process, she indirectly determines the formal model(s) (specification language) that will be used to describe the artifact. For example, when a designer chooses the Unified Process [11] to achieve the artifact design she indirectly determines the formal model defined by the UML to describe this artifact.

The existence of a formal model for the artifact determines, to a great extent, the questions and ideas that the designer can propose, since they are pre-defined by this model. In this sense, designing an artifact according to the method amounts to a stepwise instantiation of the formal model. Consider once again the motivating example shown in Fig. 1; if the designer has chosen the Unified Process, the DR will be expressed in terms of questions, ideas and arguments defined by the UML formal model for class diagrams.

The DR representation usually begins with a general question that establishes the problem to be solved. This general question can generate new questions that represent new design (sub) problems related to the main problem. For each question presented the designers can suggest ideas, formulating possible solutions to the problem expressed in the question.

According the UML formal model, the first problem to be solved in designing a class diagram is the identification of its constituting elements. Applying the vocabulary described by Kuaba, this results in instantiating the "*Question*" class with the instance "*What are the model elements?*". Fig. 3 shows a graphical representation we have created to help visualizing instances of the Kuaba ontology, showing the portion of the design regarding the alternatives to model "*Genre*" in the motivating example (Fig.1-a and Fig.1-b). In this representation, the root node is an initial question (represented as rectangles), "*What are the model elements?*", which is addressed by the ideas "*CD*", "*Genre*" and "*Name*", represented as ellipses. Notice that these values are determined by the designer's knowledge of the domain, or were extracted from the DR of a previous phase, requirements elicitation, which is not addressed in this paper.

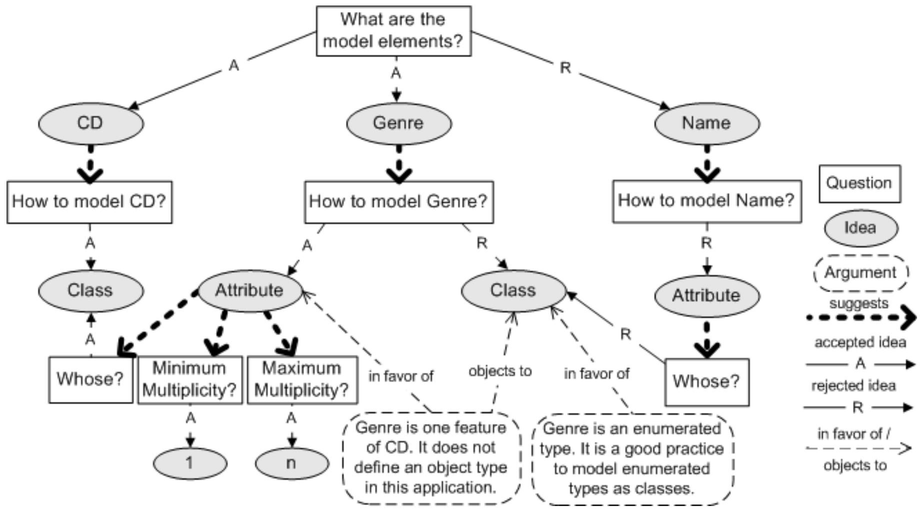


Fig. 3. A portion of a DR representation regarding "Genre" in the motivating example (Fig. 1)

Once these first alternative ideas for the CD Catalogue model elements have been established, the designer must decide how each one of them will be modeled using the UML, to make up the final artifact, the class diagram. This next step is represented in Fig. 3 by the "suggests" relation, which determines questions entailed by ideas – "*How to model CD?*", "*How to model Genre?*" and "*How to model Name?*".

The possible ideas that address these questions are determined by the UML formal model for class diagrams – elements can essentially be a class, an attribute, or an association. Accordingly, the "*Class*" and "*Attribute*" ideas linked to the "*How to model Genre?*" node are established as an instantiation of the UML formal model. Strictly speaking, the designer should consider all the other alternatives proposed by the UML, but for the sake of simplicity we have shown only these two. Since the "*Attribute*" idea, in turn, must be associated with a "*Class*" according to the UML model, the question “*Whose?*” is suggested, which in turn will be addressed by the idea corresponding to the class with that attribute it is. Similarly, the questions “*Minimum Multiplicity?*” and “*Maximum Multiplicity?*” are also defined by the UML model to be associated with the idea “*Attribute*”.

In this way, it is possible to envisage how the formal model “drives” the instantiation of the Kuaba ontology recording the DR. This formal model can be used by a support environment to suggest to the designer the possible Kuaba ontology instances that must be defined at each step of the process. The designer has to choose the desired alternative, and record the arguments for and against each option (represented as dashed rectangles in Fig. 3). It is through the “Argument” element that the designers can record the experiences and the knowledge that they are employing in the artifact design.

In Fig.3, we also show the final decision made, indicating each alternative answer to each question with an “A” (for accepted) or “R” (for rejected) label. Thus, the example represents the fact that the designer decided to accept the “*Attribute*” alternative to the question “*How to model Genre?*”, in detriment of the “*Class*” alternative.

The acceptance or the rejection of an idea as a solution to a question is recorded by the “*Decision”* element. Differently from IBIS, in our ontology the acceptance or rejection of an idea is represented as a property of the relation between the elements *Question* and *Idea*, as shown in Fig. 2. We consider that the acceptance or rejection of an idea is not an intrinsic property of the “*Idea*” element, but must be defined with respect to a certain “*Question”*, since the same idea can address more than one question, and be accepted for one and nor for another.

The sub-graph of the DR made up of “Question” and “Ideas” is actually an AND/OR graph [12] that can be seen as a goal decomposition of the root node, which is always a “Question”. The “*Question*” class in the Kuaba ontology has a “type” attribute, with possible values “AND”, “OR” and “XOR”. The value “XOR” indicates that all ideas that address this question are mutually exclusive, meaning that only one idea can be accepted as a solution to the question. The value “AND” indicates that the designer should accept all ideas that address the question or reject all of them. Finally, the value “OR” indicates that various ideas can be accepted as a solution to the question. This kind of information allows us to define rules that can suggest decisions about the acceptance or not of the proposed solution ideas. For example, the software could suggest rejecting certain ideas based in the following rule: if an idea associated with a question of type “XOR” is accepted by the designer, then all other ideas associated to this question will be rejected. This rule is formulated in F-Logic as follows:

```
FORALL Q, I1, I2, D1, D2 D2[isAccepted->>"false"]
 <- Q:question[hasType->>"XOR"; isAddressedBy->>{I1,I2};
               hasDecision->>{D1,D2}]
 AND D1:decision[isAccepted->>"true"; concludes->>I1:idea]
 AND D2:decision[concludes->>I2:idea]
 AND NOT (equal (I1,I2)).
```

Notice that, if the decision to reject the idea of modeling “Genre” as a class was the first one made by the designer, a support system could apply the rule above, and automatically propose that the idea of modeling “Genre” as an attribute be accepted, given that there are only two ideas associated with this question. At this point, the designer also has the option of not accepting this suggestion, and revising the possible answers to the original question “What are the model elements?”, rejecting “Genre” altogether. In any case, the support environment can apply consistency rules defined by the Kuaba ontology, as well as those expressed for the particular formal artifact representation being used. Therefore, the order of accepting or rejecting an idea does not affect the represented rationale.

2.2 Representing Artifacts

A useful part of a DR representation must associate the designed artifact itself, or its components, to the corresponding design decisions that led to them being the way they are. This integrates the artifact description with the DR description. In Fig. 2, this is represented by the “*Results*” relation between “*Idea*” and “*Artifact*”.

An artifact corresponds to a final design solution, made up of a set of accepted ideas in the DR representation. Therefore, in a DR representation every artifact must be associated with at least one idea, and at least one of them must have been accepted. Clearly, an artifact cannot be associated with an idea that was rejected.

In the Kuaba ontology vocabulary, artifacts are represented by two classes, *Atomic Artifact* and *Composite Artifact*. For example, in the UML formal model a class can be seen as an aggregation of attributes, and therefore an element modeled as a class can be modeled as a composite element. This is the case, for example, of class “*CD*” in Fig. 3. In Fig. 4 we show an example of an artifact representation after the decisions shown in Fig. 3 have been made. In particular, notice that the “*Genre*” element is represented as an *Atomic Artifact*, since the designer decided to model it as an attribute of the CD class.

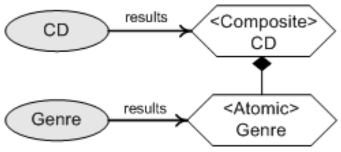


Fig. 4. Examples of Artifacts

3 Using the Representations of Design Rationale

As exemplified previously, representing DR in a more formal language and using the artifact formal model permits us to assign semantics to recorded content, enabling processing this content by computable operations. These operations, in turn, give support to new scenarios of use of DR in model based designs of software artifacts.

Consider the scenario where a designer wishes to construct a class diagram to represent information that will be used in a CD Store application. Since the online stores domain is a common domain in software design, the designer decides to perform a search for existing designs in a distributed environment, trying to find similar artifacts, before she begins a new design. As a result, she finds two different class diagrams for the CD domain with their respective DR representations. After receiving the search result and analyzing the artifacts found, the designer decides to reuse these artifacts taking advantage of the knowledge already used by other designers that is recorded in the available DR representations. In this scenario the designer can reuse a found artifact in two ways. She can import its DR representation and begin her design based on it; or she can integrate the DR representations of both found artifacts to compose a more complete solution of design.

An example of the first type of reuse is the artifact evolution process, in which a copy of a DR representation of the artifact is used as a starting point for the design, and subsequently modified, generating a new artifact (the new version of the original one). Incorporating a DR representation of an artifact means to import a set of already defined reasoning elements into the design that is being performed. Imagine that the designer had selected the first class diagram found and that in this diagram the author had considered the "*Genre*" and "*Record Label*" information items, as Fig. 5 shows.

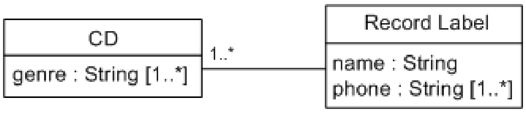


Fig. 5. Original Class Diagram

In this diagram the author decided to model the "*Genre*" information item as an attribute of a CD class and the "*Record Label*" item as a class. Fig. 6 shows the graphical representation of the reasoning elements incorporated into the new design, after the designer has selected the first diagram found. These elements represent the reasoning used by the author of the artifact to model the "*Record Label*" information

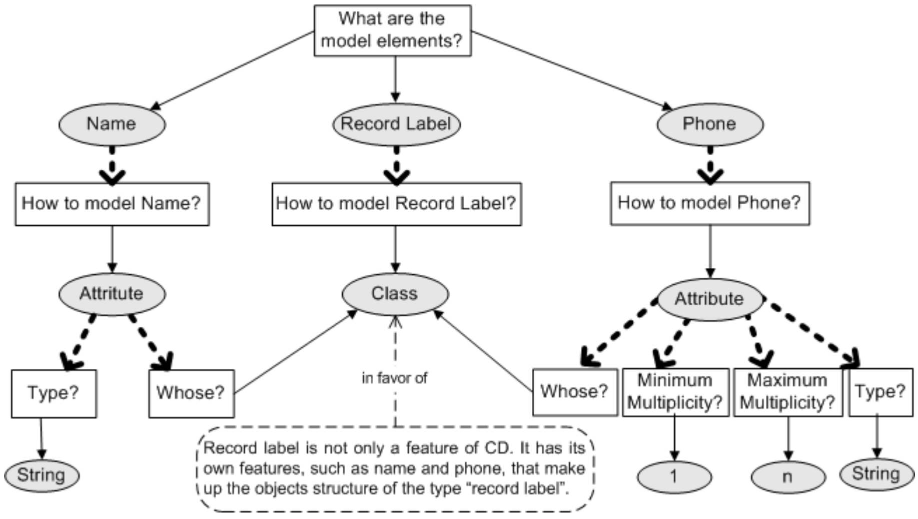

What are the model elements?
Name
Record Label
Phone
How to model Name?
How to model Record Label?
How to model Phone?
Attritute
Class
Attribute
Type?
Whose?
in favor of
Whose?
Minimum Multiplicity?
Maximum Multiplicity?
Type?
String
Record label is not only a feature of CD. It has its own features, such as name and phone, that make up the objects structure of the type "record label".
1
n
String

Fig. 6. Example of DR about the *Record Label* information item

item in his class diagram. According to this DR representation, the author considered the idea of designing "*Record Label"* as a class and the ideas of designing the "*Name*" and "*Phone*" information items as attributes of this class.

Notice that the decisions made for the imported artifact design are not incorporated to new design (there are no arrows labeled "A" and "R"). This reflects the fact of the incorporation of an existing DR representation and its modification represents a new design, or a design continuation, whose DR should also be recorded. In this new design, new decisions should be made according to the new designer's objectives, although the decisions previously taken by the author of the imported DR can be obtained from the DR representation of the original artifact.

In this first example we can suppose that, after analyzing the solution ideas incorporated into the new design, the designer had access to the idea of designing "*Record Label*" as an element of her class diagram, an idea that she had not considered before. We can also assume that the designer decided to modify the DR incorporated into her design including a new solution idea, the idea of designing the element "*Record Label*" as an attribute of a *CD* class.

Observing the decisions shown in Fig. 7 we can conclude that the designer decided to model the "*Record Label*" element as an attribute of the *CD* class and to reject the solution ideas for the "*Name*" and "*Phone*" elements.

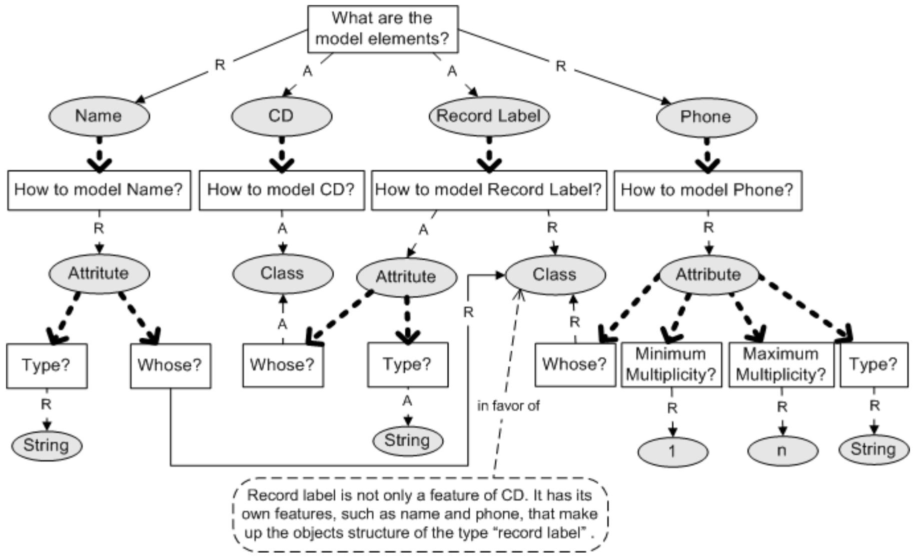


Fig. 7. DR for the new artifact, the "Record Label" attribute definition

Fig. 8 shows the design solution used by the designer after reasoning about the design of the "*Record Label*" element.

To exemplify the second way of reusing artifacts, the reuse through integration of existing DR representations, we will use the motivating example presented in section 1. In this second form of reuse, the designer selects the three artifacts found by the search service and, after analyzing the DR of these artifacts, she performs the integration of their respective representations.

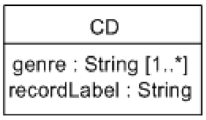


Fig. 8. Newly designed artifact, the "CD" class definition

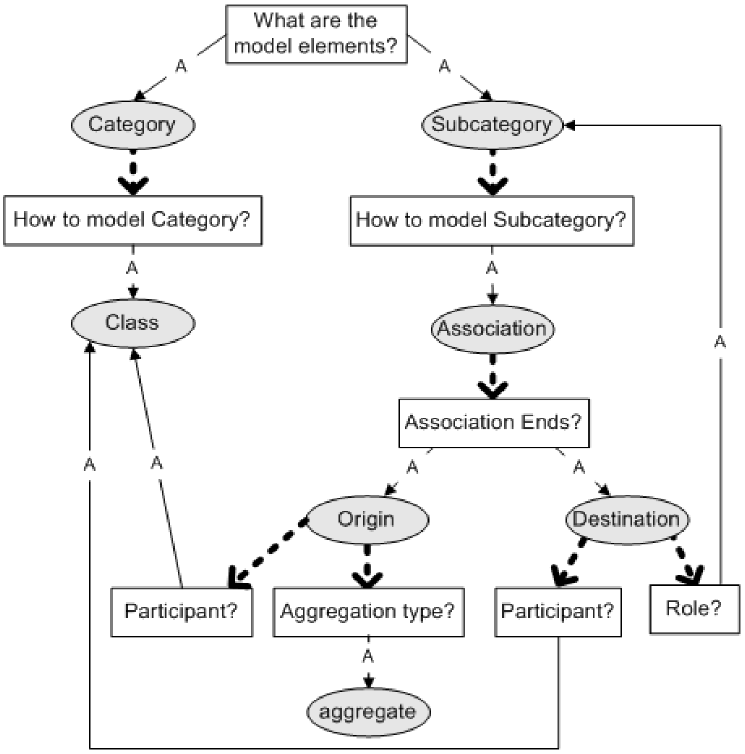


Fig. 9. Example of DR about the Category Element

Part of the DR of the first artifact selected by the designer is shown in Fig. 3. Examining this DR, the designer verifies that the author of the select artifact decided to model the "*Genre*" element as an attribute of the *CD* class after considering the use of attribute multiplicity to model more than one genre for the same CD.

Continuing her analysis, the designer consults the DR of the third artifact found, shown in Fig. 9, and verifies that the author of this artifact, instead of considering an element "*Genre*" in the class diagram, considered the element "*Category*" and decided to model this element as a class. Besides the "*Category*" element, the author also decided to include the element "*Subcategory*" in his diagram to model the concept of subcategories as an aggregation of the Category class.

Next, the designer establishes that the "*Genre*" and "*Category*" elements in each of the DR representations are actually the same concept in the CD domain, and formally specifies this identity so that a computational environment can integrate the reasoning elements related to them. After specifying the identity between these elements, the designer defines that the representation of the second artifact (Fig. 9) will be used as the basis for the integrated representation, and performs the integration of the DR representations of the selected artifacts. Fig. 10 shows the graphic representation of part of the DR that was recorded after the integration of the rationales exemplified in figures 3 and 9.

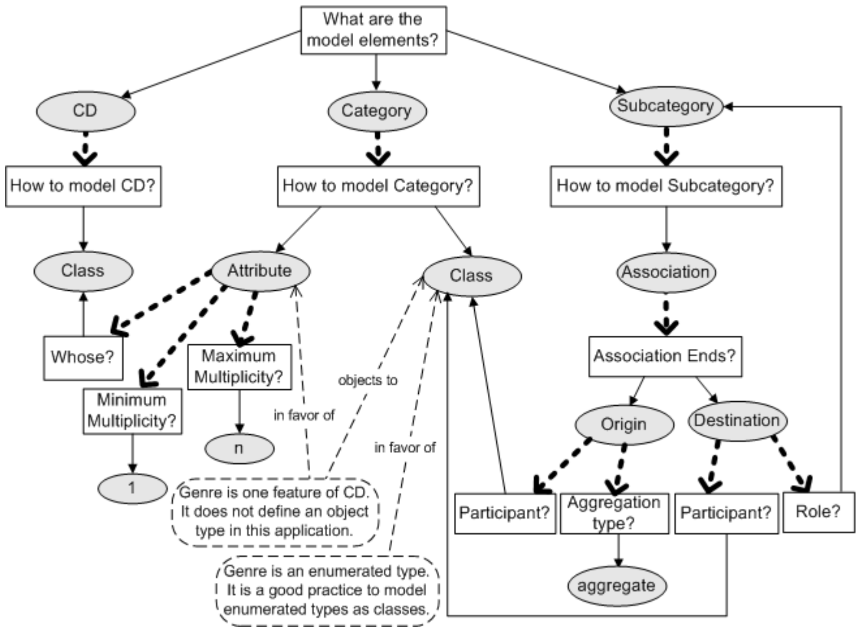


Fig. 10. Example of integrated DR

Observing the DR integration result shown in Fig. 10, we can notice that the solution idea "*Attribute*" that addresses the question "*How to model Genre?*" in Fig. 3 and all other questions, solution ideas and arguments represented there, were automatically included in the sub-tree of the question "*How to model Category?*". With this integrated DR, the designer can begin her design evaluating the solution ideas already used taking advantage of the experiences of other designers, expressed in the arguments and decision justifications included in the DR. Based on this knowledge, she can modify the DR obtained and make new decisions about how her new class diagram will be designed.

The DR represented with the Kuaba ontology vocabulary could also be used in scenarios that involve cooperation. We believe that the support operations necessary for the reuse of DR in cooperation scenarios are the same operations necessary in the scenarios presented previously. In other words, the operations just described can be applied in exactly the same way when integrating partial solutions produced by team members in a cooperative design effort. We will discuss the necessary operations on DR representations next.

3.1 Operations

The use of the DR representations requires different kinds of operations on the recorded content. The explicit and semantic representation of DR in a language formally defined and specifically designed for the description of ontologies, allows these operations to be computable by engines to support the design of new artifacts.

The operations over DR representations can be grouped into queries, operations for manipulating an existing DR representation and operations for integrating of two or more DR representations (instances of the Kuaba ontology)

The first group allows formulating relevant questions about the design process and about the produced artifact. For example, we can formulate questions such as "*What were the solution ideas considered during the Genre artifact design?*", "*Why the decision of designing the Genre artifact as an attribute was made?*", "*Who presented the solution idea adopted for the Genre artifact?*", or still "*Is artifact X related to artifact Y*?" The queries are performed according to the relations semantics and constraints defined by the Kuaba ontology vocabulary.

The operations for the manipulation of the existing DR representations are basically the same operations necessary to represent the reasoning elements (*Question*, *Idea* and *Argument*) produced by the designer during the design of a new artifact. These operations involve the creation and destruction of instances of the classes and properties defined in the Kuaba ontology. They are implemented in the majority of available ontology manipulation tools, such as Protégé [13].

The operations for the integration of two or more DR representations involve more specific treatments. These operations resemble the operations necessary to perform ontology alignment [14]. However, the operations for the integration of the DR representations defined in this work differ from the usual alignment operations, since they involve matching instances of the same ontology (Kuaba) and not the matching of taxonomies and instances defined in different ontologies. The types of operations identified for the integration of two or more DR representations are: search, copy, union and substitution.

Search operations allow the designer select which elements of the representations considered for the integration will be included in the integrated representation. For example, the designer could provide a question, and request that the search tool recover only the ideas that have arguments in its favor that are answers to this question.

Union operations allow joining the reasoning elements described in the representations considered in the integration to generate a new representation. These operations should take into account the identity specifications and the definition of the base representation previously defined by the designer. These operations can be implemented in different ways, allowing the designer to determine how the union of elements will be performed. One way would be to permit the designer to specify which parts of the representations considered should be integrated. For example, she could define the *Question* element that would be the root of the union of the representations. Or still, to allow her to restrict the elements considered during the integration, such as, for instance, requiring the union to consider only the ideas that were accepted in their respective representations.

Finally, the substitution operations allow the designer substitute an element in one representation by a corresponding element from another representation. This operation can be used for example when the designer desires to use the DR of a single representation, but she desires to substitute one of its elements by an element specified in the other representation.

Some of these operations make use of inference rules to automate part of the designer work in recording the DR of the artifact that she is building.

4 Related Work

Procedural Hierarchy of Issues (PHI) extends IBIS by simplifying the relations among issues by using the "serve" relationship only. In addition, it provides two methods to deal with design issues: deliberation (to give answers to the issue) and decomposition (to break down the issue into a variety of subissues). Differently the Kuaba ontology, PHI does not represent explicitly the decisions made during design and its justifications. Plus, it is not integrated with the artifacts descriptions and other information about the design process.

The Potts and Bruns [15] model relates entities in existing software engineering methods to IBIS-based deliberation. This model was extended by [16] in the creation of the Decision Representation Language (DRL). Similarly to the Kuaba ontology, the key difference from other DR representations, namely the integration with software engineering methods, is achieved through derivation of artifacts from alternatives. However, the Potts and Bruns model and the Kuaba ontology differ in the way they use software engineering methods. In Potts and Bruns model, the generic model's entities are refined to accommodate a particular design method's vocabulary for deriving new artifacts. For example, a new entity specific to the used design method is incorporated into the IBIS model. In Kuaba ontology the vocabulary of the design method is used in the creation of instances of the reasoning elements (*Question* and *Idea*), which allows to automate the generation of part of the values that would be informed by the users during design.

Although the works presented in [17] and [18] approach DR specifically for software engineering and focus on the (re)use of DR, they are not directly comparable to our work because they do not address the integration of DR representations to create new software artifacts. Furthermore, these works do not consider computable operations as a support for the reuse of software artifacts.

5 Conclusions

In this paper we have proposed a new way of reusing artifacts in Software Design domain. To permit a more effective reuse of DR, we have presented the use of the formal models of artifacts to represent DR using the vocabulary defined in Kuaba ontology. This vocabulary represented in a specific ontology specification language allows formulating queries on the recorded DR and to define a set of rules and operations to support the use of DR in designing new software artifacts.

One of the problems related to DR representation is the use of the formalism by the people involved in design process. We believe that the use of formal models of artifacts in a software development environment can facilitate the DR capture, since it permits automating part of generation of DR representations. Therefore, the large amount of data produced in DR representations of actual designs is significantly hidden from the designer through the use of automated support.

Our current research includes: the implementation of the operations defined in this paper to validate the reuse of software artifacts through the integration of existing DR representations; and the investigation of the use of the Kuaba ontology to represent DR also in domains where there are no well defined formal models to describe artifacts. The operations are being implemented using the Flora-2 language[2] that translates F-Logic into tabled Prolog code.

References

1. Lee, J.: Design Rationale Systems: Understanding the Issues. IEEE Expert Volume 12, No. 13 (1997) 78-85
2. Kunz, W., Rittel, H. W. J.: Issues as Elements of Information Systems. Institute of Urban and Regional Development Working Paper 131, Univ of California, Berkeley, CA, (1970)
3. McCall, R. J.: PHI: A conceptual foundation for design hypermedia. Design Studies, No.12 (1) (1991) 30-41
4. MacLean, A. et al.: Questions, Options, and Criteria: Elements of Design Space Analysis. Human-Comput. Interaction, No. 6 (3-4) (1991) 201-250
5. Lee, J., Lai, K.: What's in Design Rationale. Human-Comput. Interaction, No. 6 (3-4) (1991) 251-280
6. OMG: Unified Modeling Language Specification version 1.5. March (2003)
7. Medeiros, A. P., Schwabe, D., Feijó, B.: A Design Rationale Representation for Model-Based Designs in Software Engineering. In Proceedings of the CAiSE'05 Forum, Porto, Portugal, June (2005) 163-168
8. Gruber, T. R.: A Translation Approach to Portable Ontologies. Knowledge Acquisition, No. 5 (1993) 199-220
9. W3C: OWL Web Ontology Language Overview. W3C Recommendation. February (2004)
10. Kifer, M., Lausen, G.: F-Logic: A Higher-Order Language for Reasoning about Objects, Inheritance and Scheme. ACM SIGMOD May (1989) 134-146
11. Jacobson, I., Booch, G., Rumbaugh, J.: The Unified Software Development Process. Reading, Mass.: Addison-Wesley, (1999) 463p
12. Nilsson, N.: Principles of Artificial Intelligence. Morgan Kaufman Publishers (1986) 476p
13. Noy, N. F. et al.: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems Vol. 16, No 2, Special Issue on Semantic Web, March/April (2001) 60-71
14. Doan, A. et al.: Learning to Map between Ontologies on the Semantic Web. In Proceedings of the 11th World Wide Web Conference (WWW2002), Honolulu, Hawaii, USA, May (2002)
15. Potts, C., Bruns, G.: Recording the Reasons for Design Decisions. In Proceedings of 10th International Conference on Software Engineering, Singapore (1988) 418-427
16. Lee, J.: Extending the Potts and Bruns Model for Recording Design Rationale. In Proceedings of the 13th International Conference on Software Engineering, Austin, TX (1991) 114-125
17. Pena-Mora, F., Vadhavkar, S.: Augmenting Design Patterns with Design Rationale. Artificial Intelligence for Engineering Design, Analysis, and Manufacturing (1996) 93-108
18. Burge, J., Brown, D. C.: Rationale Support for Maintenance of Large Scale Systems. Workshop on Evolution of Large Scale Industrial Software Applications (ELISA), ICMS'03, Amsterdam, NL (2003)

[2] http://flora.sourceforge.net/

Ontology Creation: Extraction of Domain Knowledge from Web Documents

Veda C. Storey[1], Roger Chiang[2], and G. Lily Chen[1]

[1] Department of Computer Information Systems, J. Mack Robinson College of Business, Georgia State University, Box 4015, Atlanta, GA 30302
vstorey@gsu.edu, gchen@cis.gsu.edu
[2] Information Systems Department, College of Business, University of Cincinnati, Cincinnati, Ohio 45221-0211
roger.chiang@uc.edu

Abstract. Considerable research has gone into developing ontologies and applying them to a variety of applications. The extraction of domain knowledge for developing these ontologies is often performed on a manual basis. The World Wide Web contains a wealth of knowledge about an application domain; however it is embedded within web pages. This research presents a methodology for semi-automatically extracting knowledge from the World Wide Web and organizing it into domain ontologies. Initial semantics of a target domain are provided by a set of keywords. From these, web pages are identified that contain relevant information for the subject domain using search engines. Web data extraction techniques are employed to extract information from these web pages and infer how the information is related. Extracted knowledge is then organized into a domain ontology. Testing of the methodology on various application domains illustrates the feasibility of the approach.

1 Introduction

The potential of ontologies as a way to gather and organize real world knowledge about an application domain has made them popular and, potentially, very useful for a variety of applications, including the Semantic Web, and other search engines projects [1, 2]. Ontologies capture knowledge about an application domain but are often build manually. The World Wide Web contains a great deal of domain knowledge; however it is embedded within its web pages. It would be useful to develop a methodology to extract and organize domain knowledge efficiently and effectively. If this could be done for various application domains, then it would assist in the automated or semi-automated creation of domain ontologies.

The objective of this research, therefore, is to: *develop a methodology for automatically extracting and organizing domain knowledge from web pages into ontologies.* To do so, web data extraction, concept identification and relationship inferencing are applied. The contribution of the research is to provide a first step towards developing a methodology for semi-automatically generating domain ontologies. Then, the most recent knowledge of a particular domain would be readily organized on-line. The intent of this research is to move one step forward to developing a generic method to achieve this goal. The research also provides insights into the challenges of how knowledge is organized on websites and how it might be heuristically extracted. The research should also serve as a step towards making ontologies useful for the Semantic Web, and other applications.

2 Related Research

2.1 Ontologies

Ontologies are intended to be an effective way to capture and represent contextual knowledge about the real world. An ontology is a way of describing one's world and generally consists of terms, their definitions, and axioms relating them [3] although there are many different ways of defining ontologies [4]. Ontologies are found in many areas including the semantic web, natural language processing and text interpretation [5], data extraction, managing semantic heterogeneity [6], and classifying relationships in conceptual database design [7, 8].

A *domain ontology* generally consists of the terms that occur in some application domain (e.g., auction, course scheduling, airline reservations, restaurant selection) and the relationships among them [4]. Consider the following query:

"Who sells Bosendorfers in Atlanta?"

Searching an ontology for musical instruments reveals that this is a type of fine piano. Therefore, one should search piano sales stores. However, commonly used search engines such as www.google.com retrieve a multitude of results as shown in Table 1.

Table 1. Google results based on query "Who sells Bosendorfers in Atlanta?"

Keywords	**No. Google Hits**	**No. Relevant from first 10**
Bosendorfers Atlanta	73	5
Bosendorfers Atlanta grand (from WordNet)	67	8
Bosendorfers Atlanta grand model	49	9

Now, consider another query:

"Which writer won the Oscar for Best Screenplay in 1972?"

A search on Google retrieves tens of thousands of results as shown in Table 2. Of those, 60-80 percent of the initial ten results are incorrect.

This query suggests that ontologies could be beneficial when identifying whether two dissimilar words refer to the same entity. In the United States, for example, "Oscar" can refer to "Academy Award." This is common human knowledge, but not machine knowledge. Searching an ontology for movies would reveal that "Oscar" is often used as a synonym for "Academy Award." A web search would, therefore, be more effective if it incorporated such domain knowledge (as stored in an ontology) that would automatically search websites containing both Oscars and Academy Awards, even if only one of the two terms were used.

Table 2. Results retrieved from Google queries based on the query "Which writer won the Oscar for Best Screenplay in 1972?"

Keywords	No. Google Hits	No. Relevant from first 10
Which writer won the Oscar for Best Screenplay in 1972	21,800	4
Writer Oscar Screenplay 1972	23,600	2

There are many reasons for developing an automatic domain knowledge generation methodology as outlined below.

Knowledge Sharing: Domain ontologies should facilitate knowledge access and sharing for both Semantic Web applications and on-line communities on a special topic. For example, many people and organizations may post ideas, opinions, suggestions, policies, history, etc. on tsunamis. A search on Google with tsunami as the keyword yields over 22 million websites. Although this is a large number of hits, people accessing websites might not have the same domain knowledge or understanding of tsunamis. It is infeasible and impossible to manually create and maintain a domain dictionary to support emerging Internet applications. Thus, an evolving ontology would be very useful.

Semantic Information Retrieval: With the large number of web pages available for a particular domain, it is beyond a human's cognitive capability to search, retrieve, read, and understand these web pages. Although search engines are useful tools to partially solve this problem, search engines have almost reached their full potential to further improve information search on the Internet. A semantic-oriented solution is imperative to support semantic information retrieval. A domain ontology could support as the mental and conceptual map of a particular topic, so that users can conduct effective searches with the domain knowledge provided by the ontology. For example, for a user who wants to search on "knowledge management," it would be useful if he or she could access an on-line ontology on this topic first.

Domain Knowledge Updating: An automatic approach to creating domain ontology also provides additional benefits. For example, a domain ontology can evolve (grow and update) along with the evolution of the Internet, because the Internet is the good medium capturing changes and storing the most up-to-date information on most topics and issues.

2.2 Web Data Extraction and Web Mining

A number of techniques and methods have been proposed for extracting data and concepts from the World Wide Web. Data extraction tools have been developed based upon techniques from research in natural language processing, languages and grammars, machine learning, information retrieval, database and ontologies [9]. Natural language processing (NLP) tools apply techniques such as filtering, part-of-speech tagging, and lexical semantic tagging to build relationships between phrases and sentence elements so that Web data extraction rules can be derived. Such extraction rules are based on syntactic and semantic constraints that help to identify relevant information within a document. Representative tools are RAPIER [10], SRV [11], and WHISK [12]. Ontology-based tools use the ontology to locate constants present in the Web pages and to construct objects with them. The most representative ontology-based Web data extraction tool was developed by the Brigham Young University Data Extraction Group [13].

Web mining is the use of data mining techniques to automatically extract and discover information from Web documents and services [14]. It is invaluable in transforming human understandable Web content into machine understandable semantics. Web mining research draws upon research from several areas including databases, information retrieval, and artificial intelligence (AI), especially machine learning and natural language processing. Web mining techniques can be applied to: (1) automatically discover general patterns at individual websites and documents as well as across multiple sites and documents, and (2) validate and/or interpret the mined patterns. Web mining can be used to improve Web information extraction. Web mining can be divided into three categories: Web content mining, Web structure mining, and Web usage mining [15]. Web content mining is a form of text mining that discovers interesting patterns from unstructured text and hypertext data. Research on Web content mining can be used to detect co-occurrence of terms in texts. For example, co-occurrences of terms in movie Web documents may show that "Oscar" and "Academy Awards" are frequently mentioned together. Research on Web content mining should be useful for the creation of domain ontologies by extracting domain knowledge from Web documents.

2.3 Web Query Processing

Research on and interest in the Semantic Web has prompted studies that attempt to apply ontologies, and other concepts from knowledge-based systems, natural language processing, and information retrieval to make the processing of queries more effective. One study focused on developing a Semantic Retrieval System to support context-aware query processing on the web [18]. The system is based upon query expansion and contraction, using lexicons such as WordNet [17] and ontologies such as the DAML library (`www.daml.org`) to augment a query. The prototype connects to search engines (Google and AlltheWeb) to execute the augmented query. An empirical test of the methodology and comparison of results against those directly obtained from the search engines demonstrate that the proposed methodology provides more relevant results to users and shows that lexicons and ontologies provide valuable, complementary sources of knowledge. However, the results obtained are highly dependent upon the availability of good domain ontologies.

3 Ontology Construction Methodology

This section presents a methodology for semi-automatically extracting domain knowledge from websites and organizing them into domain ontologies. An overview of the methodology is given in Figure 1 and is explained below based upon an application that deals with screenplays and Academy Awards.

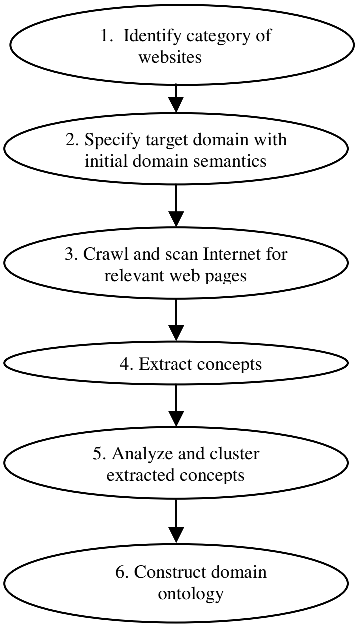

1. Identify category of websites
2. Specify target domain with initial domain semantics
3. Crawl and scan Internet for relevant web pages
4. Extract concepts
5. Analyze and cluster extracted concepts
6. Construct domain ontology

Fig. 1. Overview of ontology web extraction methodology

Step 1: Identify Category of Website: There have been several attempts to categorize the vast and diverse websites that exist on the World Wide Web. The classification suggests that there is a degree of similarity of websites that belong to the same category. Therefore, there is a higher-level (meta-level) organization of websites based on type. In addition, content may be identified as similar based on the category of the website. Referring to similarities within each category helps to identify different domain processing. Categories include eduction, news, etc. The categories from Xavier.edu are shown in Appendix A [16]. Two of them are:

1) commercial, the purpose of which is to sell products or services. The Internet address often ends with *.com*
2) entertainment, where the purpose of the website is to entertain and provide amusement. The Internet address often ends with *.com* or *.org*.

The web search intends to retrieve results from a certain category of website. Specifically, for the screenplay application, it intends to retrieve information to inform or entertain, and, therefore, aims to retrieve websites of type *entertainment* as opposed to *commercial* sites. Then, the specific entertainment site within the domain of "Movie" is searched. Under Movie, there are several subdomains, including Awards, Characters, DVD, Genres, and History, as shown in Appendix D. Table 3 shows only the awards subdomain and the terms or concepts that are related to it. The initial set of websites is generated to avoid random searching. The website classification, shown in Table 2, is used because it provides some notion of context. For example, the sites already identified as commerce sites will be searched for "buying" applications. Those for entertainment are the most appropriate for the query focusing on academy awards.

Table 3. Subdomains of entertainment domain “movie”

Subdomain	Subdomain	Term
Awards	Academy Awards Golden Globe Awards Screen Actor’s Guild Awards	Oscar

Step 2: Specify target domain with initial domain knowledge: An initial set of domain knowledge is expressed as a set of keywords that is used as the seed to identify relevant websites from which to extract information (similar to the initial knowledge for data mining). Another way to obtain an initial set of keywords is to process a simple, representative query. For example, to develop an ontology for film/movies and awards, one could start with a query such as “Which films won Oscars?” A search on Google results in almost 230,000 hits. The first of these reveals “film,” “academy award,” and “nomination” as common terms. The initial keywords are given below.

Oscar, Academy Award, screenwriter, writer, best writing, adaptation

The initial set of keywords is used as the seed to identify relevant websites from which to extract information. In this example, the keywords, “writer,” “Oscar,” ”academy award,” and “best screenplay” are used to start building an ontology for the domain. This is shown in Table 4 where they are used with WordNet to start the initial list of terms.

Table 4. Step 3: Extract keyworks from related webpages “movie, screenplay, oscar, academy award” (Source: human input)

Query Terms	Synonyms for Query Terms	WordNet Synonyms
Writer	Screenwriter	scriptwriter
Oscar	Academy Award	award, accolade, honor, honour, laurels
Best Screenplay	Best Writing, Adaptation and Best Original Story	script, book, playscript

Step 3: *Crawl and scan web pages:* The relevant pages for the example are shown in Table 5.

Table 5. Websites to crawl and scan

Website	Description
http://www.oscars.org/academyawards/	Academy awards, Academy of Motion Pictures Arts and Science
http://www.oscar.com/	Oscar awards and related information
http://www.filmsite.org/	Film information, including Oscar awards

Step 4: *Extract concepts:* There are many approaches to extracting data from the web [2]. This step involves extracting terms (nouns and verbs), their frequencies and

relationships (e.g., if one term appears in a sentence, another term also appears 50% of the time). Relevant subset/superset relationships are identified by consulting the online version of WordNet [17], `www.cogsci.princeton.edu/~wn/`) (a comprehensive lexicon of the English language). This is a very important step; because the quality of a domain ontology is determined by the extracted information.

This step identifies what kind of extracted information can best be used to create domain ontologies. Based on the input-process-output model, this step generates input (raw material, syntactical information) for the next step in inferring semantics (domain knowledge). The steps for the concept extraction are given in Table 6.

Table 6. Concept extraction

General Steps	Example
1) Open websites and analyze them	Google produces 21,800 results for query
2) Analyze terms in navigation bars	Oscar, Academy Award, screenwriter,
1. top navigation bar	writer, best writing, adaptation
2. left side navigation bar	
3. sub navigation bars	
3) Add terms to domain ontology list	

Step 5: *Analyze and cluster extracted features:* A content analysis of the extracted concepts of the web pages is carried out. This involves feature analysis and clustering methods. For each term (concept), other nouns and verbs to which they are related are included

A content analysis of the extracted features is carried out. It includes identifying how the various terms are connected to each other. This is shown in Table 7.

Table 7. Step 5: Extract keyworks from related webpage www.filmsite.org/ bestscreenplays2.html "Oscar Academy Award screenwriter writer best writing adaptation"

Terms	Frequency	Relationship
Oscar	31	
Academy Award	5	
To Win (all tenses)	101	
To Nominate	11	
Nomination	44	
Oscar + Academy Award (O/AA)	1	Oscar – academy award
O/AA + Nominated	1	Oscar – academy award -- win
Writing	7	
Screenplay	25	
Screenplay + Writing	1	Screenplay—write
Adapted + Screenplay	2	Screenplay—adapt
Original + Screenplay	4	Screenplay—original
Best + Screenplay	11	Screenplay—best
Oscar+win	20	Oscar—win
Nomination + Oscar	20	Oscar—nomination
Nomination + Win	20	Nomination--win

Step 6: *Construct domain ontology:* A domain ontology is constructed using the terms and relationships identified. The ontologies presented in Appendix B and C are organized by website type (commercial, news) with domain.

The steps in the methodology are summarized in Table 8.

Table 8. Steps in ontology creation methodology

Step	Step Name	Description	Justification
1	**Website Category**	Identify the category of website to help identify target domain	Existing classifications of websites exist; reduces search space
2	**Target domain**	Identify the target domain for creating the ontology (manual input).	The web is too large to search blindly
2.1	**Initial semantics**	Provide the initial semantics of the target domain (i.e., a set of key words)	Similar approach has been used by research in web mining
3	**Web search**	Search for web pages for the relevant information for the application domain using search engines	
4	**Extract**	Download the relevant web pages	
4.1	**Eliminate unrelated content**	Scan the web pages and strip unrelated contents using web data extraction tools that eliminate non-keywords and stop words	Stop words have been effectively applied in other, related research
4.2	**Extract features**	Extract and analyze these extracted feature (feature identification and analysis) using feature extraction and analysis tools	The purpose of application domain ontologies is to provide common terms of the domain [4]
5	**Cluster feature**	Arrange features as ontological hierarchy using clustering techniques	Standard procedure, currently based upon frequencies
6	**Organize ontologies**	Organize the ontologies using the XML structure or a relational database structure for queries.	Takes advance of standard technologies

4 Assessment

The methodology was tested on several domains. First the domain ontologies were generated based upon the methodology presented above. Then, queries were generated by application domain. Since the intended use of the ontologies is to assist in processing such queries, testing was carried out by analyzing the number of relevant web pages retrieved (out of the first 10) with and without the information that would be available in the domain ontology. The results are summarized in Table 9.

Table 9. Results

Application Domain	Testing Query	Google result without ontology terms	Google result with ontology terms
Education	Which different universities offer online degrees?	5	9
Sports-wear Sales	What stores sell Nike shoes?	4	8
Music-Concerts	When is U2 playing in Atlanta?	2	9

The assessment provides some insights into both the challenges of the development of domain ontologies; and the usefulness of domain ontologies for query processing on the World Wide Web

1. The extraction of concepts from domains, when carried out systematically, provided a number of useful terms. However, when the domain was not as closely related, the number of occurrences of common terms went down drastically. For example, websites that reported on the Oscars or Academy Awards contained much overlap in terms. However, when extracting concepts from the more generic "film" websites, there was more variation in the terms used.
2. There were generic websites, such as all airlines. However, for the specific, related websites, such as Travelocity, there were new concepts and hence terms.
3. The websites contain a wealth of information on application domain concepts and relationships.
4. Observations can be made on the format of the websites. There is a great deal of commonality in websites of a certain category. Websites that belong to the same type of classification tend to be designed with similar layouts and have a similar "look and feel". There are generic terms (shown Appendix B) that appear on almost all websites of a certain category (e.g., all "commercial" websites have a shopping cart, etc.). However, the websites may use synonyms (e.g. shopping cart/basket/mycart or "plane/airfare/flight"), and differ in the exact term used. This would make ontology domains very useful.
5. The use of the correct domain information can both decrease the number of hits while increasing the relevancy.

5 Conclusion

A methodology for creating domain ontologies has been presented. The methodology is based upon data extraction and other search techniques to take advantage of the terms and concepts that are embedded within web pages. The methodology has been applied to various application domains. The results are encouraging, with future research needed to refine the methodology and to further test and apply the results. Further work is also needed to augment the methodology to incorporate more sophisticated clustering analysis and learning techniques. A prototype that implements the methodology needs to be developed that incorporates good natural language parsing techniques and, perhaps interfaces with other lexicons or ontologies. The prototype could be integrated with existing tools that enhance web query processing. Finally, the application of the research to other domains is needed and further empirical assessment carried out, both to assess the usefulness of the research and to provide insights into what further work is needed on concept extraction from the web, domain ontology development, and domain ontology applications to web query processing.

References

1. Chiang, R., Chua, E. H., Storey, V. C.: A Smart Web Query Engine for Semantic Retrieval of Web Data," presented at Data and Knowledge Engineering Special Issue on International Conference on Applications of Natural Language to Information Systems, (NLDB '00), 2001.
2. Embley, D. W.: "Toward Semantic Understanding: An Approach Based on Information Extraction Ontologies," presented at ACM International Conference Proceeding Series; Proceedings of the fifteenth conference on Australasian database, Dunedin, New Zealand, 2004.
3. Gruber, T. R.: "A Translation Approach to Portable Ontology Specifications," *Knowledge Acquisition*, vol. 5, pp. 199-220, 1993.
4. Weber, R.:"Ontological Issues in Accounting Information Systems," in *Researching Accounting as an Information Systems Discipline*, S. a. A. Sutton, V, Ed. Sarasota, FL: American Accounting Association, 2002.
5. Dahlgren, K.: "A Linguistic Ontology," *International Journal of Human-Computer Studies*, vol. 43, pp. 809-818, 1995.
6. Kedad, Z., Métais, E.: "Dealing with Semantic Heterogeneity During Data Integration.," presented at Conceptual Modeling - ER'99, 18th Intl. Conference on Conceptual Modeling, Lecture Notes in Computer Science 1728, Paris, France, 1999.
7. Bergholtz, M., Johannesson, P.: "Classifying the Semantics of Relationships in Conceptual Modeling by Categorization of Roles," Madrid, Spain June 28-29 2001.
8. Storey, V. C.: Classifying and Comparing Relationships in Conceptual Modeling," *IEEE Transactions on Knowledge and Data Engineering*, vol. forthcoming, 2005.
9. Laender, A. H. F., Ribeiro-Neto, B. A., Silva, A. S. d.,Teixeira, J. S.: "A Brief Survey of Web Data Extraction Tools," *ACM SIGMOD Record*, vol. 31, pp. 84 - 93, 2002.
10. Califf, A. M. E., Mooney, R. J.: "Relational learning of pattern-match rules for information extraction," presented at Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence, Orlando, Florida, 1999.
11. Freitag, D.: "Machine Learning for Information Extraction in Informal Domains," *Machine Learning*, vol. 39, pp. 169 - 202, 2000.
12. Soderland, S.: Learning Information Extraction Rules for Semi-Structured and Free Text," *Machine Learning*, vol. 34, pp. 233 - 272, 1999.
13. Embley, D. W., Campbell, D. M., Jiang, Y. S., Ng, Y.-K., Smith, R. D., Liddle, S. W., Quass, D. W.: Conceptual-model-based data extraction from multiple-record Web pages," *Data & Knowledge Engineering*, vol. 31, pp. 227-251, 1999.
14. Etzioni, O.:"The World-Wide Web: Quagmire or Gold Mine?," *Communications of the ACM archive*, vol. 39, pp. 65-68, 1996.

15. Kosala, R., Blockeel, H.:Web Mining Research: A Survey," *SIGKDD Explorations*, vol. 2, pp. 1-15, 2000.
16. Xavier, `http://www.xavier.edu/library/xututor/evaluating/types_of_websites.cfm,`" vol. 2005.
17. Fellbaum, C.:Introduction," in *WordNet: An Electronic Lexical Database*. Cambridge, Mass.: The MIT Press, 1998, pp. 1-19.
18. Burton-Jones, A., Storey, V.C., Sugumaran, V., and Purao, S., "A Heuristic-based Methodology for Semantic Augmentation of User Queries on the Web," *Proceedings of the 22nd International Conference on Conceptual Modeling (ER'03)*, Chicago, Illinois, 13-16 October 2003, pp.476-489.

Appendix

Appendix A. Types of Websites summarized from Xavier.edu [16]

Type	Purpose	Example
Organizational	The purpose of this type of website is to advocate an individual's opinion or a group's point of view. The Internet address often ends with *.org*	www.w3.org
Commercial	The purpose of this type of website is to sell products or services. The Internet address often ends with *.com*	www.amazon.com
Entertainment	The purpose of this type of website is to entertain and provide amusement. The Internet address often ends with *.com*	www.people.com
Government	The purpose of this type of website is to provide information produced by government agencies, offices, and departments. Usually, information provided by government websites is very reliable. The Internet address often ends with *.gov*	www.firstgov.gov
News	The purpose of this type of website is to provide information about current events. The Internet address often ends with *.com*	www.cnn.com
Personal	The purpose of this type of website is to provide information about an individual. The Internet address has a variety of endings	www.billycorgan.com
Educational	The purpose of this type of website is to provide information about an educational establishment. The Internet address ends in .edu	www.gsu.edu

Internet Service Provider	The purpose of this type of website is to promote companies and services related to the Internet. The Internet address ends in .net	www.virgin.net
Military	The purpose of this type of website is to provide information about the military. The Internet address ends in .mil.	www.defenselink.mil

Appendix B. Ontology of websites of type "commercial"

Type		General	Specific
All		Cart, Account, Wish List, Help, Search, Register, Sign In	Gift Certificates, New Releases, Top sellers Deals/Sales
Bookstore		What's New, New Releases, Best Seller, Ordering Information, (Find) Location Browse, Used	Nonfiction, Fiction, Childrens, Book Clubs
	URL	Top Column	Left Column
	Amazon	Welcome, Your Store, Books, Apparel, Electronics, Toys & Games, DVD, Tools & Hardware	Categories: Books, Music, DVD, electronics, Kids & Baby, Home & Garden, Gifts & Registries
	BooksA Million	Home, Books, Bargains, Magazines, B-2_B, FaithPoint, Audio, Hard-To-Find, Joe Muggs	Editor's Choice, Discount magazine subscriptions, Discount Card, Book Preview Clubs, Store Finder, Gift Certificates, Gift Wrap, Online Gift Delivery
	Powells Indep.	Browse selections, bestsellers, sale, used, rare books, technical, textbooks, kids, new	Categories: architecture, art, astronomy.
	Barnes and Noble	Books, Used & Out of print, business & Technology, DVD & Video, Music, Childrens, Gifts, Games & Toys, Gift Cards, Sale, Browse, Coming Soon, Recommended, Book Clubs	
Auction	All	Buy, Sell, Pay, Register, Services	
	eBay	Buy, Sell, My eBay, Community, Help	Categories – Antiques, art, Books, Business, Cameras…

Airline	All	Flights, Hotels, Cars, Rail, Vacations, Cruises, Travel, Flight, Destination, Business, About	Flight, flight hotel, hotel, car, From, To, Airports, Leave/Depart, Return Adults, Minors, Seniors Search Flights
	Travelo-city	Flights, Hotels, Cars, Rail, Vacations, Cruises, Last Minute Deals, Travel Info Center, Flight Status, destination, Business, About	Find me the best Priced Trip, flight, hotel, car, From, To, Compare surrounding airports, Exact dates, Depart, Return, Adults, Minors, Seniors, Track fares, Search Flights, Need Ideas
	Orbitz	Quick Search, flights, hotels, cars, cruises, vacations, my trips, my account, deals, news & guides, customer service	Flight, hotel, car, build your trip Flight, flight hotel, hotel, car, From, To Compare surrounding airports, Search one day before and after, Leave, Return, Travelers, Adults Minors, Seniors, Track fares, Explore destinations, Search Flights
DVD			
	Rhino	Store, music, dvd & video, rhino t-shirts, collectibles, ringtones,	New Releases, upcoming releases, rhinos finest, gift ideas, top sellers, contests, quick shop, cds, dvd/video rhino handmade, cellphone tones and pix, boxed set, vinyl, dual disc dvd-a, sacd
	Warner Bros.	Home, movies, television, dvd, mobile, games, kids, win!, music	New releases, family, tv on dvd, classics, special interest, sports

Appendix C. Ontology of websites of type "news"

Type		General	Specific
All News		Search Home	News, travel, sports
	URL	Top Column	Left Column
News	CNN	Search	World, U.S. Weather, Business, Sports, Politics, Law Technology, Science & Space, Health, Entertainment, Travel, Education, Special Reports

	USA Today	Search		News, travel, money, sports, life, tech, weather, essentials, scores, news briefs, people, today in the sky, day in pictures, video, archives, print edition, subscribe, contact us

Appendix D. Breakdown of subdomains within domain of "movie"

Domain	SubDomain	Websites	Terms
Movie	Awards	Academy Awards Golden Globe Awards Screen Actor's Guild Awards	Oscar
	Character	Etc.	Etc.
	Chats & forums		
	Contests		
	Cultures and Groups		
	Databases		
	Directories		
	DVD		
	Education		
	Film Festivals		
	Filmmaking		
	Genres		
	History		
	Home Video		
	Memorabilia		
	Multimedia		
	News and Media		
	Organizations		
	Personal Pages		
	Quotations		
	Release Schedules		
	Reviews		
	Screenwriting		
	Scripts		
	Series		
	Showtimes		
	Soundtracks		
	Studios		
	Theaters		
	Theory and Criticism		

Choosing Appropriate Method Guidelines for Web-Ontology Building

Sari Hakkarainen[1], Darijus Strasunskas[1,2], Lillian Hella[1], and Stine Tuxen[1]

[1] Norwegian University of Science and Technology,
Sem Saelandsvei 7-9, NO-7491 Trondheim, Norway
[2] Dept. of Informatics, Kaunas Faculty of Humanities,
Vilnius University, LT-44280 Kaunas, Lithuania
{sari, dstrasun, hella, stinemt}@idi.ntnu.no

Abstract. Ontology is a core component in semantic Web applications. The employment of an ontology building method affects the quality of ontology and the applicability of ontology language. An evaluation approach for ontology building guidelines is presented. The evaluation is based on semiotic quality framework, an evaluation scheme frequently applied for evaluating the quality of conceptual models. The framework is extended with situational and computational capabilities. A sample of ontology building method guidelines is analyzed in general and evaluated comparatively in a case study at an oil company in particular. Directions for further refinement of ontology building methods are discussed.

1 Introduction

The vision for the next generation web is the *semantic Web* [2], where information is accompanied by metadata about its interpretation, so that more intelligent and more accessible information-based services can be provided. The core components in the semantic Web and its applications will be the ontologies. An ontology can be seen as an explicit representation of a shared conceptualization [11] that is formal [31], and will encode the semantic knowledge and enable sophisticated information services. The quality of a semantic Web application will be dependent on the quality of its underlying ontology.

Even though tool support exists, the ontology building process still is a human intensive labour. The analysis and design phases rely on human interpretation. In practice, the biggest communication gap is observed between system designers and end-users, e.g. domain experts. System designers use a terminology at the syntactic level, often unfamiliar to the users. Similarly, the terminology used by the latter group may be difficult for the former to understand. This is apparent when trying to integrate different levels of abstraction - pragmatic, semantic and syntactic. Transition from one level to another is not trivial.

The end product of ontology building is not always a homogeneous specification, but loosely coupled ontology fragments focusing on different aspects. The co-ordination of the process of modelling and integration of different views is possible by the medium of common reference layer [26]. Ontology may be used as means to abstract from different representation formats and to relate various fragments at different abstraction levels. The quality of the interoperation and views management will depend on the quality of the used ontology. The quality of the ontology will in turn depend on factors such as 1) the appropriateness of the language used to represent the ontology, and 2) the quality of the engineering environment, including tool support and method guidelines for creating the ontology by means of that language. Method guidelines can thus be seen as an important means to make ontology building possible for a wider range of developers, e.g., not only a few expert researchers in the ontology field but also companies wanting to develop an ontology for internal or external use.

The objective is to inspect available method guidelines for Web-based ontology specification languages and to apply and extend a modelling quality framework in order to facilitate the choice of ontology building method based on requirements. The approach is to adapt the modelling language appropriateness part of the semiotic quality framework [18], define a computational framework for the analytic evaluation of method guidelines, situate the weighted criteria and to conduct a trial case study.

The paper is organized as follows. In Section 2, related work is discussed. In Section 3, the model for the evaluation is described and the computational extension of the model is introduced. In Section 4, the ontology building guidelines are evaluated and their means to achieve quality goals in general are analysed. In Section 5, the language and method guidelines are evaluated in an industrial case in particular. Finally, in Section 6 conclusions are drawn and directions for future work and for further refinement of ontology building methods are suggested.

2 Evaluating Modelling Approaches and Web-Ontologies

Related work for this paper comes from two sides: a) work on ontology representation languages and methods for these, and b) work on evaluating conceptual modelling approaches (i.e., languages, method guidelines, and tools). The intersection between these two is limited; the work on ontology languages has contained little about evaluation, and the work on evaluating conceptual modeling approaches has concentrated on mainstream approaches for systems analysis and design. However, the newer ontology languages are becoming mature enough to allow comparative analysis, given a suitable instrument.

During the last decade, a number of ontology representation languages have been proposed. The so-called traditional ontology specification languages include: CycL, Ontolingua, F-logic, CML, OCML, Telos, and LOOM. There are Web standards that are relevant for ontology descriptions in semantic Web applications, such as XML and RDF. Finally, there are the newer Web ontology languages that are based on the layered semantic Web architecture, such as OIL, DAML+OIL, XOL, SHOE, and OWL. The latter are at the foci of this study.

There exist several methodologies to guide the process ofWeb-ontology building. Usually, they describe an overall ontology development process yet not the ontology creation itself. The methodologies primarily intend to support the processes of knowledge elicitation and the management of the ontologies: [8] propose an evolving prototype methodology with six states as ontology life-cycle and includes activities related to project and ontology management, [28] propose an application driven ontology development process in five steps emphasizing the organisational value, integration possibilities and the cyclic nature of the development process, [29] propose a top-down approach for deriving domain specific ontologies from common upper level ontologies and includes steps for requirements elicitation and for implementing the derived ontologies, and [30] propose a general framework for the ontology building process consisting of four steps including quality criteria for ontology formalisation.

The above methodologies provide a life cycle in an overall ontology development process [3, 7, 30] but only a few user guidelines for actually creating the ontology. In order to increase the number, and the scale of practical applications of the semantic Web technologies, the developers need to be provided detailed instructions and general guidelines for the actual ontology creation. A limited selection of method guidelines were found for the Web ontology specification languages, which are at the foci of this study: [14] present a tutorial with method guidelines for making ontologies in the representation language OWL by means of the open source ontology editor Protégé, [5] present a user guide with method guidelines for making ontologies in the representation language DAML+OIL, again by means of Protégé, and [22] present method guidelines for making ontologies, independent of any specific representation language.

Difficult aspects to control are the human factors that affect the quality of ontology. Developer constructs an ontology based on individual interpretation of reality and perception of high quality. The human factors influence the pragmatic use of the ontology language in the construction process and in the resulting ontology. The ontology construction process in Figure 1 is related to some ontology language yet independent of any. The sequence of ontology building activities and the mapping rules between the UoD and the ontology are important aspects in ontology building. Currently, Web-ontology languages do not encompass explicit mapping between the real world phenomena and the ontological constructs.

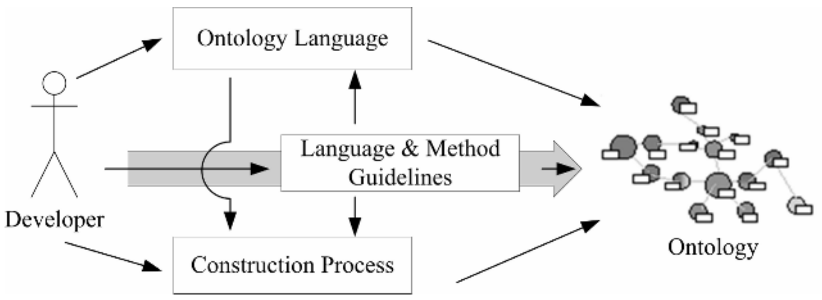


Fig. 1. Factors that affect a final ontology

A comprehensive evaluation of ontology languages was carried out in [27], covering all the above mentioned languages except OWL. It also evaluates some tools for ontology building: Ontolingua, WebOnto, WebODE, Protégé 2000, OntoEdit, and OilEd. Similarly, [4, 10, 24] evaluate various ontology languages. These studies focus on evaluating the languages and partly tools, not hands-on instructions or ontology building guidelines. Given the argumentation above, such studies are targeting the audience of highly skilled modelling experts rather than the wide spectrum of potential developers of semantic Web applications.

In conceptual modelling there are, however, a number of frameworks suggested for evaluating modelling approaches in general. For instance, the Bunge-Wand-Weber ontology [32] has been used on several occasions as a basis for evaluating modelling techniques, e.g. NIAM [33] and UML [23], as well as ontology languages in [4, 9]. The semiotic quality framework first proposed in [19] for the evaluation of conceptual models has later been extended for evaluation of modeling approaches and used for evaluating UML and RUP [15, 16]. This framework was also the one used in the above mentioned evaluation of ontology languages and tools in [27]. The framework suggested by [24] is particularly meant for requirements specifications, but is still fairly general. There are also more specific quality evaluation frameworks, e.g. [1] for process models, and [21, 25] for data and information models, respectively.

The semiotic quality framework [18] builds on an earlier framework [19]. The early version distinguishes between three *quality categories* for conceptual models (syntactic, semantic, and pragmatic) according to steps on the semiotic ladder [6]. The *quality goals* corresponding to those categories are syntactic correctness, semantic validity and completeness, and comprehension. The framework distinguishes between goals and *means to reach the goals*, where, e.g., various types of method guidelines are an example of the latter. In later extensions by Krogstie, more quality categories have been added, so that the entire semiotic ladder is included, e.g., *physical*, *empirical*, *syntactic*, *semantic*, *pragmatic*, *social*, and *organizational* quality. In this work, the framework is used for evaluating something different, namely method guidelines for ontology building.

3 Weighted Selection Criteria for Appropriateness Categories

The developers typically need instructions and guidelines for ontology creation in order to support the learning and co-operative deployment of the semantic Web enabling languages. [18] describes a model quality framework consisting of five semiotic aspects of quality modelling languages. The framework is well-suited as an evaluation instrument because it 1) distinguishes between goals and means separating what to achieve from how to achieve, 2) is closely related to linguistics and semiotic concepts, and 3) is based on a constructivist world-view, the framework recognizes that models are build from interaction between the designer and the user. The main model, see Figure 2. of the semiotic quality framework for language, here web-ontology guideline evaluation is as follows.

A - **Audience** refers to the individual, A_i, organisational, A_s, and technical, A_t actors who relate to the model. This includes human and artificial actors.
K - **Participant knowledge** is the explicit knowledge that is relevant for the audience A. This is the combined knowledge of all participants in the project.
L - **Language extension** is what can be represented according to the graphical symbols, vocabulary and syntax of the language; the set of all statements that may be informal L_i, semi-formal L_s, or formal L_f.

M - **Model externalization** is the set of all statements in an actor's model of a part of a perceived reality written in a language *L*.
I - **Social actor interpretation** is the set of all statements which the externalized model consists of, as perceived by the social audience A_i and A_s.
T - **Technical actor interpretation** is all the statements in the conceptual model *L* as they are interpreted by the technical audience A_t.
D - **Modelling domain** is the set of all statements that can be stated about a particular situation.

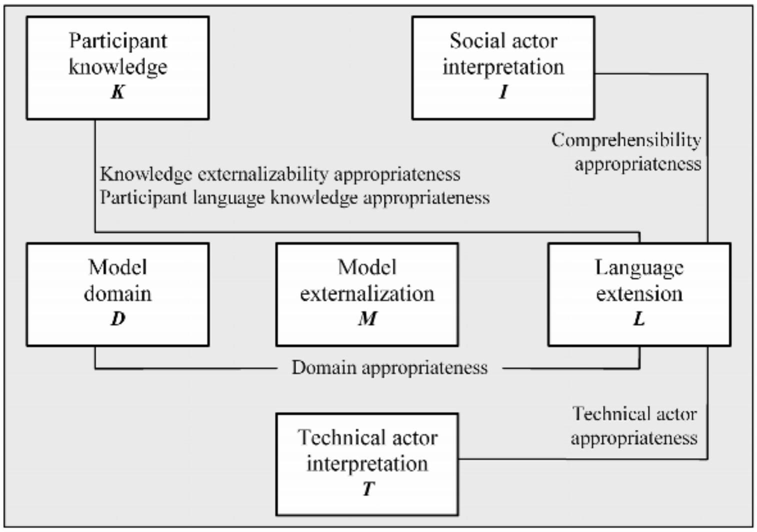

Participant knowledge K
Social actor interpretation I
Comprehensibility appropriateness
Knowledge externalizability appropriateness
Participant language knowledge appropriateness
Model domain D
Model externalization M
Language extension L
Domain appropriateness
Technical actor appropriateness
Technical actor interpretation T

Fig. 2. Semiotic Quality Framework [18]

Five appropriateness categories in the framework are here adapted for ontology building method guidelines. Selection criteria and coverage weight functions are proposed accordingly. The case study [12, 13] suggested that numerical values could be used for classification and thus qualify weighted selection techniques, such as the [20] PORE methodology for evaluation. Adapting PORE, we define coverage weights -1, 1 or 2 for each category in the sequel.

Let *CF* be an evaluation framework such that *CF* has a fixed set *A* of appropriateness categories *a*. Each *a* is a quadruple <*id*, *descriptor*, *C*, *cw*>, where *id* is the name of the category, *descriptor* is a natural language description, *C* is a set of selection criteria *c*, and *cw* defines a function of *S* that return -1, 1, or 2 as coverage weight, where *S* is a set of satisfied elements a_ic_j in the selection criteria *C* of each appropriateness category *a* in *A*. Intuitively, we define a number of selection criteria alongside an associated coverage weight function for each category in the evaluation framework. The appropriateness categories are as follows.

a_1 – ***Domain appropriateness*** *(DA)* indicates whether the method guidelines address representation of relevant facts. Ideally, $D \setminus L = \emptyset$, i.e. there are no statements in the expected application domain that cannot be expressed in the target language, and one should not be guided to express things that are not in the domain (limited number of constructs). The former criterion means that a_1c_1 – the developer is guided to make use of high expressive power whereas the latter means that a_1c_2 – there is a limited number of modelling constructs that are generic, composable and flexible in precision. The *cw* in Equation 1 holds for each modelling perspective a_1p_1– *structural* (SP), a_1p_2 – *functional* (FP), a_1p_3 – *behavioural* (BP), a_1p_4 – *object* (OP), a_1p_5 – *communication* (CP), and a_1p_6 – *actor*-role (AP).

$$cw_1(S_1) = \begin{cases} 2, & if \quad a_1c_1 \wedge a_1c_2 \in S_1. \\ 1, & if \quad a_1c_1 \vee a_1c_2 \in S_1. \\ -1, & if \quad S_1 = \varnothing. \end{cases} \tag{1}$$

a_2 – ***Participant knowledge appropriateness** (PKA)* indicates whether the method corresponds to what participant in the modelling activity perceive as a natural way of working. Ideally, K ∩ L \ L = Ø, that all the statements in the models of the languages used by the participants are part of their explicit knowledge. The *cw* in Equation 2 return values based on that the method guideline a_2c_1 –usage of statements not in a participant's knowledge should not be promoted, a_2c_2 – external representation should be intuitive, and a_2c_3 – non-intuitive representations should be introduced carefully.

$$\mathrm{cw}_2(S_2) = \begin{matrix} -1, & if & |\mathrm{S}_2| = 0. \\ 1, & if & 0 < |\mathrm{S}_2| \le 1. \\ 2, & if & 2 < |\mathrm{S}_2| \le 3. \end{matrix} \tag{2}$$

a_3 – ***Knowledge externalization appropriateness** (KEA)* indicates whether the method assists the participants in externalizing their knowledge. K ∩ L \ K = Ø, i.e. there are no statements in the explicit knowledge of the participant in the modelling activity that cannot be expressed in the target language. This appropriateness focuses on how relevant knowledge may be articulated in the language rather than what knowledge is expressed. The *cw* in Equation 3 reflects on the implication of the two partial quality goals of generality, a_3c_1 – the guidance to use the language should be as domain independent as possible, and completeness, a_3c_2 – there is guidance for all possible usages of the language.

$$cw_3(S_3) = \begin{cases} 2, & if \quad a_3c_1 \wedge a_3c_2 \in S_3. \\ 1, & if \quad a_3c_1 \vee a_3c_2 \in S_3. \\ -1, & if \quad S_3 = \varnothing. \end{cases} \tag{3}$$

a_4 – ***Comprehensibility appropriateness** (CA)* indicates whether participants are able to comprehend the method guidelines. Ideally, L \ I = Ø, i.e. all the possible statements of the language are understood by the participants using the method guidelines. The *cw* in Equation 4 reflects if a_4c_1 – the described modelling constructs are easily distinguished from each other, a_4c_2 – the number of constructs is reasonable or organized in a natural hierarchy, a_4c_3 – proposed use of modelling constructs is uniform for all the statements expressed in the target language, a_4c_4 – the guidance is flexible in the level of detail in the target language, and a_4c_5 – separation of concerns and multiple views is supported.

$$cw_4(S_4) = \begin{matrix} -1, & if\ 0 < |S_4| \leq 1. \\ 1, & if\ 1 < |S_4| \leq 3. \\ 2, & if\ 3 < |S_4| \leq 5. \end{matrix} \tag{4}$$

a_5 – ***Technical actor interpretation appropriateness*** *(TAIA)* indicates whether the method guidelines lend themselves to automated tool support or assist in support for reasoning. Ideally, T \ L = Ø, i.e. all possible mechanisms in the technical participants interpretation are supported by the target language. The *cw* in Equation 5 reflects on the implication of the partial quality goals for automatic reasoning support in the instructions provided for the target language, i.e. a_5c_1 – both formal syntax and semantics are operational and/or logical, a_5c_2 – efficient reasoning support is provided by executability, a_5c_3 – natural language reasoning is supported, and a_5c_4 – information hiding constructs are provided enabling encapsulation and independent components.

$$cw_5(S_5) = \begin{cases} 2, & if \quad a_5c_1 \wedge (a_5c_2 \vee a_5c_3 \vee a_5c_4) \in S_5. \\ 1, & if \quad a_5c_1 \vee a_5c_2 \vee a_5c_3 \vee a_5c_4 \in S_5. \\ -1, & if \quad S_5 = \varnothing. \end{cases} \tag{5}$$

The selection criteria for the appropriateness categories above are exhaustive and mutually exclusive in the categories a_1, and a_6, exhaustive in a_5, whereas the set of satisfied criteria *S* of the remaining categories may also be the empty list $\varnothing$. The coverage weight *cw* is independent of any category-wise prioritisation. Further, since the intervals are decisive for the coverage weight they can be adjusted depending on preferences of the evaluator and the stakeholder. However, when analysing different evaluation occurrences the intervals need to be fixed in comparison, but may be used as dependent variable.

4 Guidelines for Ontology Building – General Evaluation

The extended evaluation framework provides means to evaluate the development perspectives of a methodological support instrument, independent on a particular ontology language. Our method is illustrated in Figure 3, where two parallel tracks of applying the semiotic quality framework provide guidance to what an evaluation process may contain. Different levels of appropriateness highlight various aspects, such as the modelled domain, the participants' previous knowledge, or the extent to which participants are able to express their knowledge.

Three method guidelines for semantic Web-based ontology languages are categorized, namely *Denker, 2003* [5], *Knublauch et al., 2003* [14], and *Noy and McGuinness, 2001* [22]. They all support semantic Web applications and assume RDF/XML notation rather than HTML or plain XML as the underlying Web standard. The referenced Protégé2000[1] is an open-source ontology editor developed at Stanford University and uses Java technology.

[1] Hereafter abbreviated Protégé as in http://protege.stanford.edu/

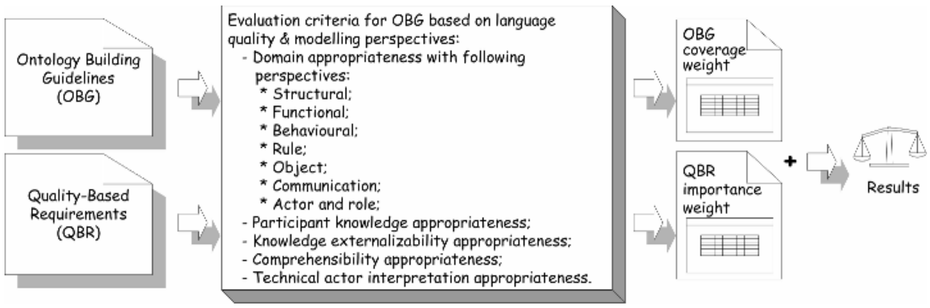

Ontology Building Guidelines (OBG)
Quality-Based Requirements (QBR)
Evaluation criteria for OBG based on language quality & modelling perspectives:
- Domain appropriateness with following perspectives:
* Structural;
* Functional;
* Behavioural;
* Rule;
* Object;
* Communication;
* Actor and role;
- Participant knowledge appropriateness;
- Knowledge externalizability appropriateness;
- Comprehensibility appropriateness;
- Technical actor interpretation appropriateness.
OBG coverage weight
QBR importance weight
+
Results

Fig. 3. The approach for the ontology method guidelines evaluation

Denker, 2003. It is a user's guide of the DAML+OIL plug-in for Protégé2000. The ontology building method is based on DAML+OIL language and Protégé as the ontology development tool. The ontology building process consists of three basic steps; create a new ontology, load existing ontologies, save ontology. The creation of new ontology consists of five types of instructions; define classes, properties (slots), instances, restrictions, and Boolean combinations.

Comment. The method does not contain any explicit description of the development process. However, the sequence of the sections in the documentation gives an indication of how to create an ontology.

Knublauch et al., 2003. It is a tutorial that was originally created for the 2nd International Semantic Web Conference. The ontology building method is based on OWL language and assumes Protégé as the ontology development tool. The ontology building process consists of seven iterative steps, namely determine scope, consider reuse, enumerate terms, define classes, define properties, create instances, and classify ontology.

Comment. The development activity requires some experience and foresight, communication between domain experts and developers, and a tool that is both comprehensible and powerful, including support for ontology evolution.

Noy and McGuinness, 2001. It is a general guide for building ontologies, called Ontology Development 101. The ontology building method is language and ontology development tool independent yet it uses Protégé in the examples. The ontology building process consists of seven iterative steps, namely determine the domain and scope of the ontology, consider reusing existing ontologies, enumerate important terms in the ontology, define the classes and the class hierarchy, define the properties of classes - slots, define the facets of the slots, and create instances.

Comment. The method set three rules for development decisions: 1) there is no single correct way to model a domain, 2) ontology development is necessarily an iterative process, and 3) concepts in the ontology should be close to objects, physical or logical, and to relationships in the domain of interest.

Table 1. Weighting method guidelines according to appropriateness categories

Criteria (*a*)	Category name	Coverage (*cw*)
Explanation		
DAML+OIL – Tutorial [5]		
a_1	**Domain appropriateness -** *DA*	
p$_1$ *Structural perspective* – Good description of the static structure of an ontology. The main construct is *thing* in the class hierarchy.		**2**
p$_2$ *Functional perspective* – No overview: no possibility to capture neither processes nor information roles.		**-1**
p$_3$ *Behavioural perspective* – Not intended to model the behaviour, no explanation of states or transitions included.		**-1**
p$_4$ *Rule perspective* – Partially: cardinality and static constraints.		**1**
p$_5$ *Object perspective* – DAML+OIL subclass inherits superclass' attributes.		**2**
p$_6$ *Communication perspective* -- Does not cover modelling of work processes or commitments between the participating actors.		**-1**
p$_7$ *Actor and role perspective* – Mentions neither actor nor role directly, but it describes properties, restrictions and instances.		**1**
a_2	**Participant knowledge appropriateness -** *PKA*	**1**
Introduces statements that are not in a participant's knowledge, unless he already has some experience. Acquaintance with a tool gives advantages. Lacks support for inexperienced users, e.g., the name for properties is called slots and not explained. Yet most part of the main concepts are intuitive. Different concepts have different visualizations.		
a_3	**Knowledge externalization appropriateness -** *KEA*	**1**
The guideline is domain independent, but it does not give guidance to all relevant usages. It lacks description of when to apply which construct, yet otherwise provides a good overview of constructs. Has some limitations to externalize complex knowledge.		
a_4	**Comprehensibility appropriateness –** *CA*	**1**
The described modelling constructs are easily distinguished from each other, yet examples of when to use which concept are missing. NL and graphical explanations and uniformity of phenomena are used. The number of constructs is reasonable or organized in a natural hierarchy. Proposed use of modelling constructs is uniform for all the statements. The guidance is flexible in the level of detail. The tutorial contributes to comprehensibility, yet not sufficiently.		
a_5	**Technical actor interpretation appropriateness** – *TAIA*	**-1**
None of the TAIA aspects are covered in the guideline.		
OWL- tutorial [14]		
a_1	**Domain appropriateness –** *DA*	
p$_1$ *Structural perspective* – Good description of the static structure of ontology. The main construct is thing in the class hierarchy.		**2**
p$_2$ *Functional perspective* – No overview: no possibility to capture neither processes nor information roles.		**-1**
p$_3$ *Behavioural perspective* – Not intended to model behaviour, no explanation of states or transitions included.		**-1**
p$_4$ *Rule perspective* – Partially: cardinality and static constraints. .		**1**
p$_5$ *Object perspective* – OWL subclass inherit superclass' attributes		**2**

Criteria (*a*)	Category name	Coverage (*cw*)
Explanation		
p$_6$ *Communication perspective* – Does not cover modelling of work processes or commitments between the participating actors.		**-1**
p$_7$ *Actor and role perspective* – Covers actor and role perspective to some degree. Applied as relations and individuals in combination.		**1**
a$_2$	**Participant knowledge appropriateness –** *PKA*	**1**
Introduces statements that are not in participant's knowledge, which can be a problem for inexperienced actors. The name for properties is slots, yet examples help inexperienced users. Most of the main concepts are intuitive. Non-intuitive representations could have been better described.		
a$_3$	**Knowledge externalization appropriateness** – *KEA*	**2**
The guideline is domain independent. Guidance to possible usages is given. It is describes extensively the use of classes, properties, individuals, mapping and descriptions. There are some concepts that are not possible to represent in the target language. Limitations to externalize complex knowledge.		
a$_4$	**Comprehensibility appropriateness** – *CA*	**2**
The described modelling constructs are easily distinguished from each other and there is uniformity of phenomena. The number of covered constructs is reasonable. Practical examples of when to use which concept are provided, including graphical explanations. The use of modelling constructs is uniform for all statements. The guidance is flexible in level of detail. Separation of concerns is supported, yet not multiple views.		
a$_5$	**Technical actor interpretation appropriateness** – *TAIA*	**-1**
None of the TAIA aspects are covered in the method guideline.		
Ontology development 101 [22]		
a$_1$	**Domain appropriateness** – *DA*	
p$_1$ *Structural perspective* – A good description of the static structure of an ontology: a formal explicit description of concepts in a domain of discourse. Constructs arranged in a natural hierarchy.		**2**
p$_2$ *Functional perspective* – No overview: no possibility to capture neither processes nor information roles.		**-1**
p$_3$ *Behavioural perspective* – Not intended to model behaviour, no explanation of states or transitions included.		**-1**
p$_4$ *Rule perspective* – Partially covered. Mentions how to use rule-based systems to extract information from ontology. Cardinality and static constraints are used. Rather than describing how to define rules in the ontology, the tutorial assumes the ontology to be used by external reasoning applications that create these rules.		**1**
p$_5$ *Object perspective* – The class-subclass phenomenon is described independent of language being different from classes and relations in object-oriented programming.		**2**
p$_6$ *Communication perspective* – Does not cover modelling of work processes or commitments between the participating actors.		**-1**
p$_7$ *Actor and role perspective* – Partially covered: Applied as relations between classes and instances of classes.		**1**
a$_2$	**Participant knowledge appropriateness** – *PKA*	**2**
Introduces some statements that may not be in a participant's knowledge. Non-intuitive concepts are introduced with a comprehensible description, often coupled with an illustration. Covers where in the process the domain knowledge is placed. Alternatives are		

Criteria (*a*)	Category name	Coverage (*cw*)
Explanation		
presented to some extent; participant can select a suitable representation. Eventually, the tool and language will narrow down the choices. The abstract methodology does not describe nor depend on any particular ontology language.		
a_3	**Knowledge externalization appropriateness** – *KEA*	**2**
The method guideline is domain independent. Extensive guidance to possible usages is given. Gives a systematic overview of the different ontology components. Good description of when and how to distinguish classes from properties. Some of the instructions might not be sufficient when a participant need to represent this information using a tool.		
a_4	**Comprehensibility appropriateness –** *CA*	**2**
The described modelling constructs are easily distinguished from each other, including illustrations. The number of constructs is reasonable. Use of modelling constructs is uniform for all statements. It is flexible in level of detail. Separation of concerns and multiple views are supported. Supports structured understanding of the ontology development process.		
a_1	**Technical actor interpretation appropriateness** – *TAIA*	**1**
Formal syntax and semantics and reasoning are partially covered.		

The general evaluation is summarized in Table 1. The table is in three parts, one for each method guideline, where the columns are the evaluation criteria id, name and coverage weight, and where every-other row describes in NL how the guideline covers the criteria. In overall, the guidelines satisfy *domain appropriateness (DA)* similarly, where none is complete in its coverage. This may be an indication of need for further analysis of the target ontology languages. "Ontology Development 101" is the most complete concerning *participant knowledge appropriateness (PKA)*, whereas "OWL- Protégé tutorial" is considered satisfactory concerning *knowledge externalization (KEA)* and *comprehensibility appropriateness (CA)*. Finally, "Ontology Development 101" is the only method guideline partially supporting *technical actor appropriateness (TAIA)*.

5 Guidelines for Ontology Building – The edi Case

The case study is based on edi (engaging, dynamic innovation) which is a system developed by a student project group [12]. edi is intended to support exchange of business ideas between the employees within the oil company, which is an integrated oil and gas company with business operations in 25 countries. At the end of 2002, there were 17 115 employees in the company. The amount of information and knowledge provided by the employees is rapidly increasing. Consequently, there is a need for more effective retrieval and sharing of knowledge. The role of edi is to provide a tool and encouragement for generating ideas, with the intention to enable the staff to focus on the most relevant activities. edi shall create a connection-point for communication and knowledge sharing between employees within various business areas and with various competence, e.g., between domain experts in oil-drilling and department managers. The plan is to utilize semantic Web technologies for that purpose. Ontologies will support common access to information and enable implementation of Web and ontology-based search. There will be participants of different quality and knowledge that are experts on creativity.

edi ***requirements.*** The overall functional requirements for edi have been analysed. Before the system can be developed a more thorough analysis needs to be conducted, however. Further, a decision about the purpose of the ontology has to be enforced. Information about the domain plays an important role in this process. It can be gathered in many ways and unavoidably, there will be many different participants involved in such a process; for instance end users, such as possible idea contributors or managers in the edi network that evaluate ideas. A requirements specification describes what the ontology should support, sketching the boundaries of the ontology application, and list valuable knowledge sources. Furthermore, oil industry as a business is in constant change, where globality makes the changes even more complex. In overall, edi needs to have a high durability, be adaptable to changes in the environment, be highly maintainable, and to have high reliability in order to secure the investment. A careful analysis needs to be made early in the process that places elaborate quality requirements on the ontology development environment.

Quality-based requirements. An ontology should be built in a way that support automatic reasoning and provide basis for high quality Web-based information services. An assumption is that a high quality engineering process assures high quality end product. The quality of ontology building process depends on the environmental circumstances under which the ontology is used. A model is expected to have high degree of quality if it describes complete set of steps and instructions for how to arrive at a model that is valid with respect to the domain and the language(s) it supports.

In the following, the quality requirements are categorized according to the appropriateness categories in the extended evaluation framework. Metrics is proposed for prioritising the appropriateness criteria adapting PORE methodology. The ontology building guidelines are evaluated in a particular situation, based on the above edi requirements. Importance of appropriateness category is calculated as follows.

Let *R(CF)* be a set of weighted requirements such that *R* has a fixed set *RA* of categories *ra*, where categories in *RA* correspond with each category *a* in *A* of an evaluation framework *EF*, i.e. *RA* = *A*, and *a* ∈ *A*, *ra* ∈ *RA*, and where *ra* is a triple <*id*, *descriptor*, *iw*>, where *id* is the name of the appropriateness requirement category, *descriptor* is a NL description of an appropriateness requirement, and *iw* defines a function of *I* that returns 0, 3, or 5 as importance weight (see Equation 6) based on priorities and policy of the company, where *I* is a set of importance-judged elements *ra* in the selection criteria *C* of each category in *RA*.

$$iw_{ra}(I) = \begin{cases} 0, & \text{if } ra \text{ may be satisfied, is optional,} \\ 3, & \text{if } ra \text{ should be satisfied, is recommended,} \\ 5, & \text{if } ra \text{ must be satisfied. is essential,} \end{cases} \quad (6)$$

The stakeholder prioritises the evaluation aspects according to the edi quality-based requirements, where an importance weight (0, 3 or 5) is assigned to each appropriateness category as in Equation 6. In table 2, the columns are the appropriateness requirement category id, the name and the importance weight, where every-other row represent a sub-column of a NL description of the requirement. In overall, it can be observed that *DA* in average and *TAIA* are the most important evaluation aspects for edi, whereas *CA* is the least important.

Table 2. Weighted edi requirements on the SQF appropriateness categories

Req. id	Requirement category name	Importance *(iw)*
Requirement description		
ra$_1$	**Domain appropriateness –** *DA*	
p$_1$ *Structural perspective* – Must be covered.		**5**
p$_2$ *Functional perspective* – edi need not support this perspective.		**0**
p$_3$ *Behavioural perspective* – It is recommended to support the behavioural perspective, e.g., owner of an idea should be able to update his contribution, i.e. change some state.		**3**
p$_4$ *Rule perspective* – Must be supported. This enables representation of constructs, such as a reply to the author if mandatory information is missing in the idea.		**5**
p$_5$ O*bject perspective* – Users of edi should be registered, i.e. a hierarchy of the different persons based on their position in the organisation. This is typically covered by the, and is done by creating classes and subclasses.		**3**
p$_6$ C*ommunication perspective* – The main focus of the is the actors cooperation within work processes through mutual commitments. For edi we do not focus on actors reaching mutual commitments, but rather automatic reasoning systems such as agents. Agents will play an important part of the system, and the communication perspective must to be covered to make this a realisation.		**5**
p$_7$ *Actor and role perspective* – It is important that an actor, such as a person that creates an idea, has a role, for example research officer. This is necessary to distinguish between an idea delivered by a person that has knowledge within the research area and those less familiar with the area. Hence, the actor and role perspective must be supported in the edi case.		**5**
ra$_2$	**Participant knowledge appropriateness –** *PKA*	**3**
The development of edi ontology will involve many participants which might have different language models. The goal of participant knowledge appropriateness is that all statements in the language models used by the participants are part of their explicit knowledge. It is not essential to accomplish this to a full extent, since less experienced users do not have to participate in the whole process. They can still contribute and pass on the unacquainted parts to the experienced users, thus increasing participant knowledge appropriateness. On the other hand it is possible to make less experienced participants contribute more by, for example offering relevant courses they can attend.		
ra$_3$	**Knowledge externalization appropriateness** – *KEA*	**3**
It will be very difficult for an inexperienced user to know how to externalize knowledge using a specific language. Therefore, a well written tutorial and perhaps some training most likely are preferable to increase the developers' knowledge externalizability appropriateness.		
ra$_4$	**Comprehensibility appropriateness –** *CA*	**0**
Comprehensibility appropriateness implies that all possible statements of the language are understood by the participants. No comprehensibility appropriateness is necessary to fulfil to a full degree in this case. This is caused by the fact that everyone part of the development process does not need to attend to the same degree. It is no problem that some people attending have more experience than others, as this does not have to affect the overall quality. One might use the knowledge of participants that have low understanding, assuming		

Req. id	**Requirement category name**	**Importance *(iw)***
Requirement description		
that there are others that have high degree of understanding. A language to support this is important. There are many ways to increase comprehensibility. Descriptions, graphics, etc are helpful to increase comprehensibility.		
ra_5	**Technical actor interpretation appropriateness -** *TAIA*	**5**
Technical actor interpretation appropriateness is a major requirement for the system. The ontology must be understandable for agents, thus enabling efficient reasoning of the content. This is one of the most important uses of the system.		

Recall the coverage weights (-1, 1 or 2) expressing how well the tutorials satisfy these evaluation criteria from Table 1. As the next step in the situated evaluation, the importance weights in Table 2 are multiplied by the coverage weights into a total weight *tw*. The individual products here are used as comprehensive feasibility rates that motivate and guide the edi stakeholder's assessment of method guideline. Thus the total weights express how well the guidelines satisfy each evaluation aspect.

Finally, a total coverage weight *tw* is calculated for each ontology building method guideline, as in Equation 7, where the sum of importance weights from Table 2 multiplied by the coverage weights from Table 1 is computed for all the categories. Table 3 shows the results of the situated evaluation of ontology building method guidelines based on their coverage of quality aspects and their importance for the edi application. The results are used to facilitate the edi stakeholder's final choice of ontology building method guidelines.

$$Tw_i = \sum_{a \in \text{A},\ \text{ra} \in \text{RA}} (cw_a \times iw_{ra}) \tag{7}$$

Table 3. Final evaluation of ontology building guidelines for edi case

Criteria (*a*)	Importance weight (*iw*)	DAML+OIL-Tutorial		OWL-Tutorial		Ontology development 101	
		Coverage weight (*cw*)	**Total (*tw*)**	Coverage weight (*cw*)	**Total (*tw*)**	Coverage weight (*cw*)	**Total (*tw*)**
DA-a_1p_1	5	2	**10**	2	**10**	2	**10**
DA-a_1p_2	0	-1	**0**	-1	**0**	-1	**0**
DA-a_1p_3	3	-1	**-3**	-1	**-3**	-1	**-3**
DA-a_1p_4	5	1	**5**	1	**5**	1	**5**
DA-a_1p_5	3	2	**6**	2	**6**	2	**6**
DA-a_1p_6	5	-1	**-5**	-1	**-5**	-1	**-5**
DA-a_1p_7	5	1	**5**	1	**5**	1	**5**
PKA-a_2	3	1	**3**	1	**3**	2	**6**
KEA-a_3	3	1	**3**	2	**6**	2	**6**
CA-a_4	0	1	**0**	2	**0**	2	**0**
TAIA-a_5	5	-1	**-5**	-1	**-5**	1	**5**
		Tw$\mathbf{Tw_{DAML+OIL\ Tutorial}}$**:**	**19**	$\mathbf{Tw_{OWL\ Tutorial}}$**:**	**22**	$\mathbf{Tw_{OntDev101}}$**:**	**35**

Apparently, some appropriateness levels help differentiating the guidelines better than others in the edi context. *PKA* and *TAIA* both highlight one guideline as the most feasible, whereas *KEA*, and *CA* highlight one as the least feasible. Together with *TAIA*, *DA* is the appropriateness that (in average) covers least of what is expected. Finally, even though the results in Table 3 indicate “Ontology Development 101” being the most appropriate, it is considered not to be sufficient for edi. The application is complex in nature, and therefore requires better coverage of *DA*, both in the target language and *TAIA* in the guideline itself.

6 Concluding Remarks

An evaluation of three different method guideline documents for construction of ontologies was conducted. The evaluation was instrumentalized using a framework previously proposed for evaluating modelling approaches in general, e.g. as used in [16] for evaluating UML. Evaluation was performed in two steps, one general evaluation (i.e., their applicability for building ontologies in general) and one for a particular case (i.e., how appropriate are these methods for ontology development within the edi project). The major results are as follows:

– The framework showed potential for supporting the selection of method guidelines. However, some adaptation was required in the interpretation of the various appropriateness-categories as defined in the [18] framework.
– In both steps, the method “Ontology Development 101” [22] came out on top. This was also the only method guideline, which was independent of the specific representation language.
– Major weaknesses were identified for all the methods, reflecting the current immaturity of the field of web-based ontology building.

The contribution of this paper is twofold: *First*, an existing evaluation framework was tried out with other evaluation-objects than it has been used for previously; *Second*, numerical values and metrics were incorporated to the evaluation framework for the quality factors and thus qualification of weighted selection was enabled. The industrial case study suggests the proposed evaluation of method guidelines is useful in selection, quality assurance, and engineering of ontology building guidelines.

The concrete ranking of methods may be of limited use, as new ontology languages and method guidelines are developed and the existing languages evolve and became more mature. Nevertheless, it can be useful in terms of guiding the current and future creators of such languages and their method guidelines. Drawing attention to the weakness of current proposals, these can be mended in future proposals, so that there will be higher quality languages and method guidelines to choose from in the future. The underlying assumption here is that high quality method guidelines may increase and widen the range and scalability of the semantic Web ontologies and applications.

The method of situated evaluation seems promising, yet a single case only indicates applicability. In other words we have not shown general applicability nor given substantial evidence of the feasibility of the metrics or of the situated evaluation. An attempt to specialise the evaluation was also made in [17] using a single case. The general evaluation framework is there used as theoretical background for a feature analysis rather than as an observation instrument as here. Our study is an attempt to situate the evaluation so that it supports re-use, simulation, and maintenance as well as specialisation. As mentioned, the pragmatic merits of the proposed metrics and method still remains to be substantiated.

There are other interesting topics for future work, such as supplementing the theoretical evaluations with empirical ones as larger scale semantic Web applications arise utilizing the empirical nature of [18], as well as evaluating more methods as they emerge. Further possibilities are in investigating the appropriateness of the formalisation quality criteria in the [30] Unified methodology as a complement to the semiotic quality framework in order to conduct evaluation of the process oriented methodologies that were out of scope of this study.

References

1. Becker, J., Rosemann, M., von Uthmann, C. Guidelines of Business Process Modeling. In W. Aalst, J. Desel, A. Oberweis (Eds.): *Business Process Management: Models, Techniques and Empirical Studies*. LNCS 1806, Springer-Verlag, Berlin. pp. 30-49, 2000.
2. Berners-Lee T, Handler J, Lassila O. The Semantic Web. *Scientific American*, May 2001.
3. Corcho, O., Fernández-López, M., Gómez-Pérez, A. Methodologies, tools and languages for building ontologies: where is their meeting point?, *Data and Knowledge Engineering*, 46(1) pp. 41 – 64, 2003.
4. Davies, I., Green, P., Milton, S., Rosemann, M. Using Meta-Models for the Comparison of Ontologies. In *Proceedings of the 8th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in System Analysis and Design* (EMMSAD'03), Velden, Austria, June 16-17, 2003.
5. Denker, G.: DAML+OIL Plug-in for Protégé 2000 – User's Guide. SRI International AI Center Report 7/8/03, 2003.
6. Falkenberg, E.D., Hesse, W., Lindgreen, P., Nilsson, B.E., Han Oei, J.L., Rolland, C., Stamper, R.K., van Asche, F.J.M., Verrjin-Stuart, A., Voss, K. FRISCO - A Framework of Information Systems Concepts. IFIP WG 8.1 Technical Report, 1997.
7. Fernández-López, M. Overview of Methodologies for Building Ontologies. In Benjamins, V.R., Chandrasekaran, B., Gómez-Pérez, A., Guarino, N., Uschold, M. (Eds.): *Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods* (KRR5) Stockholm, Sweden, 1999.
8. Fernándes-Lopez, M., Gómez-Péres, A., Juriso, N. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *Proceedings of AAAI-97 Spring Symposium on Ontological Engineering*. Stanford University, 1997.
9. Gemino, A., Wand, Y. Evaluating modelling techniques based on models of learning. *Communications of the ACM* 46(10), pp. 79-84, 2003.
10. Gómez-Péres, A., Corcho, O. Ontology Languages for the Semantic Web. *IEEE Intelligent Systems*, pp. 54-60, 2002.
11. Gruber, T.R. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2), pp. 199-220, 1993.
12. Hakkarainen, S., Hella, L., Tuxen, S.M., Sindre, G. Evaluating the quality of web-based ontology building methods: a framework and a case study. In Barzdins, J. (ed.): *Proceedings of 6th International Baltic Conference on Databases and Information Systems* (Baltic DBIS'04), volume 672 of CSIT, pp. 451-466. University of Latvia, Riga, Latvia, 2004.

13. Hakkarainen, S., Strasunskas, D., Hella, L., and Tuxen, S.M.. Classification of web-based ontology building method guidelines: a case study. In *Proceedings of the 10th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design* (EMMSAD'05), June 2005.
14. Knublauch, H, Musen, M.A., Noy, N.F. Creating Semantic Web (OWL) Ontologies with Protégé. Tutorial at 2nd Intl. Semantic Web Conf., Sanibel Island Florida USA. October 20-23, 2003.
15. Krogstie, J. Using a Semiotic Framework to Evaluate UML for the Development of Models of High Quality. In Siau, K. and Halpin, T. (eds.): *Unified Modeling Language: Systems analysis, design, and development issues*. IDEA Group Publishing, 2001.
16. Krogstie, J. Evaluating UML Using a Generic Quality Framework. In Favre L (ed.): *UML and the Unified Process*. IDEA Group Publishing, 2003.
17. Krogstie, J.and de Flon Arnesen, S. Assessing enterprise modeling languages using a generic quality framework. In J. Krogstie, T. Halpin, and K. Siau (eds.), *Information Modeling Methods and Methodologies*, number 1537-9299 in ATDR, chapter IV, pp. 63–79. IDEA Group Publishing, 2004.
18. Krogstie J, Sølvberg A.. *Information Systems Engineering – Conceptual modelling in a quality perspective*, Kompendiumforlaget, Trondheim, Norway, (2003).
19. Lindland, O.I., Sindre, G., Sølvberg, A. Understanding Quality in Conceptual Modeling. *IEEE Software*, 11(2), pp. 42-49, 1994
20. Maiden, N.A.M., Ncube, C.: Acquiring COTS Software Selection Requirements. *IEEE Software*, 15(2), pp 46-56, 1998.
21. Moody, D.L., Shanks, G.G., Darke, P. Evaluating and Improving the Quality of Entity Relationship Models: Experiences in Research and Practice. In T. Wang Ling, S. Ram, and M.-L. Lee (eds.): *Proceedings of 17th International Conference on Conceptual Modelling* (ER '98), LNCS 1507, Springer-Verlag, Singapore, 1998.
22. Noy, N.F., McGuinness, D.L. Ontology Development 101: A Guide to Creating Your First Ontology'. Stanford Knowledge Systems Laboratory Technical Report KSL-01-05, March 2001.
23. Opdahl, A.L. and Henderson-Sellers, B. Ontological evaluation of the UML using the Bunge-Wand-Weber model. *Software and Systems Modelling* (SoSyM) 1(1), pp. 43-67. Springer, 2002.
24. Pohl, K. Three dimensions of requirements engineering: a framework and its applications. *Information Systems* 19(3), pp. 243-258, 1994.
25. Schuette, R. Architectures for evaluating the quality of information models – a meta and an object level comparison. In J. Akoka, M. Bouzeghoub, I. Comyn-Wattiau, and E. Métais (eds.): *Proceedings of the 18th International Conference on Conceptual Modelling* (ER'99), Paris, France, LNCS 1728, Springer-Verlag, 1999.
26. Strasunskas, D. and Hakkarainen, S. Process of Product Fragments Management in Distributed Development. In R. Meersman, Z. Tari, D. Schmidt et al (Eds.): *Proceedings of the 11th International Conference on Cooperative Information Systems* (CoopIS'2003), Springer-Verlag, LNCS2888, pp. 218-234, 2003.
27. Su, X. and Ilebrekke, L. Using a semiotic framework for a comparative study of ontology languages and tools. In J. Krogstie, T. Halpin, and K. Siau (eds.): I*nformation Modeling Methods and Methodologies*, number 1537-9299 in ATDR, chapter XIV, pages 278–299. IDEA Group Publishing, 2004.
28. Sure, Y., Studer, R. On-To.Knowledge Methodology – Final Version. Institute AIFB, University of Karlsruhe. September 26, 2002.

29. Swartout, B., Ramesh P., Knight, K., Russ, T.: Toward Distributed Use of Large-Scale Ontologies. In *Proceedings of Symposium on Ontological Engineering of AAAI*. Stanford, California, 1997.
30. Uschold, M. Building Ontologies: Towards a Unified methodology. In *Proceedings of the 16th Conf. of the British Computer Society Specialist Group on Expert Systems*. Cambridge, UK, 1996.
31. Uschold, M., Gruninger, M. Ontologies: Principles, methods and applications. *Knowledge Engineering Review*, 11(2), pp. 93-155, 1996.
32. Wand, Y., Weber, R. Mario Bunge's Ontology as a formal foundation for information systems concepts. In: Weingartner, P. and Dorn, G. (eds.): *Studies on Mario Bunge's Treatise*, Rodopi, Atlanta, 1990.
33. Weber, R., Zhang, Y. An analytical evaluation of NIAM's grammar for conceptual schema diagrams. *Information Systems Journal* 6(2), pp. 147-170, 1996.

Conceptual Model Based Semantic Web Services

Muhammed Al-Muhammed[1], David W. Embley[1], and Stephen W. Liddle[2]

[1] Department of Computer Science
[2] Rollins Center for eBusiness, Brigham Young University, Provo, Utah 84602, U.S.A
{mja47, embley}@cs.byu.edu, liddle@byu.edu

Abstract. To achieve the dream of the semantic web, it must be possible for ordinary users to invoke services. Exactly how to turn this dream into reality is a challenging opportunity and an interesting research problem. It is clear that users need simple-to-invoke-and-use services. This paper shows that an approach strongly based on conceptual modeling can meet this challenge for a particular type of service—those that involve establishing an agreed-upon relationship, such as making an appointment, setting up a meeting, selling and purchasing products, or establishing employee job assignments. For these services, users can specify their requests as free-form text and then interact with the system in a simple way to complete the specification of a service request, if necessary, and invoke the service. Our system uses a conceptual-model-based information extraction ontology to (1) recognize the request and match it with an appropriate ontology, (2) discover and obtain missing information, and (3) establish agreed-upon, conceptual-model-constrained relationships with respect to the desired service. The paper lays out our vision for this type of semantic web service, gives the status of our prototype implementation, and explains how and why it works.

Keywords: Services, semantic web services, service specification, service invocation, conceptual-model-based services.

1 Introduction

In open and ever-growing environments such as the world wide web, the amount of information and the number of services becoming available makes performing tasks such as finding information and finding and invoking services of interest quite challenging for web users. The semantic web along with personal software agents and web service systems purport to offer a solution to this challenge. But exactly how this solution will play out is still unclear.

In this paper we offer a unique approach to turn the vision of semantic web pioneers (e.g. [1]) into reality for everyday tasks such as scheduling appointments, selling, buying, making job assignments, and so forth. Our approach to this challenge centers around a *task ontology.* A task ontology can be thought of as having two component ontologies: (1) a *domain ontology* that defines concepts in a domain of a task along with relationships among these concepts and (2) a *process ontology* that defines generic processes for doing tasks. With a task ontology in hand, we address the following fundamental problems.

1. *Task Specification.* The first key issue is how to allow users to request services. We intend to let users assume the existence of an intelligent agent within the system and specify their service requests textually in any way they wish.
2. *Task Recognition.* Given a service request, our system attempts to recognize the specified task. This recognition process extracts information from the service request and matches it against known domain ontologies to find the proper task ontology for the service request.
3. *Task Execution.* Given a recognized task ontology and the information extracted from a service request, the system specializes the process ontology within the recognized task ontology to perform the service. The specialized process ontology has the ability to gather the information it needs by interacting with the system or the user or both to obtain missing required information. The specialized process ontology also has the ability to recognize specified constraints and determine whether they are satisfied. There may be a need to negotiate with users to relax task constraints when it is apparent that it is not possible to complete the task given the current constraints. Finally, the specialized process ontology has the ability to establish the necessary relationships to complete the service request.

Our contribution in this paper is to show that it is possible to build a system to automate everyday tasks, such as scheduling appointments, buying and selling, and making job assignments, whose invocation results in establishing agreed-upon relationships in a domain ontology. Within this scope of services, the system's behavior is limited only by the richness of task ontologies, which can be independently enriched by system specialists. Our approach contributes to the vision of the semantic web in the sense that it offers the following significant advantages. (1) The system allows for free-form task specification, and thus, it does not impose any programming paradigm to specify tasks, nor does it have a set of pre-specified tasks from which a user can choose. (2) The system reasons about the request based on a task ontology and synergistically gathers the information it needs to generate software capable of performing the task.

We present our vision for task-ontology-based services in three major parts: task specification, which allows users to textually specify tasks (Section 2); task recognition, which finds the domain of a specified task and specializes processes to perform the task (Section 3); and task execution (Section 4). We give the status of our prototype implementation along with future work we are considering in Section 5, and we conclude with Section 6.

2 Task Specification

We explain task specification using an example. A typical usage of our approach is to schedule appointments. We use a somewhat simplified version of the example described by Berners-Lee, et al., in their vision paper, "The Semantic Web" [1]. In our example, a user of the semantic web wants to schedule an appointment with a service provider—a dermatologist. The user does not have any particular dermatologist in mind, but wants one that meets some constraints regarding appointment time, date, the location of the service provider, and the type of insurance the service provider accepts.

To use our approach to accomplish this task, the user first specifies the task by "simply" stating what needs to be done. Suppose the user states the following.

> I want to see a dermatologist next week; any day would be OK for me, at 4:00. The dermatologist should be within 5 miles from my home and must accept my insurance.

Before this statement is made, our proposed system has no clue regarding the domain of the task nor any clue regarding how it can be done. Therefore, this specification needs to go through a task recognition step, which we discuss next.

3 Task Recognition

The objective of task recognition is to determine the domain of a specified task. Our approach uses a task ontology to meet this objective. Therefore, we first introduce the two components of a task ontology, namely a *domain ontology* and a *process ontology*, respectively introduced in Sections 3.1 and 3.2. In Section 3.3, we describe how our approach determines which task ontology to use, among the many we assume exist.

3.1 Domain Ontology

A *domain ontology* specifies named sets of objects, which we call *object sets* or *concepts*, and named sets of relationships among object sets, which we call *relationship sets*. Figure 1 shows a small part of a conceptual model representation of a domain ontology for scheduling an appointment.[1] The domain ontology consists of concepts such as *Date*, *Time*, and *Service Provider* that can be used to schedule appointments with service providers such as doctors and auto mechanics. The conceptual model has two types of concepts, namely lexical concepts (enclosed in dashed rectangles) and nonlexical concepts (enclosed in solid rectangles); it also provides for explicit concept instances (denoted as large black dots). A concept is *lexical* if its instances are indistinguishable from their representations. *Time* is an example of a lexical concept because its instances (e.g. "10:00 a.m." and "2:00 p.m.") represent themselves. A concept is *nonlexical* if its instances are object identifiers, which represent real-world objects. *Dermatologist* is an example of a nonlexical concept because its instances are identifiers such as, say, "D_1", which represents a particular person in the real world who is a dermatologist. We designate the main concept in a domain ontology by marking it with "–> •" in the upper right corner. We designate the concept

[1] In practice, we would need a much larger and richer ontology for service providers. We have limited our ontology to those concepts needed in our running example, plus a few more to explicitly illustrate concepts not needed for our sample task. We indicate by having *Auto Mechanic* in our example, for instance, that there are many more types of service providers.

Appointment in Figure 1 as the main concept because this domain ontology is for making appointments. Figure 1 also shows relationship sets among concepts, represented by connecting lines, such as *Appointment is on Date*. The arrow connections represent functional relationship sets, from domain to range, and non-arrow connections represent many-many relationship sets. For example, *Service Provider has Name* is functional from *Service Provider* to *Name* (i.e. a service provider has only one name), and *Service Provider provides Service* is many-many (i.e. a service provider can provide many services and a service can be provided by many service providers). A small circle near the connection between an object set O and a relationship set R represents optional, so that an instance of O need not participate in a relationship in R. For example, the circle on the *Appointment* side of the relationship set *Appointment has Duration* states that an instance of *Appointment* may or may not relate to an instance of *Duration* (i.e. there need not be a specified duration for an appointment). A triangle in Figure 1 defines a generalization/specialization with a generalization connected to the apex of the triangle and a specialization connected to its base. For example, *Dermatologist* is a specialization of *Doctor*.

The concepts in our domain ontology are augmented with data frames [2]. A *data frame* describes the information about a concept. We capture the information about a concept in terms of its external and internal representation, its contextual keywords or phrases that may indicate the presence of an instance of the

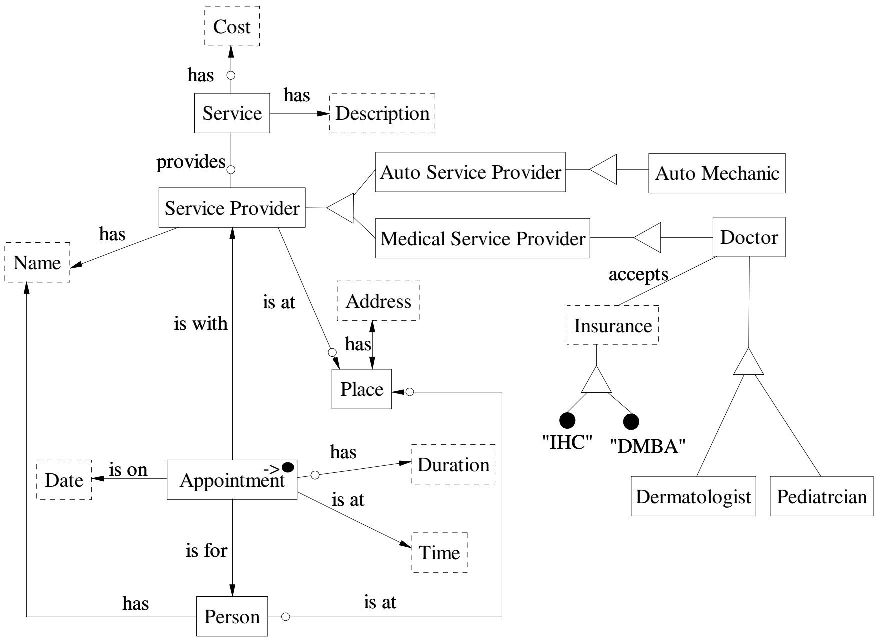


Fig. 1. Conceptual-model view of a domain ontology for appointments (partial)

concept, operations that convert between internal and external representations, and other manipulation operations that can apply to instances of the concept along with contextual keywords or phrases that indicate the applicability of an operation. Figure 2 shows sample (partial) data frames for the concepts *Time*, *Date*, *Address*, *Distance*, *Dermatologist*, and *Appointment*. The *Time* data frame, for example, captures instances of this concept that end with "AM" or "PM" (e.g. "2:00 PM" and "2:00 p.m"). As Figure 2 shows, we use regular expressions to capture external representations. A data frame's context keywords/phrases are also regular expressions (often simple lists of keywords/phrases separated with "|"). For example, the *Distance* data frame in Figure 2 includes context keywords such as "miles" or "kilometers". In the context of one of these keywords, if a number appears, it is likely that this number is a distance. The operations of a data frame can manipulate a concept's instances. For example, the *Distance*

```
Time
…
textual representation: ([2-9]|1[012]?):([0-5]\d)[aApP]\.?[mM]\.? | …
…
end

Date
…
NextWeek(d: Date)                          Tomorrow (s: String)
returns (Boolean)                          returns (Date)
context keywords/phrases: next week |      context keywords/phrases: tomorrow | next day | …
  week from now | …                        …
end                                        end

Address
…
DistanceBetween (a1: Address, a2: Address)
returns (Distance)
…
end

Distance
internal representation : real
textual representation: ((\d+(\.\d+)?)|(\.\d+))
context keywords/phrases: miles | mile | mi | kilometers | kilometer | meters | meter | …
…
LessThan(d1: Distance, d2: Distance)       LessThanOrEqual(d1: Distance, d2: Distance)
returns (Boolean)                          returns (Boolean)
context keywords/phrases: less than | < | … context keywords/phrases: within | not more than |
…                                            ≤ | …
end                                        …
                                           end

Dermatologist                                     Appointment
internal representation: object id                internal representation: object id
…                                                 …
context keywords/phrases: [Dd]ermatologist | skin context keywords/phrases: appointment |
  doctor | …                                        want to see a[n]? | …
…                                                 …
end                                               end
```

Fig. 2. Some sample data frames (partial)

data frame includes the operation *LessThan* that takes two instances of *Distance* and returns a Boolean. The context keywords/phrases of an operation indicate an operation's applicability, for example, context keywords/phrases such as "less than" and "<" apply to the *LessThan* operation. A nonlexical concept such as *Dermatologist* often only has context keywords or phrases. Figure 2 shows that the *Dermatologist* data frame includes a regular expression which includes words and phrases that could indicate the presence of the concept of a dermatologist.

3.2 Process Ontology

A *process ontology* describes an execution pattern in a domain. Figure 3 shows our process ontology as specialized for scheduling appointments. We represent process ontologies and specialized process ontologies as statenets [3], a represen-

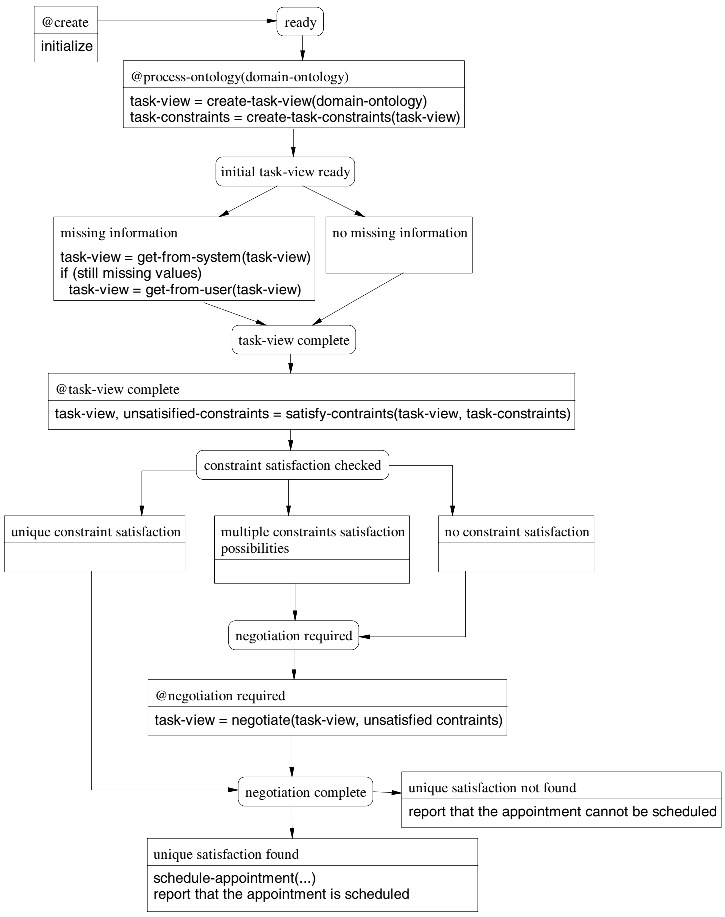

@create
initialize
ready
@process-ontology(domain-ontology)
task-view = create-task-view(domain-ontology)
task-constraints = create-task-constraints(task-view)
initial task-view ready
missing information
task-view = get-from-system(task-view)
if (still missing values)
task-view = get-from-user(task-view)
no missing information
task-view complete
@task-view complete
task-view, unsatisified-constraints = satisfy-contraints(task-view, task-constraints)
constraint satisfaction checked
unique constraint satisfaction
multiple constraints satisfaction possibilities
no constraint satisfaction
negotiation required
@negotiation required
task-view = negotiate(task-view, unsatisfied contraints)
negotiation complete
unique satisfaction not found
report that the appointment cannot be scheduled
unique satisfaction found
schedule-appointment(...)
report that the appointment is scheduled

Fig. 3. Process ontology specialized for scheduling appointments

tation that lets us specify standard Event-Condition-Action (ECA) rules [4,5]. The statenet in Figure 3 is "specialized" from the general pattern in the sense that (1) all but the two final actions (schedule and do not schedule) are parameterized by the domain ontology but otherwise fixed over all services for which the system operates; and (2) given the domain ontology, the system can fully generate these two final actions. Thus, any specialized process ontology depends only on the domain ontology. Significantly (and somewhat surprisingly), this means that system developers need never write code for services of the type our system handles; specifying domain ontologies is sufficient to fully specify the services.

In this section, we describe the ECA rules used to construct a process ontology and the execution pattern for the process ontology. We leave the details of the subprocesses on which the process ontology depends to be discussed in Section 4. As we shall see, all of these subprocesses are domain-independent. Domain-independence is what makes it possible to automatically generate specialized process ontologies without having to write any code.

A process ontology consists of states, represented as rounded rectangles (e.g. *ready* and *initial task-view ready* in Figure 3), and transitions, represented as divided rectangles (e.g. the *@create/initialize* transition in Figure 3). In the top part of a transition, we specify triggers, which are events or conditions or both. Events are prefixed by "@" (read "at"); examples include *@process-ontology(domain-ontology)* and *@task-view complete*, where the former is a parameterized event that triggers the transition when the event occurs, and the latter is a non-parameterized event that triggers the transition when the task view is complete. Actions appear in the bottom part of divided rectangles. The actions in a particular transition execute when the trigger of the transition fires. Examples include *create-task-view(domain-ontology)* and *get-from-system(task-view)*, which both invoke subprocesses in our system.

The general flow of the process ontology is as follows. Once triggered and given a domain ontology, the process ontology invokes the subprocess *create-task-view(domain-ontology)* to create a *task-view*, which is the part of a domain ontology that matches with the user-specified task, and then invokes the subprocess *create-task-constraints(task-view)* to find and list the applicable constraints for the task. If all concepts in the task view that are required to have values have already obtained their values from the user-given task specification, the process ontology enters the *task-view complete* state; otherwise the process ontology obtains values for these concepts from system repositories using the subprocess *get-from-system(task-view)* and obtains values from the user it cannot obtain from system repositories using the subprocess *get-from-user(task-view).* Next, the process ontology checks for constraint satisfaction using the process *satisfy-constraints(task-view, task-constraints)* and enters the *constraint satisfaction checked* state. If constraint satisfaction is unique (exactly one set of values satisfies the constraints), no negotiation is necessary, so the process enters the *negotiation complete* state; otherwise if there are multiple sets of values or if there are no sets of values that satisfy the constraints, the process ontology enters the *negotiation required* state. During the negotiation phase, the system and user work together in an attempt to find a unique solution. If a unique solution is found, the process ontology schedules the appointment; otherwise the process ontology reports that the appointment cannot be scheduled.

3.3 Task Ontology Recognition

The task ontology recognition process selects from among potentially many domain ontologies deployed on the semantic web the (correct) domain ontology for a task specification. The recognition process takes the set of available domain ontologies and a task specification as input, and returns the domain ontology that best matches with the task specification as output. The recognition process works in two steps. First, for each domain ontology, the recognition process applies concept recognizers in the data frames to the task specification and marks every concept that matches a substring in the task specification. Second, the process computes a rank value for each domain ontology with respect to the task specification and then selects the domain ontology that ranks highest.

When the recognition process executes for the domain ontology in Figure 1, the data frames in Figure 2, and the task specification in Section 2, it produces as output a marked-up ontology. In the case of our running example, the system marks *Appointment*, *Date*, *Time*, *Place*, *Insurance*, and *Dermatologist* in Figure 1. The concept recognizer in the data frame for *Dermatologist* recognizes the constant value "dermatologist" in the task specification, and therefore the concept *Dermatologist* is marked. Likewise, a recognizer in the *NextWeek* operation in the *Date* data frame recognizes "next week"; in the *Time* data frame recognizes the constant value "4:00"; in the *Appointment* data frame recognizes "want to see a"; in the *Place* data frame recognizes "my home"; and would, in the *Insurance* data frame, recognize "insurance"; and therefore these concepts are marked. The recognized substrings cover a large part of the task specification; for our running example, we assume that no other ontology covers the task specification as well and therefore that the system selects the *Appointment* task ontology.

4 Task Execution

The process ontology is responsible for executing tasks. As mentioned in Section 3.2, the process ontology invokes subprocesses that either execute the same for all domain ontologies or can be automatically generated from any given domain ontology. In this section we discuss these subprocesses and justify our claim that all code needed to execute any service can be fixed in advance or automatically generated.

4.1 Task View Creation

Task view creation takes a marked domain ontology as input and produces a task view as output. Although not quite so simple because spurious object sets may be marked, the process basically operates on its input as follows. It keeps

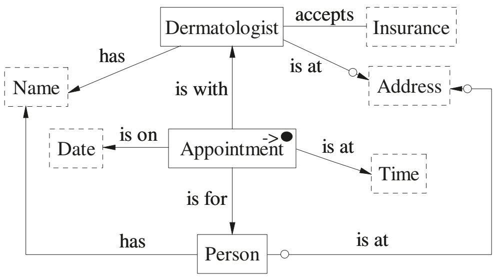


Fig. 4. Task view for the task specified in Section 2

the main concept of the domain ontology (the concept marked with "–> •"), all marked concepts, and all concepts that mandatorily depend on the main concept. It prunes away all other concepts along with all their relationships as well as any marked concepts considered to be spurious because they conflict with other marked concepts in the sense that constraints of the ontology do not allow both, and these other marked concepts rank higher in applicability to the task specification. In addition, the process replaces generalization concepts by marked specializations, if any, and replaces non-lexical concepts by lexical concepts when there is a one-to-one correspondence. The derived sub-ontology, consisting of the concepts and relationships among the concepts that remain, is called the *task view*. Observe that this process is domain independent—it operates identically for any defined domain ontology.

Referring to our running example, the resulting task view is in Figure 4. The process does not prune *Appointment* because it is the main concept. It does not prune *Date*, *Time*, *Place*, *Insurance*, and *Dermatologist* because they are marked, and it does not prune *Person*, *Name*, and *Service Provider* because they mandatorily depend on the main concept. Finally, the marked specialization *Dermatologist* replaces its generalization *Service Provider*, and the lexical concept *Address* replaces the non-lexical concept *Place* with which it has a one-to-one correspondence.

4.2 Task Constraint Creation

Given the task view and the operations in the data frames associated with the concepts of a task view, the system generates any additional constraints imposed on a task beyond those that are already part of the conceptual-modeling constraints of the task view. It then combines them with the constraints in the task view to produce the full set of constraints. The result is a formal statement in terms of predicate calculus that must be satisfied in order to service the request.

Task constraint creation operates as follows.

1. *Get the Boolean operations implied by the task specification.* The task-constraint-creation process finds all operations in the data frames whose recognizers match substrings in the task specification and whose return types are Boolean. In our running example, the process finds the operator *NextWeek(d: Date)* because, as Figure 2 shows, it is Boolean and one of its

context phrases is "next week", which appears in the task specification in Section 2. Similarly, the process finds *LessThanOrEqual(d1: Distance, d2: Distance)* based on recognizing "within", "5", and "miles" and recognizes *Equal(i1: Insurance, i2: Insurance)* based on recognizing "insurance".

2. *Get constant values from the task specification that can serve as parameter values of the Boolean operations.* The data frames recognize and extract "5" as a *Distance* and "4:00" as a *Time*. In our running example, we thus obtain *LessThanOrEqual(d1: Distance, "5")* and *Time("4:00")*. [2]
3. *Get operations that depend on the task view and can provide values for parameters of the Boolean operations.* For each Boolean operation, the process considers the type(s) of the input parameter(s). If one or more input parameters has a type that does not match a concept in the task view, the process tries to find an operation in the data frames whose input parameter types are concepts in the task view and whose return type matches the type of the input parameter; if successful, it replaces the input parameter with this operation. Referring to our example, *LessThanOrEqual(d1: Distance, "5")* has the input *d1* of type *Distance*, which does not appear in the task view. Since, however, *Address* does appear in the task view and the operation *DistanceBetween(a1: Address, a2: Address)* returns a *Distance*, the task-constraint-creation process can do a substitution, yielding *LessThanOrEqual(DistanceBetween(a1: Address, a2: Address), "5")*.
4. *Get sources, within the task view, for values of Boolean operations.* To determine the source of values for the input parameters of the Boolean operations, the process makes use of the relationships in the task view. For example, the operation *DistanceBetween(a1: Address, a2: Address)* has two input parameters of type *Address*. According to the relationships between the concepts in the task view, *Address* is related to both *Dermatologist* and *Person*. The process can therefore infer that the value of one of the address parameters comes from a relationship in *Dermatologist is at Address* and value of the other comes from a relationship in *Person is at Address*. The process leaves any input parameter that it cannot determine as a free variable. Because *Insurance* is related only to *Dermatologist*, the process determines that the source of the value of one input parameter comes from a relationship in *Dermatologist accepts Insurance* and leaves the other as a free variable. Also, since the relationship set *Dermatologist accepts Insurance* is many-many, the process binds the parameter *i2* with the existential quantifier to declare that any one value of *i2* that satisfies the generated predicate calculus statement $\exists i_2(Dermatologist(x)acceptsInsurance(i2) \wedge Equal(i1, i2)) \wedge Insurance(i1) \wedge Insurance(i2))$ is enough.

Figure 5 shows the resulting predicate calculus statement. Our objective, as we continue, will be to provide values for free variables such that there is one and only one appointment (i.e. one and only one value for the non-lexical argument x_0 in Figure 5). Before continuing, however, observe that the process

[2] Note that *Time(x)* is a one-place predicate. Every object set in a domain ontology is a one-place predicate, and every n-ary relationship set is an n-place predicate.

$$
\begin{aligned}
&Appointment(x_0)\, is\, with\, Dermatologist(x_1) \wedge Appointment(x_0)\, is\, for\, Person(x_2)\\
&\wedge Appointment(x_0)\, is\, on\, Date(x_3) \wedge Appointment(x_0)\, is\, at\, Time(\text{“4:00”})\\
&\wedge Dermatologist(x_1)\, has\, Name(x_4) \wedge Dermatologist(x_1)\, is\, at\, Address(x_5)\\
&\wedge \exists x(Dermatologist(x_1)\, accepts\, Insurance(x) \wedge Insurance(x_6) \wedge Equal(x, x_6))\\
&\wedge Person(x_2)\, has\, Name(x_7) \wedge Person(x_2)\, is\, at\, Address(x_8)\\
&\wedge \ldots\\
&\wedge NextWeek(x_3) \wedge LessThanOrEqual(DistanceBetween(x_5, x_8),\ \text{“5”})\\
&\wedge \forall x \forall y(Person(x)\, is\, at\, Address(y) \Rightarrow Person(x) \wedge Address(y))\\
&\wedge \forall x(Person(x) \Rightarrow \exists^{\leq 1} y(Person(x)\, is\, at\, Address(y)))\\
&\wedge \ldots
\end{aligned}
$$

Fig. 5. Generated predicate calculus statement (partial)

that generates the predicate calculus statement is domain independent because its algorithms are the same for all domains. The process makes use of only the information provided by the task view and the associated data frames. Once these are available, the process can discover constraints and produce a predicate calculus statement using fixed algorithms that work for all domains.

4.3 Obtaining Information from the System

Given the predicate calculus statement in Figure 5, the system can generate a query for the system's appointment databases. Assuming that the appointment database has a view definition that corresponds with its ontology, the generation of a relational calculus query is straightforward. We simply cut the predicate calculus statement down to include only those relationship sets that appear in the database's ontological view. We also combine the relationship sets from the database's ontological view that are connected to the primary object set. For our running example, the generated query is in Figure 6. In Figure 6 *Appointment is with Dermatologist and is on Date and is at Time* is the relationship set obtained by combining *Appointment is with Dermatologist*, *Appointment is on Date*, and *Appointment is at Time*; note that we do not also combine *Appointment is for Person* because this relationship set is not part of the stored appointment database for making appointments—only available appointment dates and times are in the database. Observe that given a predicate calculus statement generated by the task-constraint-creation process and the ontological view of the selected task ontology, the system can always generate this query; thus, this query generation process is domain independent.

Execution of the generated query returns a set of partially filled-in interpretations using the information extracted by the system. For example, the system

$$
\begin{aligned}
&\{< x_1, x_3, x_4, x_5, x > \mid\\
&\quad Appointment\, is\, with\, Dermatologist(x_1)\, and\, is\, on\, Date(x_3)\, and\, is\, at\, Time(\text{“4:00”})\\
&\quad \wedge Dermatologist(x_1)\, has\, Name(x_4) \wedge Dermatologist(x_1)\, is\, at\, Address(x_5)\\
&\quad \wedge Dermatologist(x_1)\, accepts\, Insurance(x)\}
\end{aligned}
$$

Fig. 6. Generated predicate calculus statement

might substitute D_0 for x_1 in Figure 5, *"5 Jan 05"* for x_2, *"Dr. Carter"* for x_4, and so on, depending on the actual information extracted from relevant sources. The meaning of this interpretation is that dermatologist D_0, who is *Dr. Carter* has an appointment available on *5 Jan 05*, which is "next week" with respect to our assumed execution date, *30 Dec 04*.

4.4 Obtaining Information from a User

After generating partial interpretations, there will still be free variables that need to be handled. In our example, the remaining free variables are x_0, which is the object we are trying to establish; x_2, which is the person for whom the appointment is being made; x_6, which is the insurance in the request; x_7, which is the name of the person for whom the appointment is being made; and x_8, which is the address of the person for whom the appointment is being made. We can establish the variable x_0, which represents the appointment, if we can obtain the remaining variables, which, of course, are exactly the ones we need to obtain from the user.

When the system can recognize which free variables need values (equivalently, which concepts need values), the process of obtaining values from the user is domain independent. Thus, if a lexical concept C requires a value from the user, the system can prompt the user with the standard phrase, "What is the C?" For nonlexical concepts, the system can generate object identifiers, as needed. For our running example, the system would ask: (1) "What is the Insurance?", (2) "What is the Name?", and (3) "What is the Address?". These may well be understood in the context of the task being specified, but it is likely to be better if the system can ask questions in context by verbalizing[3] the context with respect to the primary concept. In this case, for (2) the system would say, "Appointment is for Person, and Person has Name. What is the Name?" and for (3) would say, "Appointment is for Person, and Person has Address. What is the Address?". There is no context for the insurance other than the context established in the statement of the task specification in Section 2, so for (1) the system would just say, "What is the Insurance?".

For our running example, we assume that the user responds by answering the three questions as: (1) "IHC", (2) "Lynn Jones", and (3) "300 State St., Provo". Observe that if the user had originally added the sentence, "The appointment is for my daughter, Lynn Jones; we live at 300 State St. in Provo; and my insurance is IHC." to the task specification in Section 2, then the system could have extracted this information, and could have executed without asking the user for additional information.

4.5 Constraint Satisfaction and Negotiation

At this point in the process, the system has one free variable, namely the nonlexical object we are trying to establish (e.g., the *Appointment*). If there is only

[3] Verbalization according to [6] and verbalization with respect to the model-equivalent language for OSM [7] are examples of worked-out verbalizations that could be used in our application.

one interpretation that satisfies all the constraints, we are ready establish the object and finalize the process. In our example, an interpretation can only be satisfied if the dermatologist accepts IHC insurance and the distance between the two addresses is less than 5 miles. Assuming this is the case, the system can make the appointment for the user.

However, for the sake of further discussing constraint satisfaction and negotiation, suppose that the dermatologist accepts IHC insurance but that the distance between the addresses is 6 miles. In this case, the interpretation does not satisfy the constraints, and the system must either fail to make an appointment or find a way to relax one or more constraints. One way to negotiate would be to take a generated potential interpretation that does not satisfy the constraints, display the constraints that are not satisfied, and ask the user what to do. Note that this way of negotiating is fully independent of the domain, since the system is able to identify and list each constraint that is not satisfied. For our running example, the system might display the following (hopefully, sprinkled with a lot more syntactic sugar than exemplified here):

```
The following constraint(s) are not satisfied:
LessThanOrEqual(DistanceBetween("600 State St., Orem", "300 State St., Provo"), "5")
    where DistanceBetween("600 State St., Orem", "300 State St., Provo") = 6
What do you wish to do?
```

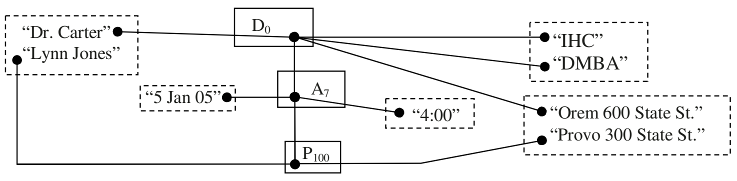
Unfortunately, the system is now beginning a free-form conversation, which it is not in a position to handle. The system can, however, ask the user to edit the task specification, giving looser constraints and then reentering it, by adding, "Please respond by editing your task specification, giving constraints that can be satisfied." The user may, of course, not wish to edit the task specification, in which case, the system reports that it cannot make an appointment satisfying all the constraints.

To further discuss the possibilities, we next consider the system response if there are two or more interpretations that satisfy all the constraints. In this case, the system can respond by offering the alternatives and allowing the user to select one. If there are many interpretations that satisfy all the constraints, we must provide a way to control the potential overload on the user. In the worst case, we can arbitrarily present any k possibilities, where k is small, and let the user select one. As part of our future work, we can likely find ways to have the system rank them, and then present the top-k to the user.

4.6 Process Finalization

At this point in the process, the system either has a single solution or has agreed with the user that there is no solution. Hence, it is straightforward to know what to do. What is interesting here is that although this code cannot be written in advance because it does depend on the domain ontology, it can be generated on-the-fly by code that can be written in advance.

In our example, we assume that the user replaces "5 miles" with "6 miles," and thus that there is a unique solution. Hence, the process ontology schedules the appointment using the action *schedule-appointment("Lynn Jones", ...).*



NextWeek("5 Jan 05")
Person(P_{100}) is at Address("300 State St., Provo") $\wedge$
Dermatologist(D_0) is at Address("600 State St., Orem") $\wedge$
LessThanOrEqual(DistanceBetween("600 State St., Orem", "300 State St., Provo"), "6")
$\exists x_6$ (Dermatologist(D_0) accepts Insurance(x_6) $\wedge$ Equal("IHC", x_6))

Fig. 7. The scheduled appointment

Figure 7 shows the scheduled appointment. As shown, appointment A_7 is scheduled for person P_{100} whose name is "*Lynn Jones*", with dermatologist D_0 whose name is "*Dr. Carter*" on date "*5 Jan 05*" at time "*4:00*" at address "*600 State St., Orem*". The process ontology notifies the user that the appointment is successfully scheduled.

The subprocess *schedule-appointment(...)* is domain dependent because it needs knowledge about what object sets should be filled in with which objects in order to schedule an appointment. However, given the ontology and the values for the free variables obtained from the unique interpretation created by this time during the execution of the process ontology, the system can use this knowledge to automatically generate code for this last part of the process. Observe that this holds for any domain so long as the objective is to insert an object into an object set of interest and then satisfy all applicable constraints. Thus, since this is exactly the kind of service our system provides, it is always possible for the system to generate the finalization step for any domain ontology.

5 Prototype Implementation Status and Future Work

The success of our conceptual-model-based, semantic-web-services system will depend largely on its ability to successfully extract information from free-form text specifications of desired services. Our long-standing work on information-extraction ontologies [8] provides the basis for this component of our system. Building on our work on information-extraction ontologies, we have implemented an initial end-to-end prototype that accepts free-form text specifications; finds an appropriate task ontology for the specification (if one exists); produces a task view for the specification; determines whether there is missing information; and, if not, establishes the primary object and thus performs the service. Our system does not yet obtain and use task-specified functions; does not yet use its system database to obtain values for free variables, and does not yet do negotiation. Adding these features is part of our current work.

In the future, we expect to investigate more explicitly the boundaries of applicability. Should the system, for example, be required to handle more complex cases? The system as currently envisioned does not, for instance, allow users to compose tasks nor to specify conditional or iterative tasks. As currently proposed, a user can compose tasks only by specifying two successive tasks, e.g. make an appointment with a dermatologist and then make an appointment for a haircut. For conditional tasks, such as "if I can see Dr. Peterson within a week, make an appointment with Dr. Peterson; otherwise make an appointment with Dr. Carter," the user would have to query the system to determine whether an appointment could be made with Dr. Peterson within a week and then, depending on the answer, either make an appointment with Dr. Peterson or Dr. Carter. Further, it is also not clear whether the system needs to handle complex task specifications that would require the system to use natural language processing or other techniques to disambiguate sentence structures or to resolve pronoun references. As the system now stands, users would have to become used to its limited ability to actually understand. Whether this is sufficient to be serviceable for the general public is not yet known, but there is reason to believe it is, and there is reason to believe that this approach will be more widely acceptable to ordinary users than systems that allow service selection [9] or require an artificial, formalized subset of natural language [10].

6 Concluding Remarks

We have described a system that makes it possible for ordinary users to invoke services using form-free task specifications. As salient features, the system strongly relies on (1) conceptual modeling, which forms the basis for both domain ontologies and process ontologies, (2) extraction ontologies, which allows the system to obtain the information it needs to match user requests with an appropriate domain ontology, and (3) domain-independent process ontologies that can be automatically specialized for any given domain, which makes our approach work across domains without need for manual configuration.

Acknowledgements

This work is supported in part by the National Science Foundation under grant IIS-0083127 and by the Kevin and Debra Rollins Center for eBusiness at Brigham Young University under grant EB-05046.

References

1. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American **284** (2001) 34–43
2. Embley, D.W.: Programming with Data Frames for everyday Items. In Medley, D., Marie, E., eds.: Proceedings of AFIPS Conference, Anheim, California (1980) 301–305
3. Embley, D.W., Kurtiz, B.K., Woodfield, S.N.: Object-Oriented Systems Analysis: A Model Driven Approach. Yourdon Press, Englewood Cliffs, New Jersey (1992)

4. Widom, J., Ceri, S.: Active Database Systems. Morgan–Kaufmann, San Mateo, California (1995)
5. Papamarkos, G., Poulovassilis, A., Wood, P.T.: Event-Condition-Action Rule Languages for the Semantic Web. In Cruz, I.F., Kashyap, V., Decker, S., Eckstein, R., eds.: Proceedings of the first International Workshop on Semantic Web and Databases (SWDB 2003), Humboldt-Universität, Berlin, Germany (2003) 309–327
6. Halpin, T.: Business rule verbalization. In: Proceedings of the 3rd International Conference on Information Systems Technology and its Applications, Salt Lake City, Utah (2004) 39–52
7. Liddle, S., Embley, D., Woodfield, S.: An active, object-oriented, model-equivalent programming language. In Papazoglou, M., Spaccapietra, S., Tari, Z., eds.: Advances in Object-Oriented Data Modeling. MIT Press, Cambridge, Massachusetts (2000) 333–361
8. Embley, D., Campbell, D., Jiang, Y., Liddle, S., Lonsdale, D., Ng, Y.K., Smith, R.: Conceptual-model-based data extraction from multiple-record web pages. Data & Knowledge Engineering **31** (1999) 227–251
9. Agarwal, S., Handschuh, S., Staab, S.: Surfing the Service Web. In: Proceedings of the Second International Semantic Web Conference (ISWC2003), Sanibel Island, Florida (2003) 211–226
10. Bernstein, A., Kaufmann, E., Fuchs, N.: Talking to the semantic web – a controlled english query interface for ontologies. AIS SIGSEMIS Bulletin **2** (2005) 42–47

Automatically Grounding Semantically-Enriched Conceptual Models to Concrete Web Services

Eran Toch, Avigdor Gal, and Dov Dori

Technion – Israel Institute of Technology,
Technion City, Haifa 32000, Israel

Abstract. The paper provides a conceptual framework for designing and executing business processes using semantic Web services. We envision a world in which a designer defines a "virtual" Web service as part of a business process, while requiring the system to seek actual Web services that match the specifications of the designer and can be invoked whenever the virtual Web service is activated. Taking a conceptual modeling approach, the relationships between ontology concepts and syntactic Web services are identified. We then propose a generic algorithm for ranking top-K Web services in a decreasing order of their benefit vis-á-vis the semantic Web service. We conclude with an extention of the framework to handle uncertainty as a result of concept mismatch and the desired properties of a schema matching algorithm to support Web service identification.

1 Introduction

Web services allow universal connectivity and interoperability of applications and services, using well-accepted standards as UDDI, WSDL, and SOAP. Current Web service standards focus on syntactic, operational details for implementation and execution rather than semantic capabilities description. A recent development enables the specification of semantic Web services. The semantic Web [5] aims to extend the World-Wide-Web by representing data on the Web in a meaningful and machine-interpretable form. The semantic Web is based on a set of languages that provide well-defined semantics and enable the markup of complex taxonomic and relations between entities on the Web. Ontologies, commonly defined as specifications of a conceptualization, [20] serve as the key mechanism for the semantic Web by allowing concepts to be globally defined and referenced. A leading language for ontology modeling for the semantic Web is the *Web Ontology language (OWL)* [9], providing a semantic markup for the definition of concept classes, relationships among them, and their instances. Several methods for annotating Web services with semantic metadata have been proposed. One of the most prominent methods is OWL-S [3]. Based on OWL, it provides an ontology for Web services, enabling a description of the service's profile, process model and its grounding - a mapping to the syntactic definition of the concrete Web service.

We envision a world in which some Web services have semantic descriptions, while others are only syntactically defined (using WSDL, for example). In particular, designers can define a "virtual" Web service as part of their business processes. An execution engine is required to look for actual Web services that match the specifications of the designer and can be invoked whenever the virtual Web service is activated. The design of semantic Web services can be an iterative process, starting from rough design, and gradually refine the design based on feedback from some mechanism that grounds the semantic Web service to some existing Web services.

It is the aim of this paper to provide a framework for a model-driven design using semantic Web services. Taking a conceptual modeling approach, relationships between ontology concepts and syntactic Web services are identified. We then propose a generic algorithm for ranking top-K Web services in a decreasing order of their benefit vis-á-vis the semantic Web service. We conclude with a discussion on extending the framework to handle uncertainty that stems from concept mismatch and the desired properties of a schema matching algorithm to support Web service identification.

The main contribution of this work is twofold. At the conceptual level, we introduce a method for designing business processes as a composite set of Web services. At the algorithmic level, we provide a generic algorithm for ranking concrete Web services with respect to their suitability in fitting a semantic Web service description, in effect offering a model-driven approach for service-oriented computing. The semantic Web service serves as a conceptual model. Rather than generating a code out of the model, the model is implemented by locating and invoking existing services. It is worth noting that the concrete Web services are not necessarily annotated with semantic meta-data, and may be described as WSDL documents, reflecting the current state of affairs. Finally, we discuss the characterization of requirements for a schema matching algorithm should satisfy to qualify for interfacing with the Semantic Web.

The rest of the paper is organized as follows. Section 2 presents the model and formally defines the problem. The use of ontologies in ranking Web services is given in Section 3, followed by an algorithm for the matching process (Section 4). Section 5 discusses an extension to support semantic heterogeneity. Section 6 contains a related work. The paper concludes with a summary and future work (Section 7).

2 Model and Problem Definition

In this section, we provide a formal definition of the two main elements of our model, namely Web services (Section 2.1) and Semantic Web services (Section 2.2). We conclude with a formal introduction of the problem at hand (Section 2.3).

2.1 Web Services

Web services are loosely coupled software components, published and invoked across the Web. Several XML-based standards ensure the regulation of discovery and the interaction of Web services. In particular, UDDI allows Web services to be discovered through a keywords search. A Web Services Description Language (WSDL) document describes the interface and communication protocol of Web services. In this paper, we use restricted WSDL definition, ignoring namespaces, faults handling, and communication issues. Therefore, a Web service is a quadruple, $WS = (T, M, O, A)$, where:

- T is a finite set of types. A type can be primitive (*e.g.*, integer) or complex, described by an XML schema.
- M is a finite set of messages. Each message is defined by a name and a type, $t \in T$.

– O is a finite set of operations provided by the service.
– $A : M, R \rightarrow O$ is a finite set of assignments, each of which assigns a set of messages in $\{m_1, m_2, ..., m_n\} \in M$ and $R = \{input, output\}$ to an operation $o \in O$. Each message can serve as either an input or an output of the operation.

Current Web service architecture suffers from several limitations. In particular, although Web services are designed to provide distributed interoperability among applications, lack of semantic definition of these applications make the automatic integration and discovery of Web services a difficult task.

2.2 Semantic Web Services

Applying the advances of the Semantic Web to Web services, resulted in OWL-S [3]. OWL-S is a language for specifying Web service ontology, based on OWL, which augments current Web services architecture with semantic metadata. It provides a set of markup language constructs for describing the properties and capabilities of Web services, facilitating the automation of Web service tasks, including automated discovery, execution, composition and interoperation. An OWL-S ontology includes three sections, namely a profile ontology (what the service does), a process-model ontology (how it works) and a grounding ontology (how it can be used). The profile ontology extends the UDDI language, providing semantic annotation for the parameters the service accepts and provides, as well as general information describing the service.

We use as a case study, a semantic Web service named *Book Price*. The service receives a book title and a currency, locates the book's information, retrieves a price quote for it and convert it into a desired currency.[1] Figure 1 provides a visual illustration of the service using OPM/S [15], which serves as a modeling and visualization method for semantic Web services. OPM/S is an extension of Object-Process Methodology (OPM) - a conceptual object-oriented and process-oriented modeling language that supports the semantic Web [13,14]. OPM/S models are composed of two entity types, namely services (represented as ellipses), and parameters that pass between (and possibly modified by) services, represented as rectangles. Semantics is annotated by tagging the entities with their ontological concepts in the upper-left corner of the entity.

The *Book Price* service returns the price of a book given its name and a desired currency. The service is composed of three atomic services, namely *Book Finder*, *Price Finder,* and *Currency Converter*. *Book Finder* receives a book name and produces the book information, if the book was found. *Price Finder* returns the book price in dollars, and *Currency Converter* converts the price to the desired currency. The example illustrates our notion of designing semantic Web services as an iterative process. In particular, note that the output of *Book Finder* is a rather fuzzy term of *Book Info*. As we will show later, this term is grounded in an ontology, yet it leaves the designer some maneuvering space. The designer has no particular preference at this time as to the exact form of *Book Info*, as long as it can serve as an appropriate input to *Price Finder*.

The process-model is defined as a workflow of processes, each being a quadruple $PR = (IN, OUT, E, P)$, where IN is a set of input parameters, OUT is a set of output parameters, E is a set of the process effects and P is a set of preconditions. Processes

[1] Available at: http://www.mindswap.org/2004/owl-s/services.shtml

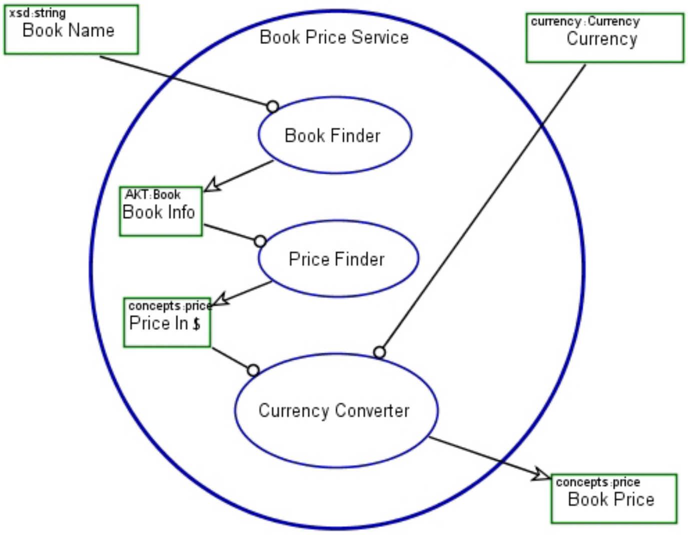


Fig. 1. Book-Price Service

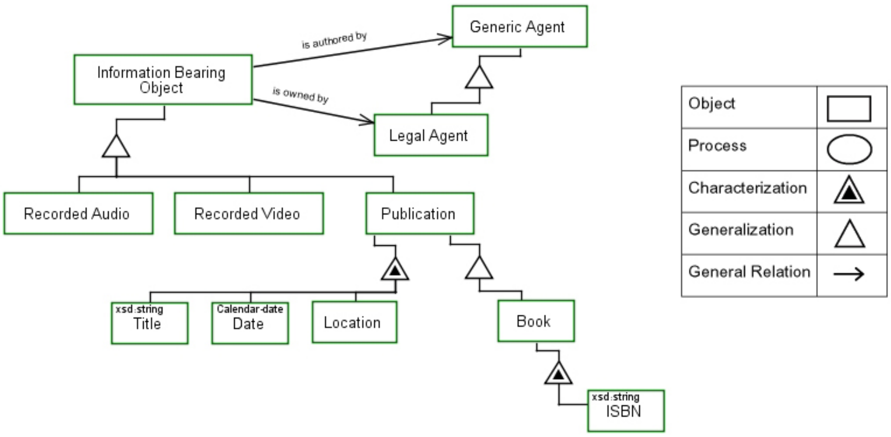


Fig. 2. The AKT Portal Ontology, visualized in OPM

can be atomic or composite. Atomic processes are invoked in a single step and can be mapped directly to a WSDL operation. Composite processes, in contrast, represent a complex structure of processes. Formally, a composite process augments the process structure described above with a set of subprocesses (either atomic or composite), executed according to a certain control construct (such as parallel, sequential, conditioned, *etc*). For instance, the three subprocesses of the *Book Price Service* are executed sequentially starting from the top process. The last section of the OWL-S ontology is the grounding ontology. It provides a mapping between the atomic processes to the WSDL definition of the concrete Web service.

Elements of the profile and process-model sections, such as input and output elements, can be mapped to concepts in accompanying ontologies or to primitive XML datatypes. To illustrate this mapping, we use the AKT portal ontology [1], visualized in Figure 2 using OPM The *Book Info* parameter object in the semantic Web service (Figure 1) is mapped to a *Book* concept, described in the ontology. Given an ontology with a set of concepts C, the function $tag : IN \cup OUT \cup E \cup P \rightarrow C^*$, maps a parameter to its underlying set of concepts. A closer look at the portal ontology reveals the relationships between the *Book* concept to other concepts. The *Book* concept is a specialization of the *Publication* concept. *Publication* is characterized by *Location*, *Date* and *Title*, and is a specialization of *Information Bearing Object*.

2.3 Problem Definition

Web service discovery is a process, in which a Web service is matched based on given specifications. In this work we focus on specifications that are given as semantic Web services. Given an *atomic* process PR and a set $\mathcal{OP} = \{OP_1, OP_2, ...OP_p\}$ of operations within WSDL-described Web services, let $\rho_{PR} = (\preceq_{PR}, \mathcal{OP})$ be a partial order of operations, representing their relative fit for *implementing* PR. Therefore, if $OP_i \preceq_{PR} OP_j$ then OP_j is better suited to be executed as an implementation of PR than OP_i. Typically, ρ_{PR} may not be known in advance and currently a manual intervention on a grand scale may be required to ensure a selection of a suitable Web service.

In an attempt to automate the process and avoid gross errors in the discovery process, we propose the ranking of the best top-K suitable Web services, rather than providing a single Web service. Formally, given a process PR, a domain ontology ON, and a set $\mathcal{OP} = \{OP_1, OP_2, ..., OP_p\}$ of available Web services, we wish to generate a ranked mapping $\mathcal{OP}' = \left\{OP_{(1)}, OP_{(2)}, ..., OP_{(k)}\right\}$ of K Web services such that:

- $\forall i < j \leq k \; OP_j \preceq_{PR} OP_i$, and
- $\forall k < l \; OP_l \preceq_{PR} OP_k$.

3 Web Service Ranking Using Ontologies

This section provides a conceptual ontology-based model for Web service ranking. Section 3.1 describes context classes and mark vectors, which are the primary tools for conceptual analysis of semantic Web services. Section 3.2 describes the method of ordering results of queries for Web services. Section 3.3 provides a detailed example.

3.1 Classification into Context Classes

We start our conceptual analysis by observing that ontologies provide a natural ranking mechanism that can be derived from the semantics of ontological constructs. Given a concept $c \in C$, annotated as the "anchor" concept, all concepts in C can be classified into one of four *context classes*, as follows:

Exact. Concepts that have identical semantic meaning,including c itself and its properties. OWL provides relations such as *equivalentClass* to define concept equivalence.

Table 1. Classification of Semantic Relations to context classes

Context Class	**OWL relations**
Exact	direct mapping, owl:equivalentClass, owl:equivilantProperty, owl:sameIndividualAs, owl:ObjectProperty, owl:DatatypeProperty
Specific	owl:subClass, owl:intersectionOf, owl:oneOf, individual, $\{x \mid \forall c \in Specific, x = ObjectProperty(c) \vee x = DatatypeProperty(c)\}$
General	owl:unionOf, super-class, inverse(property), class-of $\{x \mid \forall c \in General, x = ObjectProperty(c) \vee x = DatatypeProperty(c)\}$
Negation	owl:complementOf, owl:disjointWith

General. Concepts that supply higher-level context. For instance, the *Publication* concept is a super-class of the *Book* concept and therefore falls under the category of General with respect to *Book*.

Specific. Concepts that provide a more specific context. *Book* belongs to the Specific class of *Publication*.

Negation. Concepts which have explicit contradicting meaning. For instance, in OWL, if class c_1 *disjointWith* class c_2, then an instance of c_1 cannot be an instance of c_2.

While this classification is not new, careful attention should be given to properties, to which actual instance values are attached. Recall that our goal is to rank Web services according to their adequacy for the task at hand. Therefore, parameters from the semantic Web service description should be mapped to input and output messages of operations. Whenever a parameter is not grounded in a property element of an ontology, we should translate this grounding in terms of properties. Therefore, a class is represented by a subset of its properties, while a relation is represented by a subset of the properties of the class(es) with which it is associated. We should emphasize that this is not generally true, and such simplification is needed due to the simple format of WSDL.

Table 2. Mark vectors and their classifications

Exact	**General**	**Specific**	**Negation**	**Ranking Category**
1	1	1	0	ACCURATE
1	1	0	0	
1	0	1	0	
1	0	0	0	CONTEXT-LESS
0	1	1	0	CONTEXT-ONLY
0	1	0	0	
0	0	1	0	
0	0	0	1	INVERSE
0	1	0	1	UNCERTAIN
0	0	1	1	
0	1	1	1	
1	0	0	1	
1	1	0	1	
1	0	1	1	
1	1	1	1	
0	0	0	0	EMPTY

If a concept is mapped to a certain context class, its properties (both object properties and datatype properties) are added to the corresponding context class.

Given an ontology with a set of concepts C, there are $2^n - 1$ interesting combinations of concepts from C, ranging from individual concepts to a set of all concepts. Any such combination $C' \subseteq C$ is a possible CNF query to a Web service search engine. Let the response to such query be all Web services for which these concepts are considered relevant by the search engine. For example, given a query $C' \subseteq C$, Woogle [12] returns all Web services for which all concepts in C' appear in their WSDL description.

Given a concept $c \in C$, a query $C' \subseteq C$ can be mapped to a vector $mark = (e, g, s, n)$ of binary variables, representing the context classes Exact, General, Specific and Negation, respectively. A variable is assigned with a value of 1, if exists $c' \in C$ such that c' belongs to the relevant context class of c. We now consider the AKT portal ontology (Figure 2) for some examples. Assuming the anchor concept is *Book*, then the queries $\{Book\}$ and $\{Book \wedge ISBN\}$ are mapped to the mark vector $(1,0,0,0)$ since they contain concepts from the exact context class. The query $\{Book \wedge ISBN \wedge Title\}$ contains both Exact concepts (*Book* and *ISBN*) and a General concept (*Title*). Hence, it is mapped to the vector $(1,1,0,0)$.

Table 2 provides the complete mark vectors set. The right column groups the vectors into 5 ranking categories according to the matching pattern derived from the vector. For instance, the ACCURATE category contains results that originate from both direct mapping concepts and context concepts. Thus, it has the potential to produce results with high accuracy. The CONTEXT-LESS category contains results that originated from direct mapping concepts only, without a match to supporting context concepts.

3.2 Ranking of Mark Vectors

The ranking of Web services relies on ranking of mark vectors, based on the marginal benefit of the concepts that form the vector. Each context class has a different contribution to the vector. For instance, a concept that belongs to the General class provides context to the query and potentially increases its precision. On the other hand, concepts that belong to the Negation class may lead to erroneous results. A partial order between mark vectors is proposed in Figure 3. Mark vectors of the ACCURATE category are ranked higher than any other category, because their compatible queries contain both exact and contextual concepts. Specifically, it is considered more accurate than vectors of the CONTEXT-LESS and CONTEXT-ONLY categories. However, no order is determined between CONTEXT-LESS and CONTEXT-ONLY categories. Order between vectors within a category is relevant to the ACCURATE and CONTEXT-ONLY categories, and it is based on the weighted sum of matching concepts. The three top classes represent positive queries, those originating from direct and contextual concepts. The three bottom classes represent negative queries, which will retrieve empty, unfavorable or doubtful results. Queries which are assigned to the same mark vector may have an internal ranking between them, according to the number of concepts that form the query. If C_1 and C_2 are two queries, and the number of concepts in C_1 is higher than the number of concepts in C_2, then C_1 is considered more precise and therefore it is ranked higher.

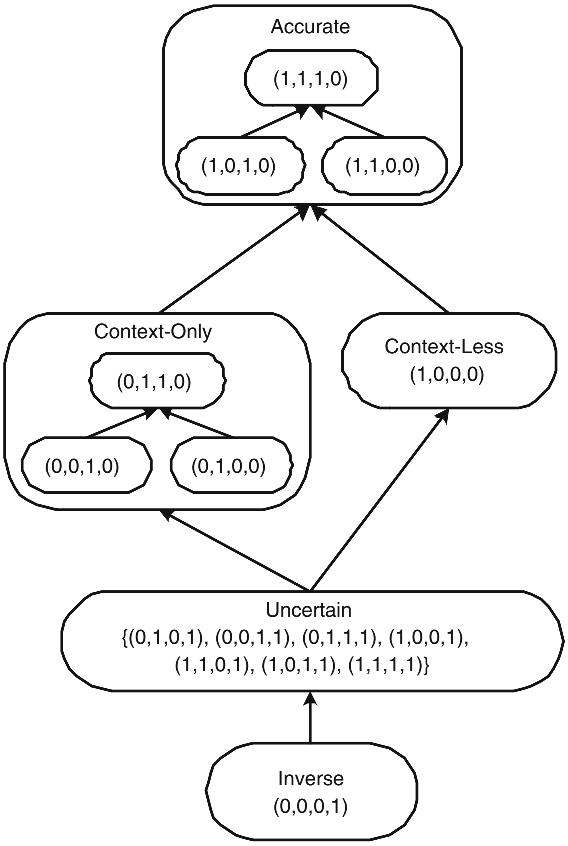


Fig. 3. Partial Order between Mark Vectors

3.3 Example

Consider the *Book Price* Service, as described in Figure 1, and the AKT portal ontology (Figure 2). To ground the first atomic process of the service - *Book Finder* - we describe the output parameter, *Book Info*, using a set of concepts from the portal ontology. The set C will be defined as all elements of the ontology (including properties), and the anchor concept is defined as *Book*. The context classes for *Book* are defined as follows:

- $Exact = \{Book, ISBN\}$
- $Specific = \phi$
- $General = \{Publication, Title, Date, Location, IsAuthoredBy, IsOwnedBy \ldots\}$
- $Negation = \phi$

The next step is to build a set of queries $C' \subseteq C$. For instance, the following list contains a subset of the conjunctive normal form queries that can be constructed from concepts of C, and their compatible mark vector:

$$\{Book\}, \{Book \wedge ISBN\} \Rightarrow (1,0,0,0)$$
$$\{Book \wedge ISBN \wedge Publication\}, \{Book \wedge ISBN \wedge Publication \wedge Title\} \ldots \Rightarrow (1,1,0,0)$$

adding concepts to the query can potentially increase its precision (and decrease its recall). Therefore, each query in the list is ranked higher than the subsequent query on its left hand side. Furthermore, queries that are assigned to mark vectors of the ACCURATE class are ranked higher than ones of the CONTEXT-LESS class. Figure 4 describes the outcome of executing the queries through the Woogle search engine [12] on real Web services. The dashed circles represent sets of services that were retrieved using a single query. For instance, the following services were returned in response to the $\{Book\}$ query: *Travel SerkoA* , *K4THotel*, *Neil Finn Travel*, *Barnes & Noble Quote Service* and *Book Info Service*. These services belong to two different domains: the travel domain, where the word Book is used in the context of booking a flight, and the publication domain, which is the one we need. Intersection of this set with the set induced by $\{ISBN\}$, results in a subset of the previous set. It is worth noting that the *Book Info Service*, which is the only service that answers the requirements, is retrieved by the query $\{Book \wedge ISBN \wedge Publication \wedge Title\}$. This query is ranked higher than $\{Book \wedge ISBN\}$, so the final ranking will be:

Book Info Service $\prec_{PR}$ *Barnes & Noble Quote Service* $\prec_{PR}$ *Neil Finn Travel*...

4 Generic Ranking Algorithm

We now present a generic ranking algorithm that takes as input an atomic process *PR*, a domain ontology *ON*, and a set of WSDL-described operations $\mathcal{OP}$, and returns a set of operations $\mathcal{OP}'$, which are ranked according to their capability of implementing

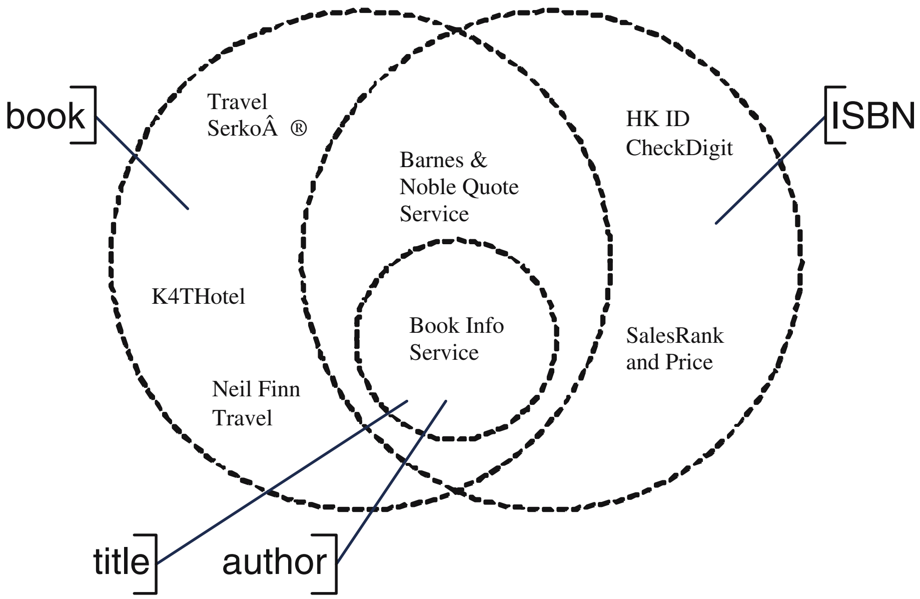


Fig. 4. Matching Results for the BookFinder Process

PR. The basic idea behind the algorithm is to infer affinity between a process and each operation, and use it as a measure for ranking. Affinity is derived from the mappings between process parameters $(p_1, p_2, ..., p_k)$ and operation messages $(m_1, m_2, ..., m_n)$. Such a mapping, as presented in Section 3, is provided through the ontological concepts, associated with parameters. Figure 5 visualizes this notion for parameter p_1. p_1 is represented by concepts C_1 and C_2, which are directly mapped to message m_1.

An implicit assumption in the ensuing discussion is that two operations that take the same parameters as input perform the same intended activity. Researchers have argued against this assumption [32]. Therefore, we provide a motivation for keeping it in this specific context. According to the model, the behavior of a semantic Web service is expressed using composite processes. Such flexibility allows a designer to use **atomic** processes for constructing composite ones. Therefore, one can assume that *PR* is sufficiently simple, or else it would be designed as a composite process and be further decomposed into atomic processes. While "sufficiently simple" is a vague metric, we assume that the output variance of such processes is limited, implying that the interface of the operation is sufficient for matching purposes. Moreover, Web service designers who wish external programs to use their proposed Web services are expected to develop simple, well-documented processes, again supporting our assumption.

The basic idea behind the algorithm is to use the underlying ontologies to produce a contextual matching of parameters, as defined in Section 3. The algorithm analyzes the semantic relationships between concepts and produces a ranking of the parameter matching results, which is reflected later in a ranking between the operations.

The algorithm iterates through all the parameters of a process *PR*, extracting the concepts related to that parameter. A mapping is established by procedure *MatchMessages* between each parameter and each operation message. The mapping is an assignment $Mapping : (message, parameter) \rightarrow [0,1]$, assigning a *matching score* to each message-parameter pair. The overall ranking of the operation set is calculated by the function $score(OP)$, which is described below. Finally, a ranking between the operations is established according to the matching score.

An important aspect of the algorithm is the use of related concepts. The set *Concepts* holds concepts, extracted from the ontology. Each concept is tagged with its relation to

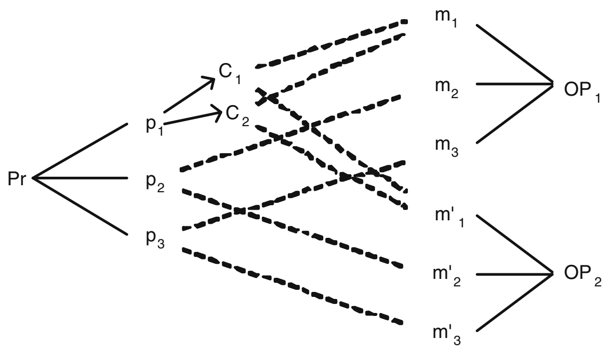


Fig. 5. Mapping between semantic processes and WSDL operations

Algorithm 1 Rank Operations

```
Input: $PR$, $ON$, $\mathcal{OP} = \{OP_1, OP_2, ..., OP_p\}$
Output: $\mathcal{OP}' = \{OP_{(1)}, OP_{(2)}, ..., OP_{(k)}\}$

$Mappings \leftarrow \phi$
$Messages \leftarrow \bigcup_{OP_i \in \mathcal{OP}} A^{-1}(OP_i)$
for all $paremeter \in PR$ do
  for all $c \in concept(parameter)$ do
    $Concepts \leftarrow (c, \text{“}direct\text{“})$
    $Concepts \leftarrow (related(c, ON), \text{rtype})$
  end for
  $ParameterMapping \leftarrow MatchMessages(Concepts, Messages)$
  $Mappings \leftarrow Mappings \cup (parameter, ParameterMapping)$
end for
for all $OP_i \in OP$ do
  $\mathcal{OP}' \leftarrow \mathcal{OP}' \cup OP_i$
  for all $OP_{(j)} \in \mathcal{OP}'$ do
    $OP_{(j)} \preceq_{PR} OP_{(i)} \Leftrightarrow score(OP_{(j)}) \geq score(OP_{(i)})$
  end for
end for
```

the original concept *(i.e.*, the concept to which the parameter is tagged). For example, in the *BookPrice* process definition (Figure 1), the parameter *Book Info* is mapped directly to the concept *Book* of the AKT portal ontology (Figure 2). We define the concept *Book* as a *semantic anchor* within the ontology, and it is tagged with the *direct* tag. The $related(c, ON)$ function retrieves a list of concepts which are related to the anchor concept in the ontology *ON* and tags each concept according to its semantic relation type (*rtype*) to the anchor object.

After characterizing the concepts according to their semantic relation, *MatchMessages* is called in order to compute a matching score between the parameter and each of the available messages. The core of *MatchMessages* is elaborated in Section 3. Basically, it produces a mapping between the set of concepts, representing the OWL-S process parameter set, and each set of messages, specified in the interface of the WSDL operations. A precondition of the existence of any semantic correspondence between the message and the parameter is data-type equivalence between the two. The function *MatchDataType* compares the types of the parameters of the corresponding concepts to be matched. For instance, if the primitive type of the direct-mapping concept of the parameter is *xsd:string* and the primitive type of the message is *xsd:float*, then the message cannot match the parameter.

If the message passed the data-type test, the procedure applies a virtual matching function, *GenericMatch*, in order to calculate the similarity between each concept and each message. The abstract function can be implemented using string matching, linguistic similarity or schema matching techniques.[2]

[2] See [18,29,25,11] for examples of matching methods.

The last stage of the algorithm involves the ranking of the operations according to their overall matching score. An operation $OP_{(j)}$ is ranked higher than operation $OP_{(i)}$ if it has a higher or equal score. The calculation is defined as follows:

$$score(OP) = \frac{1}{|Parameters|} \sum_{pr,m \wedge A(m)=op} Mapping(pr,m) w_{pr} \cdot \prod_{pr} h(OP,pr)$$

$$h(OP,pr) = \begin{cases} 1 \text{ if } pr \text{ is mapped by at least one message } m \in OP, \\ \quad \text{such that } Mapping(pr,m) > 0 \\ 0 \text{ otherwise} \end{cases}$$

The function averages the matching scores (*Mapping*) for each message that participate in some assignment A and each of the process parameters ($pr \in Parameters$). Parameter importance is specified using a weight w_{pr}, which is associated with each parameter. To omit operations that have only a partial mapping to the process, we use h to assign a value of zero whenever the mapping of the operation does not supply a corresponding message for each of the process parameters.

5 Web Services and Schema Matching

Schema matching is the task of matching between concepts describing the meaning of data in various data sources (*e.g.* database schemata, XML DTDs, HTML form tags, *etc.*). As such, schema matching is recognized as one of the basic operations required by the process of data integration [7]. Due to its cognitive complexity [8], schema matching has traditionally been performed by human experts [22]. As the automation level of data integration increases, the ambiguity inherent in concept interpretation is becoming a major obstacle to effective schema matching. For obvious reasons, manual concept reconciliation in dynamic environments (with or without computer-aided tools) is inefficient and at times close to impossible. Introduction of the Semantic Web vision [5] and shifts toward machine-understandable Web resources and Web services have underlined the pressing need for automatic schema matching.

Attempting to address these data integration needs, several heuristics for automatic schema matching have been proposed and evaluated in the database community (*e.g.*, see [4,10,21,16,26,31,18]). However, as one would expect, recent empirical analysis has shown that there is no single dominant schema matcher that performs best, regardless of the data model and application domain [17], and such schema matcher may never be found. Finally, due to the unlimited heterogeneity and ambiguity of data, none of the existing heuristics can find optimal mappings for many pairs of schemata.

Bearing these observations in mind and striving to some robustness in the matching process, an approach studied in [2,17,24] suggested to generate not one, but K top-ranked mappings, examining them (either iteratively or simultaneously) until a sufficiently effective mapping is found. In this work, we adopt this research direction and aim at extending Web service ranking to support imprecise matching. Observe that an operation that implements *PR* fully is expected to have corresponding input and output messages, which have the same semantics in spite of name differences. Therefore, matching an ontology concept with a Web service input and output parameter may carry with it a degree of uncertainty. As a simple example, consider the concept *Title*. This concept may be matched (by one or another matching algorithm) with a concept *Book Title*, and assigned a similarity of 0.5. *Title* belongs to its own Exact context class, and therefore the *e* variable in the *marks* vector should be assigned a non-negative number. A natural extension to the approach presented in this work, to support the uncertainty that stems from partial mappings, can be done by allowing each variable in the *marks* vector to accept values in $[0,1]$, corresponding to the similarity measure as determined by the matching algorithm(s) of choice. Such a change entails a revision in the partial order among different *marks* vectors, as given in Figure 3. We defer this extension to an extended version of this work.

Even though ontology languages such as OWL provide a formal set of constructs and relations, the construction of semantic Web services and ontologies may vary significantly among designers. This difference, which may be due to the methodologies, conventions, purposes and even the "style" the designers exhibit, can greatly affect the results of the algorithm described in this work. Therefore, an analysis of the way ontological knowledge is modeled and understood by humans is necessary. Specifically, research of the implications of different relations in ontological languages such as OWL is relevant to our work. Emerging works in this issue include [6] and [30], but further research is needed in this field.

6 Related Work

Our work presents an approach for grounding descriptions of semantic (possibly virtual) Web services, which exhibit a rich conceptual model, with physical (possibly not semantic) Web services, through their flat WSDL description. In this section we discuss other efforts that tackle similar problems. Paolucci et al. [28] proposed a method for matching semantic Web services. The work uses the OWL-S profile ontology as a method for describing the capabilities of services, and proposes an algorithm that matches service requesters and advertisers. The work requires the availability of full semantic description of both service requester and service advertiser in order to perform the matching. Furthermore, it requires a common ontology, or at least two connected ontologies. Klein et al [23] have used process ontology in order to match between services. The work proposes an indexing schema for services in order to promote efficient matching of services. Similar to [28], [23] requires services to be semantically annotated and indexed before matching can be executed. Our work differs primarily in the problem definition - we are concerned with grounding services to WSDL descriptions, not with matching semantic Web services. We demonstrate that a major contribution to a matching process can be made even if ontology exists only for one side of the match.

A prominent example of an architecture that supports dynamic composition of Web services is Proteus [19]. Proteus suggests an information mediating approach towards building dynamic compositions of Web services. It exhibits a multi-level architecture, which includes wrapping existing data sources with Web services, locating Web services through attribute search, building dynamic integration plans and efficiently executing the services using compression techniques. However, this approach requires Web services to be manually annotated, using a specialized ontology, in order to be reused.

METEOR-S [29] is a framework for annotating WSDL descriptions with semantic metadata. It provides a framework for semi-automatic mapping between WSDL elements and ontological concepts. The matching algorithm is based on linguistic matching at the single concept level which is enhanced with schema-based matching between the XML schema specification of WSDL elements and the ontological structure. Our work differs in the direction of matching and in its nature. While the matching algorithm used in METEOR-S takes as input a WSDL document and matches it to an ontology, our proposed algorithm performs the opposite task: it takes an ontology-powered conceptual model and tries to match it with WSDL documents. This difference has considerable implications on the algorithm. While METEOR-S is basically a schema matching algorithm, our approach acknowledges the rarity of rich XML schemas in WSDL documents. The lack of such schemas is compensated for by projecting the ontological knowledge onto keyword queries.

Our work is also related to efforts for developing matching algorithms for non-semantic Web services. Woogle [12] is a search engine for Web services. A similar goal is shared by [27], which uses a different algorithm. Woogle accepts keyword queries and returns results according to information in WSDL documents, including message parameters. Our proposed framework is complementary to that of Woogle in the type of queries it accepts. Our algorithm decomposes conceptual models into keywords and then uses a generic algorithm in order to match the keywords. Woogle can be used as an implementation for the latter task.

7 Conclusion

This paper describes a conceptual framework for designing composite business processes using semantic Web services, grounding them with existing (either semantic or other) Web services. A designer defines a rough draft of a semantic Web service to be searched. The system then searches for existing Web services that match the specifications and ranks them according to their fit with the proposed process design. Modeling languages such as OPM/S can be used for rapid specification of semantic Web service.

We use ontologies as the main vehicle for conveying semantics and utilize ontological constructs in the ranking process. We then propose a possible extension of the framework to handle poor Web service specifications and semantic heterogeneity. An extended version of this work will include a fully specified methodology for designing composite business processes in an environment with varying levels of semantic specifications. Other extensions of this work include efficient algorithmic solutions to the ranking problem, using pruning and indexing.

Acknowledgement

The work of Gal and Dori was partially supported by TerreGov, a European Commission 6^{th} Framework IST project and the Technion Fund for the Promotion of Research.

References

1. The akt reference ontology. http://www.aktors.org/publications/ontology/, 2002.
2. A. Anaby-Tavor. Enhancing the formal similarity based matching model. Master's thesis, Technion-Israel Institute of Technology, May 2003.
3. A. Ankolekar, D.L. Martin, Z. Zeng, J.R. Hobbs, K. Sycara, B. Burstein, M. Paolucci, O. Lassila, S.A. Mcilraith, S. Narayanan, and P. Payne. DAML-S: Semantic markup for web services. In *Proceedings of the International Semantic Web Workshop (SWWS)*, pages 411–430, July 2001.
4. J. Berlin and A. Motro. Autoplex: Automated discovery of content for virtual databases. In C. Batini, F. Giunchiglia, P. Giorgini, and M. Mecella, editors, *Cooperative Information Systems, 9th International Conference, CoopIS 2001, Trento, Italy, September 5-7, 2001, Proceedings*, volume 2172 of *Lecture Notes in Computer Science*, pages 108–122. Springer, 2001.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The semantic Web. *Scientific American*, May 2001.
6. Abraham Bernstein, Esther Kaufmann, Christoph Bu"rki, and Mark Klein. How similar is it? towards personalized similarity measures in ontologies. In *7. Internationale Tagung Wirtschaftsinformatik*, February 2005.
7. P.A. Bernstein and S. Melnik. Meta data management. In *Proceedings of the IEEE CS International Conference on Data Engineering*. IEEE Computer Society, 2004.
8. B. Convent. Unsolvable problems related to the view integration approach. In *Proceedings of the International Conference on Database Theory (ICDT)*, Rome, Italy, September 1986. In *Computer Science*, Vol. 243, G. Goos and J. Hartmanis, Eds. Springer-Verlag, New York, pp. 141-156.
9. M. Dean, G. Schreiber, F. van Harmelen, J. Hendler, I. Horrocks, M. McGuinness, P.F. Patel-Schneider, and S. Stein. OWL web ontology language reference. Working draft, W3C, March 2003.
10. A. Doan, P. Domingos, and A.Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In Walid G. Aref, editor, *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, Santa Barbara, California, May 2001. ACM Press.
11. A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the eleventh international conference on World Wide Web*, pages 662–673. ACM Press, 2002.
12. Xin Dong, Alon Y. Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Simlarity search for web services. In *VLDB*, pages 372–383, 2004.
13. D. Dori. *Object-Process Methodology - A Holistic Systems Paradigm*. Springer Verlag, 2002.
14. D. Dori. Visweb - the visual semantic web: unifying human and machine knowledge representations with object-process methodology. *VLDB*, 13(2):120–147, 2004.
15. D. Dori, E. Toch, and I. Reinhartz-Berger. Modeling semantic web services with opm/s a human and machine-interpretable language. In *Third International Workshop on Web Dynamics, WWW 2004*, New York, 2004.
16. N. Fridman Noy and M.A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, pages 450–455, Austin, TX, 2000.
17. A. Gal, A. Anaby-Tavor, A. Trombetta, and D. Montesi. A framework for modeling and evaluating automatic semantic reconciliation. *VLDB Journal*, 2004. to appear.
18. A. Gal, G. Modica, H.M. Jamil, and A. Eyal. Automatic ontology matching using application semantics. *AI Magazine*, 2004. to appear.

19. Shahram Ghandeharizadeh, Craig A. Knoblock, Christos Papadopoulos, Cyrus Shahabi, Esam Alwagait, José Luis Ambite, Min Cai, Ching-Chien Chen, Parikshit Pol, Rolfe R. Schmidt, Saihong Song, Snehal Thakkar, and Runfang Zhou. Proteus: A system for dynamically composing and intelligently executing web services. In *ICWS*, pages 17–21, 2003.
20. T.R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
21. B. He and K. Chen-Chuan Chang. Statistical schema matching across Web query interfaces. In *Proceedings of the ACM-SIGMOD conference on Management of Data (SIGMOD)*, pages 217–228, San Diego, California, United States, 2003. ACM Press.
22. R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 51–61. ACM Press, 1997.
23. Mark Klein and Abraham Bernstein. Towards high-precision service retrieval. *IEEE Internet Computing*.
24. G. Koifman, A. Gal, and O. Shehory. Schema mapping verification. In H. Davulcu and N. Kushmerick, editors, *Proceedings of the VLDB-04 Workshop on Information Integration on the Web*, pages 52–57, Toronto, Canada, August 2004.
25. J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. In *Proceedings of the International conference on very Large Data Bases (VLDB)*, pages 49–58, Rome, Italy, September 2001.
26. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the IEEE CS International Conference on Data Engineering*, pages 117–140, 2002.
27. Mourad Ouzzani and Athman Bouguettaya. Efficient access to web services. *IEEE Internet Computing*, 8(2):34–44, 2004.
28. Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347, 2002.
29. A. Patil, S. Oundhakar, A. Sheth, and K. Verma. Meteor-s web service annotation framework. In *Proceedings of WWW 2004*, pages 553–562, New York, NY, May 2004.
30. M. Andrea Rodríguez and Max J. Egenhofer. Determining semantic similarity among entity classes from different ontologies. *IEEE Trans. Knowl. Data Eng.*, 15(2):442–456, 2003.
31. P. Rodriguez-Gianolli and J. Mylopoulos. A semantic approach to XML-based data integration. In *Proc. of the International Conference on Conceptual Modelling (ER'01)*, pages 117–132, Yokohama, Japan, 2001. Lecture Notes in Computer Science, Springer-Verlag.
32. A.M. Zaremski and J.M. Wing. Specification matching of software components. *ACM Transactions on Software Engineering and Methodology*, 6(4):333–369, 1997.

Transforming Web Requirements into Navigational Models: AN MDA Based Approach

Pedro Valderas, Joan Fons, and Vicente Pelechano

Department of Information System and Computation, 46022 Cami de Vera s/n,
Technical University of Valencia, Spain
{pvalderas, jjfons, pele}@dsic.upv.es

Abstract. Model Driven Architecture (MDA) is being adopted as a new development strategy. MDA is based on both the definition of models at different levels of abstraction and the application of consecutive transformations in order to obtain code from these models. However, little methodological support is provided to both define and apply model-to-model transformations. In this work, we introduce a strategy based on graph transformations that allow us to automate the derivation of the navigational model of the OOWS method from early requirements specifications, by following an MDA-based development process. In order to define and apply the graph transformations the Attributed Graph Grammar tool is used. In addition, due to the OOWS PIM-to-Code transformation capabilities, we show how this strategy allows us to obtain prototypes of web applications from early requirements specifications.

1 Introduction

Model Driven Architecture (MDA) [8] is being adopted by a significant number of proposals that provide support for software development. MDA introduces a development process where three basic models are proposed: *computation-independent models* (CIMs) that focus on the requirements of the system, *platform independent models* (PIMs) that describe the system using high-level constructs that hide the necessary details of particular platforms, and *platform specific models* (PSMs) that are created from the PIM models by including specific platform details. Besides the specification of these models, MDA proposes that a set of consecutive transformations should be applied in order to transform these models into code. However, MDA does not specify any concrete technique for building software from models. In this sense, in the web applications development area, little methodological support is provided to both define and apply model-to-model transformations.

In this work, we introduce a strategy that is based on graph transformations in order to define and automatically apply model-to-model transformations. This strategy has been used to automate the derivation of the navigational model of the OOWS method [2] (PIM model) from the OOWS requirements model (CIM model). In order to do this, we use the Attribute Graph Grammar tool [6] that allows us to easily define and apply graph transformations. In addition, due to the OOWS PIM-to-Code strategy capabilities, the automatic transformation between the OOWS CIM and PIM models allows us to directly obtain web application prototypes from the CIM model where the early requirements of the web application are described. In this way, the contribution of this work is twofold:

- We provide a precise way to automatically perform model-to-model transformations. This contribution allows us to provide a fully MDA approach that supports automatic model transformation.
- We provide a mechanism to obtain fully operative web Application prototypes from a requirements model.

This paper is organized as follows: Section 2 presents a brief overview of the work related to MDA and web engineering. Section 3 presents the OOWS method development process based on MDA. Sections 4 and 5 present an overview of the OOWS requirements model and the OOWS navigational model respectively. Section 6 proposes a strategy to automatically perform a model-to-model transformation between these two models. Section 7 explains how web prototypes are obtained from the OOWS navigational model. Finally, conclusions are presented in Section 8.

2 MDA and the Web Engineering

In the context of web engineering, many different methods, some partly or fully based on UML [1] (see e.g. OOHDM [15], UWE [16], WSDM [17], OO-H [18], WebML [19]) and others based on more formal foundations (see e.g. WIS [20]), have been proposed for the specification of web applications. These methods provide analysts with a development process where several models are built to capture the different aspects (such as information, navigation and functionality) of a web application.

All these approaches can be easily adapted to fit the Model Driven Architecture. However, few works have been published where this adaptation is explicitly explained. OO-H proposes in [21] the definition of different models following the MDA standard in order to reduce the gap between software architectures and web application design. In [22] a behavioural semantics has been included within OOHDM to develop a valid PIM for MDA. Although these approaches focus on MDA model definitions (PIM model especially), model transformations are not explicitly presented. This work introduces guidelines and tools to both define and automatically perform a CIM to PIM transformation.

3 OOWS: AN MDA-Based Method for Web Application Development

This section presents a brief overview of the OOWS method [2] that follows the MDA proposal (see Figure 1). In the Computation-Independet Model (CIM), the early requirements of a web application are described. To do this, we define first a hierarchy of tasks that allows us to identify the tasks that users must achieve. Next, we describe these tasks from the interactions that users require of the web application.

Once the CIM model is defined, the OOWS PIM model is derived from the task-based CIM model by applying what MDA calls a model-to-model transformation. In the Platform Independent Model (PIM), OOWS proposes the definition of several models which describe the different aspects of a web application. The system static structure and the system behaviour are described in three models (*class diagram* and *dynamic* and *functional* models) that are borrowed from an object oriented software production method called OO-Method [10]. The navigational aspects of a web application are described in a *navigational model* [2].

Finally, we obtain code from the PIM models by performing what MDA call an automatic transformation [8]. The OOWS PIM model provides all the information needed for obtaining fully operative web application prototypes by means of automatic code generation [3]. This prototype is implemented by following a three-tier architecture. The information and functionality (*Application* and *Persistence* tier) of the web application is generated by the OlivaNova tool [4] from the OO-Method models (structural and behavioural model). The navigational structure (*Presentation* tier) of the web application is generated by the OOWS case tool following directives specified in design templates [2].

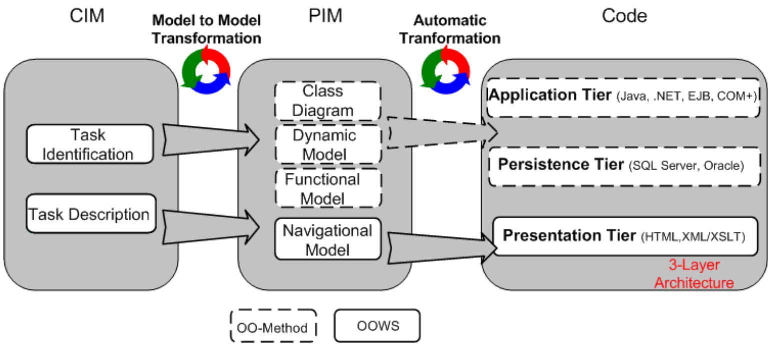


Fig. 1. The MDA-based OOWS method

This paper presents a strategy that allows us to automate the derivation of the PIM navigational model from the task-based CIM model. Information about how the PIM structural and behavioural models are obtained from a requirements specification is explained in [14]. In order to more easily understand the presented model-to-model transformation we introduce first a (necessary) brief overview of the task-based CIM model and the PIM navigational model. Finally, we introduce the OOWS strategy for obtaining code from the navigational model which allows us to obtain web application prototypes from early requirements specifications.

4 The CIM: A Task Description for the Early Requirements Specification

In the OOWS CIM model, the early requirements of a web application are described by: (1) identifying the set of tasks that users must perform and (2) describing these tasks from the system-user interaction.

4. 1 Task Identification

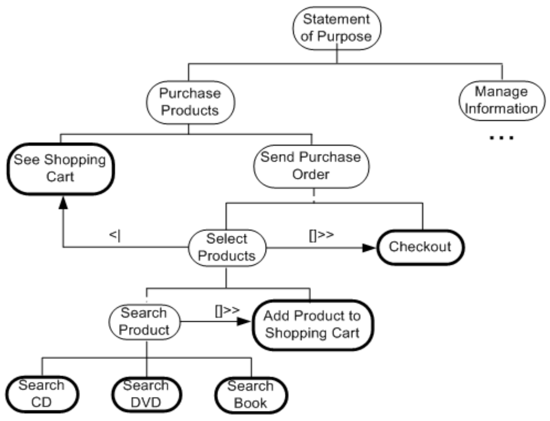


Fig. 2. Task taxonomy of the web sale application

To identify tasks, we propose the construction of a task taxonomy taking a statement of purpose that describes the goal for which the application is being built, as the starting point.

To define the task taxonomy, the statement of purpose is considered as the most general task. As the statement of purpose defines the general requirements of the system, these requirements can be grouped by means of semantic and/or functional relationships. They can also be grouped by analyst/stakeholder experience. This allows us to obtain an initial set of general tasks. These general tasks can be refined to obtain more specific ones until a group of *elementary tasks* are obtained. An *elementary task* is defined as a task that when divided into subtasks involves either the user or the system, but not both. In addition, we propose to enrich this taxonomy by indicating temporal relationships among tasks. To do this, we have used the relationships introduced by the CTT approach (ConcurTaskTree) [9].

Figure 2 shows a partial view of the task taxonomy that we obtain from the statement of purpose of a web application for the sale of audiovisual and bibliographic products (CDs, DVDs and books). In order to easily identify the elementary tasks they are circled with a thicker line. The user's needs that are described in the statement of purpose have been grouped into two tasks: *Purchase Products* and *Information Management*. The *Purchase Products* task is refined into the *See Shopping Cart* elementary task and the *Send Purchase Order* task. From the *Send Purchase Order* task, consecutive refinements are applied until the task taxonomy of Figure 2 is obtained. Furthermore, according to the temporal relationships defined in the task taxonomy (see arrows of Figure 2) the *Select Products* task must always be performed before the *Checkout* task ([]>> *Enabling* relationship [9]). In addition, during the *Select Products* task the user can suspend it (resuming it after) to see the state of his/her shopping cart (<|, *Suspend/Resume* relationship [9]). Finally, the *Search Product* task must be always performed before the *Add Product to Shopping Cart* elementary task ([]>> *Enabling* relationship [9]).

4.2 Task Description: Activity Diagrams and Interaction Points

Once the task taxonomy is built, elementary tasks are described to define how user can achieve them. To do this, we extend traditional description where user and system actions are described by indicating explicitly when (at which exact moment) the interaction between user and system is performed. In this way, we introduce the concept of **interaction points (IP)**. In each IP, the system provides the user with information that is related to an entity[1]. Moreover, access to operations can also be provided. In this sense, with both the information and the operations the user can perform several actions: he/she can select information (as a result the system provides the user with new information) and he/she can activate an operation (as a result the system carries out an action).

To perform descriptions of this kind, we propose a graphical notation based on UML *activity diagrams* [1] (see Figure 3). Each node (activity) represents an IP (solid line) or a system action (dashed line). Furthermore, the number of information instances[2] that the IP includes (cardinality) is depicted as a small circle in the top right side of the primitive. Finally, each arc represents (1) a user action (selection of information or operations) if the arc source is an IP or (2) a node sequence if the arc source is a system action. Figure 3 shows the description of the *Search CD* elementary task (the shaded numbers are not part of the notation). According to Figure 3: (1) The *CD Search* task starts with an IP where the system provides the user with a list (cardinality *) of music categories. (2) The user selects a category. (3) The task continues with an IP where the system informs about the CDs of the selected category. Depending on the user action there are two ways (A and B) to continue with the task: **A)** (4a) The user selects a CD. **(5a)** The task ends with an IP where the system provides the user with a description of the selected CD. **B)** (4b) The user selects a search operation. (5b) The system performs a system action which searches the CDs of an artist. (6b) Since the search result is a list of CDs it is shown in the IP where the full list of CDs is previously shown. Then the task continues in point 3.

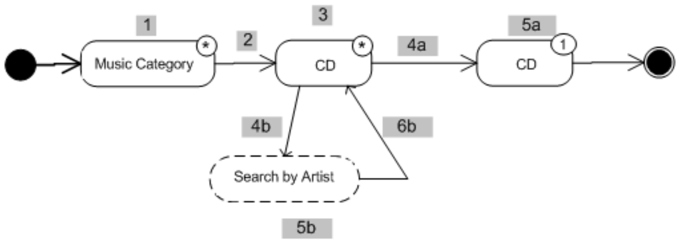


Fig. 3. *CD Search* elementary task

As we can see, details about the information exchanged between the user and the system are not described (we just indicate the entity to which the information is related). To do this, we propose a technique based on information templates that is introduced next.

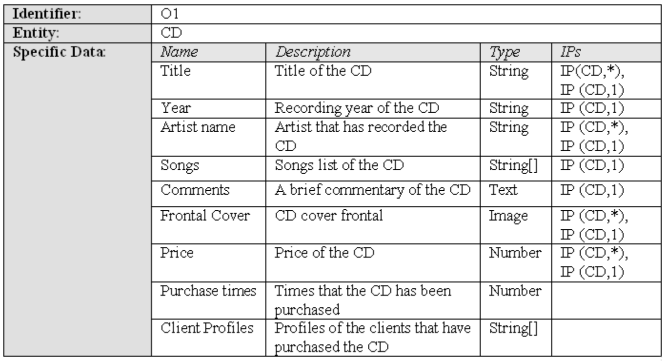
Describing the system data. The information that must be stored in the system is defined by means of a template technique that is based on data techniques such as the CRC Card [23]. We propose the definition of an information template (see Figure 4)

[1] Abstraction of any object of the real world that belongs to the system domain (e.g. client, product, invoice, etc.).

[2] Given a system entity (e.g. client), an information instance is considered to be the set of data related to each element of this entity (Name: Joseph, Surname: Elmer, Telephone Number: 9658789).

for each entity identified in the description of a task. In each template we indicate an identifier, the entity and a *specific data* section. In this section, we describe the information in detail by means of a list of specific features associated to the entity. For each feature we provide a name, a description and a data type. In addition, we use these templates to indicate the information shown in each IP. For each feature we indicate the IPs where it is shown (if there is any). To identify an IP we use the notation: IP *(Entity, Cardinality)*.

According to the template in Figure 4, the information that the system must store about a CD is (see the specific data section): the CD title, the artist name, the front cover and the price that are shown in the IP (CD,1) and the IP (CD,*)[3]; the recording year, some comments about the CD and the list of songs that are shown only in the IP (CD,1); and finally, the times that a CD has been bought and the client profiles that usually purchase it which are not shown in any IP.



Identifier:	O1			
Entity:	CD			
Specific Data:	*Name*	*Description*	*Type*	*IPs*
	Title	Title of the CD	String	IP(CD,*), IP (CD,1)
	Year	Recording year of the CD	String	IP (CD,1)
	Artist name	Artist that has recorded the CD	String	IP (CD,*), IP (CD,1)
	Songs	Songs list of the CD	String[]	IP (CD,1)
	Comments	A brief commentary of the CD	Text	IP (CD,1)
	Frontal Cover	CD cover frontal	Image	IP (CD,*), IP (CD,1)
	Price	Price of the CD	Number	IP (CD,*), IP (CD,1)
	Purchase times	Times that the CD has been purchased	Number	
	Client Profiles	Profiles of the clients that have purchased the CD	String[]	

Fig. 4. An information template

Task descriptions allow us to define a computation-independent view of the system (CIM model). Following the MDA strategy, a platform-independent view (PIM model) must be obtained from the CIM model.

5 The PIM: The Navigational Model

The OOWS navigational model is represented by a directed graph (which defines the navigational structure) whose nodes are *navigational contexts* and its arcs denote *navigational links* (see Figure 5A). A navigational context (represented by a UML package stereotyped with the *«context»* keyword) defines a view on the class diagram that allows us to specify an information retrieval. A navigational link represents navigational context reachability: the user can access a navigational context from a different one if a navigational link between both has been defined.

A navigational context is made up of a set of *navigational classes* that represent class views over the classes of the class diagram (including attributes and operations).

[3] IPs defined in the *Search CD* elementary task, see Figure 3.

Each navigational context has one mandatory navigational class, called a *manager class* and optional navigational classes to provide complementary information of the manager class, called *complementary classes*. All navigational classes must be related by unidirectional binary relationships, called *navigational relationships* that are defined upon an existent relationship in the class diagram. Figure 5B shows the *CD* navigational context which is made up of the manager navigational class (CD) and a complementary navigational class (Artist). These classes are related by means of a navigational relationship. This navigational context provides the user with the CD title, the artist's name, the recording year, the songs, the front cover, the price and some commentaries about the CD (navigational classes attributes). In addition, the user can invoke the *Add_to_Cart* functionality to add the CD to the shopping cart (manager class operation).

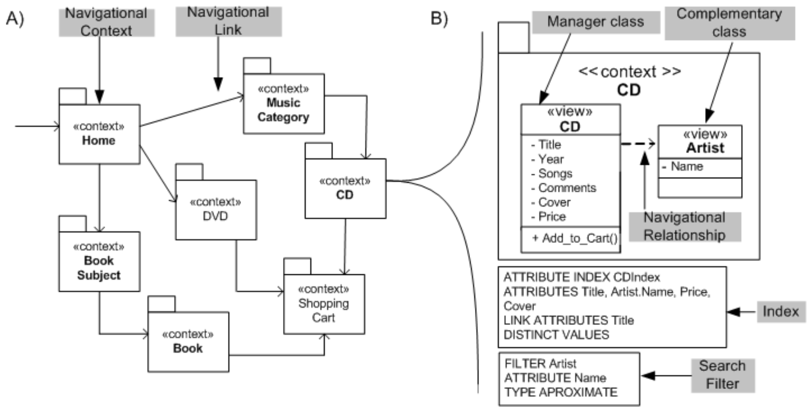


Fig. 5. The OOWS navigational model for the web sale application

Moreover, for each context, we can also define: (1) *Search filters* that allow us to filter the space of objects that retrieve the navigational context. The CD navigational context allows the user to find all the CDs of a specific artist (see search filter in Figure 5B). (2) *Indexes* that provide an indexed access to the population of objects. Indexes create a list of summarized information allowing the user to choose one item (instance) from the list. This selection causes this instance to become active in the navigational context. The CD navigational context provides the user with a list of CDs where the CD title, the artist name, the price and the cover are shown (see index in Figure 5B).

6 From Task Descriptions to the OOWS Navigational Model

In this section, we present a strategy based on graph transformations that allow us to automatically derive the OOWS navigational model from task descriptions. This strategy is divided into two main stages:

1. *Definition of mapping rules*: We must identify the set of mappings between the source-model primitives and the target-model primitives that allow us to achieve the model-to-model transformation. Section 6.1 introduces a graph technique to define mapping rules.
2. *Application of mapping rules*: Once mapping rules are defined (by means of graph transformations) they must be applied into the source model in order to obtain the target model. Section 6.2 introduces a strategy based on the AGG tool [6] that allows us to automatically apply graph transformations.

6.1 Graph Transformations: Defining the Mapping Rules

Graph transformations are specified using transformation systems. Transformation systems rely on the theory of graph grammars [12]. A transformation system is composed of several transformation rules. A rule is a graph rewriting rule equipped with negative application conditions and attribute conditions [13]. Figure 6 illustrates how a transformation system is applied to a graph G: when (1) a Left Hand Side (LHS) matches into G and (2) a Negative Application Condition (NAC) does not match into G (3) the LHS is replaced by a Right Hand Side (RHS). G is transformed into G′. All elements of G not covered by the match are considered as unchanged. All elements contained in the LHS and not contained in the RHS are considered as deleted. To add more expressiveness to transformation rules, variables may be associated to attributes within a LHS. These variables are initialized in the LHS and their value can be used to assign an attribute in the expression of the RHS. An expression may also be defined to compare a variable declared in the LHS with a constant or with another variable. This mechanism is called *attribute condition*.

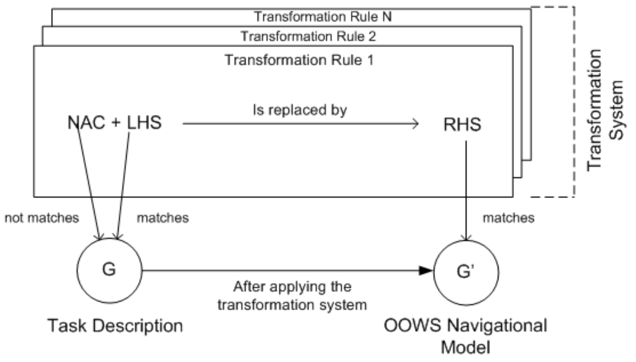


Fig. 6. A transformation system

We have chosen a graph transformation technique because it is (1) **Visual**: every element within a graph transformation based language has a graphical syntax; (2) **Formal**: a graph transformation is based on a sound mathematical formalism (algebraic definition of graphs and category theory) and enables verifying formal properties on represented artifacts; (3) **Seamless**: it allows representing manipulated artifacts and rules within a single formalism.

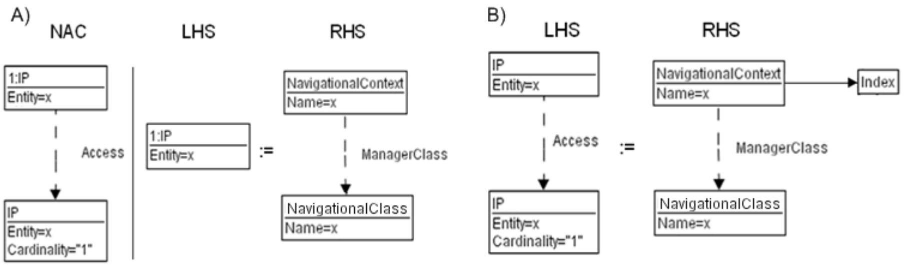


Fig. 7. Graph transformation rules

In order to derive the OOWS navigational model from a task description we have defined a set of transformation rules. Due to space constraints, we only present two representative rules that can be seen in Figure 7. The rest of rules are identified in [5]. The rule of Figure 7A says that when an IP (LHS) is found it must be transformed into a Navigational Context with its Manager Navigational Class (RHS). However, this rule is not applied if the IP provides access to another IP that provides the user with information about one instance of the same entity (NAC). These cases are handled by the rule of Figure 7B whose RHS defines in the target graph a Navigational Context with its Manager Navigational Class and an Index (index attributes are defined by other rule). In addition, the variable *x* that has been defined in the IPs of the LHS of both rules, gives the name to the Navigational Context and the Navigational Class included in the RHS.

6.2 Automatically Applying Graph Transformations

Once mapping rules have been defined by means of graph transformations they must be applied into the source model (in our case a task description) in order to transform it into the target model (in our case an OOWS navigational model). To do this automatically, we propose the use of the Attribute Graph Grammar (AGG) tool [6].

The AGG tool allows us to automatically transform a source *graph* into a target *graph* by applying graph transformations. We just need to load the source graph into the AGG tool, introduce the definition of the graph transformations and then, the system automatically applies the rules to obtain the target graph.

However, models are not always defined as *graphs*. For instance, the OOWS CIM and PIM models are not graphs. In these cases, in order to fully automate the model-to-model transformation by means of the AGG tool [6] several steps must be fulfilled (see Figure 8): (1) First, the source model must be represented as a correct AGG graph. (2) Next, this graph is loaded in the AGG tool and the graph transformations are applied. Once transformation is finished, the obtained graph represents the target model. (3) Finally, the this final graph must be translated into the correct target modelling language.

Next, we present a strategy to automatically achieve these steps. It has been used to allow the automatic derivation of the OOWS navigational model from task descriptions.

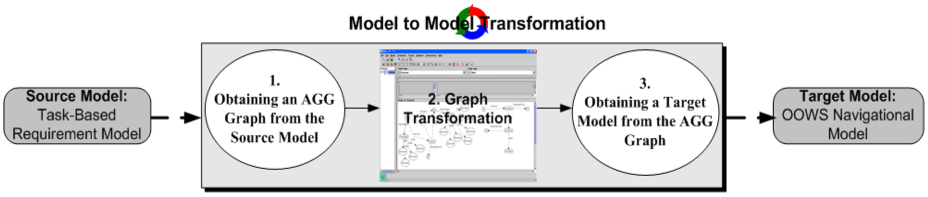


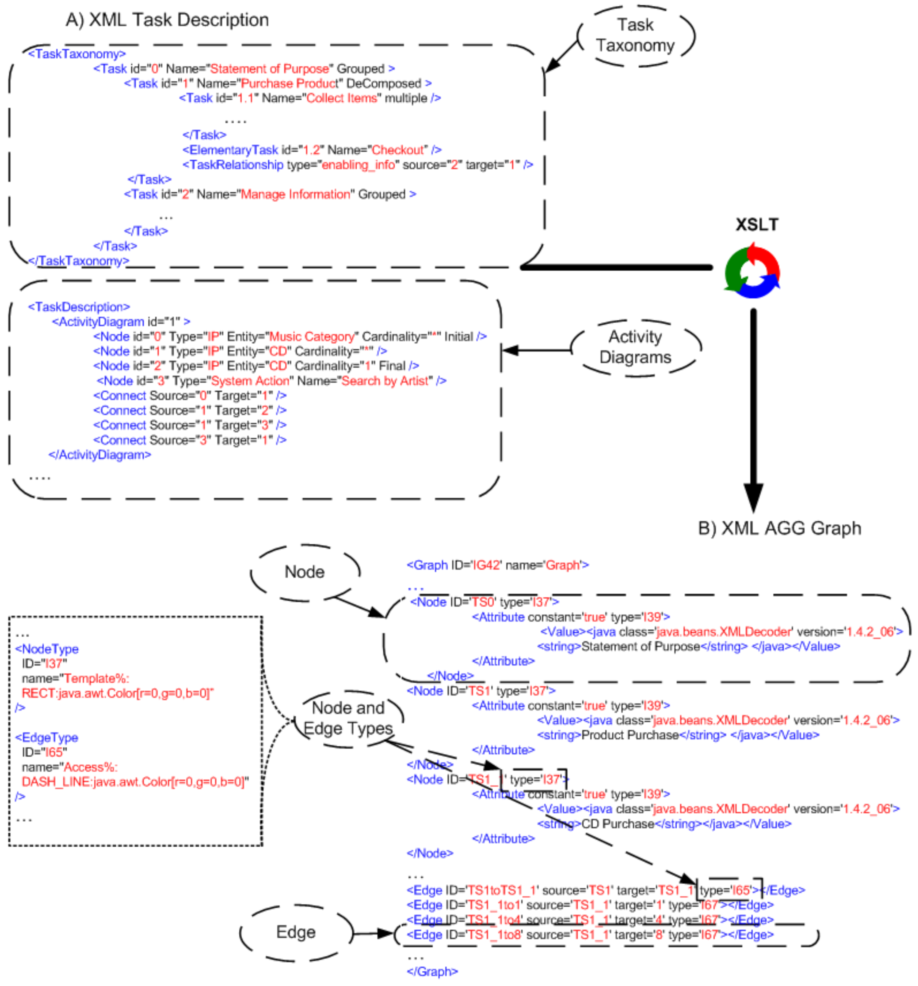
Fig. 8. MDA transformation steps

Obtaining an AGG Graph from Task Descriptions. To automate the transformation of a task description into an AGG graph we propose a strategy based on translating XML documents. On one hand, the CIM model should be specified in an XML document (Figure 9A shows a partial view of the XML specification of the task description presented in this work). This XML specification is built by means of XML elements that are defined from the task elements presented in Section 4. On the other hand, the AGG system also uses XML to store its graphs by means of four XML elements: the *NodeType* and *EdgeType* elements that allow us to specify which kind of nodes and edges can be defined in the graph; and the *Node* and *Edge* elements that allow us to define the graph. The *NodeType* and *EdgeType* elements are defined by means of a name and a set of graphical representation properties. The *Node* and *Edge* elements are defined by means of a name, a set of attributes and a type. The type is a reference to a previously defined *NodeType* or *EdgeType* element. Figure 9B shows a partial view of an XML AGG graph.

As Figure 9 shows, we obtain an XML AGG graph from an XML task description by means of an XSL transformation [11]. This transformation is performed by following the next steps (see Figure 10):

(1) We transform the task taxonomy into an AGG graph: First, tasks are transformed into graph nodes. Next, refinement and temporal relationships define graph edges. Although the task taxonomy is basically used to identified elementary tasks we represent the whole taxonomy as a graph to extract (during the transformation process) navigational semantics from the temporal relationships.
(2) Each activity diagram is represented as an AGG graph: IPs and System Actions are transformed into graph nodes. Activity diagram's arcs define graph edges.
(3) These defined AGG graphs are joined into a single graph: each node that represents an elementary task is connected to the node that represents the initial IP or System Action of the activity diagram that describes this task.
(4) Information templates are represented as AGG graphs: We define a graph node for each information template. Next, each template's entity feature is represented as a node which is connected to the corresponding template node.
(5) Finally, these new AGG graphs are connected to the graph obtained in point 3: each node that represents an entity feature is connected to the node that represents the IP where it is shown.

Figure 10 shows the graphical representation of the AGG graph that we obtain by applying the XSLT transformation into the task description presented in this work.



```
<TaskTaxonomy>
        <Task id="0" Name="Statement of Purpose" Grouped >
            <Task id="1" Name="Purchase Product" DeComposed >
                  <Task id="1.1" Name="Collect Items" multiple />
                        ....
                  </Task>
                  <ElementaryTask id="1.2" Name="Checkout" />
                  <TaskRelationship type="enabling_info" source="2" target="1" />
            </Task>
            <Task id="2" Name="Manage Information" Grouped >
                  ...
            </Task>
        </Task>
</TaskTaxonomy>
```

```
<TaskDescription>
    <ActivityDiagram id="1" >
        <Node id="0" Type="IP" Entity="Music Category" Cardinality="*" Initial />
        <Node id="1" Type="IP" Entity="CD" Cardinality="*" />
        <Node id="2" Type="IP" Entity="CD" Cardinality="1" Final />
        <Node id="3" Type="System Action" Name="Search by Artist" />
        <Connect Source="0" Target="1" />
        <Connect Source="1" Target="2" />
        <Connect Source="1" Target="3" />
        <Connect Source="3" Target="1" />
    </ActivityDiagram>
....
```

```
...
<NodeType
  ID="I37"
  name="Template%:
  RECT:java.awt.Color[r=0,g=0,b=0]"
/>

<EdgeType
  ID="I65"
  name="Access%:
  DASH_LINE:java.awt.Color[r=0,g=0,b=0]"
/>

...
```

```
<Graph ID='IG42' name='Graph'>
...
 <Node ID='TS0' type='I37'>
          <Attribute constant='true' type='I39'>
                  <Value><java class='java.beans.XMLDecoder' version='1.4.2_06'>
                  <string>Statement of Purpose</string> </java></Value>
          </Attribute>
    </Node>
<Node ID='TS1' type='I37'>
          <Attribute constant='true' type='I39'>
                  <Value><java class='java.beans.XMLDecoder' version='1.4.2_06'>
                  <string>Product Purchase</string> </java></Value>
          </Attribute>
</Node>
<Node ID='TS1_1' type='I37'>
          <Attribute constant='true' type='I39'>
                  <Value><java class='java.beans.XMLDecoder' version='1.4.2_06'>
                  <string>CD Purchase</string></java></Value>
          </Attribute>
</Node>
...
<Edge ID='TS1toTS1_1' source='TS1' target='TS1_1' type='I65'></Edge>
<Edge ID='TS1_1to1' source='TS1_1' target='1' type='I67'></Edge>
<Edge ID='TS1_1to4' source='TS1_1' target='4' type='I67'></Edge>
<Edge ID='TS1_1to8' source='TS1_1' target='8' type='I67'></Edge>
...
</Graph>
```

Fig. 9. Translation of XML documents

Graph Transformations on the AGG tool. AGG (Attributed Graph Grammars tool) [6] can be considered to be a genuine programming environment based on graph transformations. It provides 1) a programming language enabling the specification of graph grammars and 2) a customizable interpreter enabling graph transformations. AGG was chosen because it allows the graphical expression of directed, typed and attributed graphs (for expressing specifications and rules). It has a powerful library containing notably algorithms for graph transformation, critical pair analysis, consistency checking and application of positive and negative conditions.

Figure 11 shows a snapshot of the AGG user interface. Frame 1 is the grammar explorer. Frames 2, 3 and 4 enable to specify sub-graphs composing a production: a negative application (frame 2), a left hand side (frame 3) and a right hand side (frame 4). The host graph on which a production will be applied is represented in frame 5.

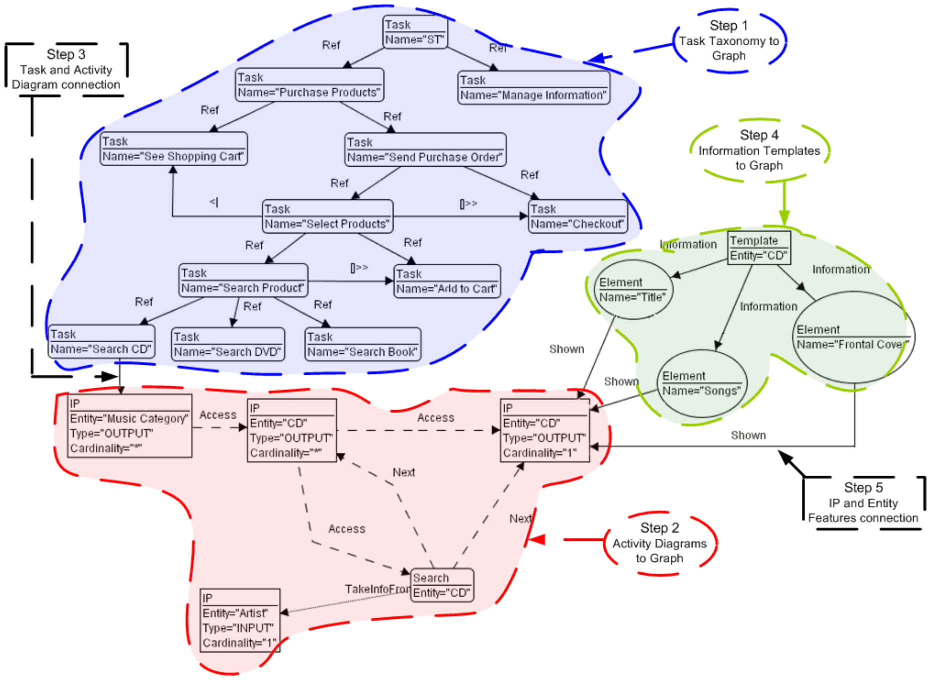
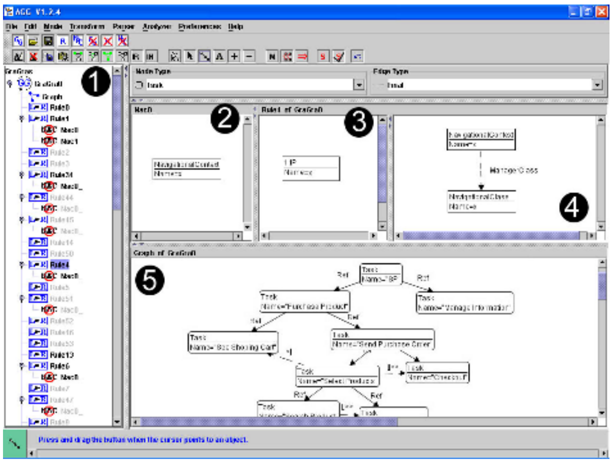


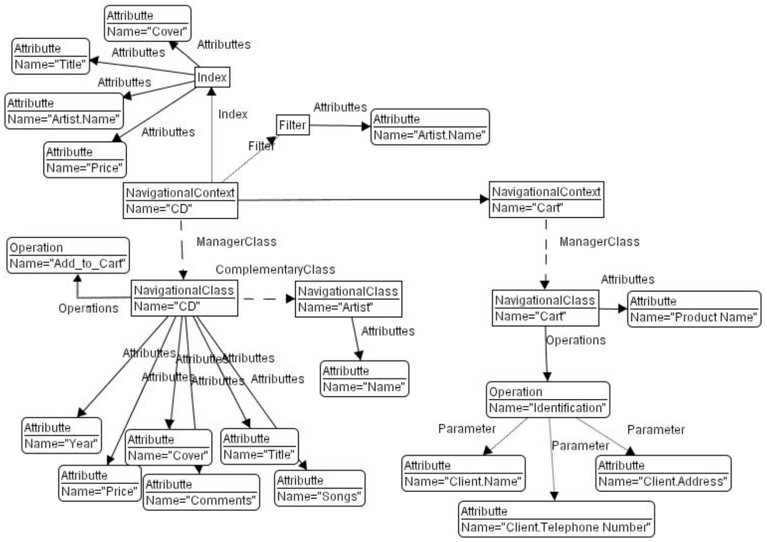
Fig. 10. Representation of a task description by means of an AGG graph



Press and drag this button when the cursor points to an object.

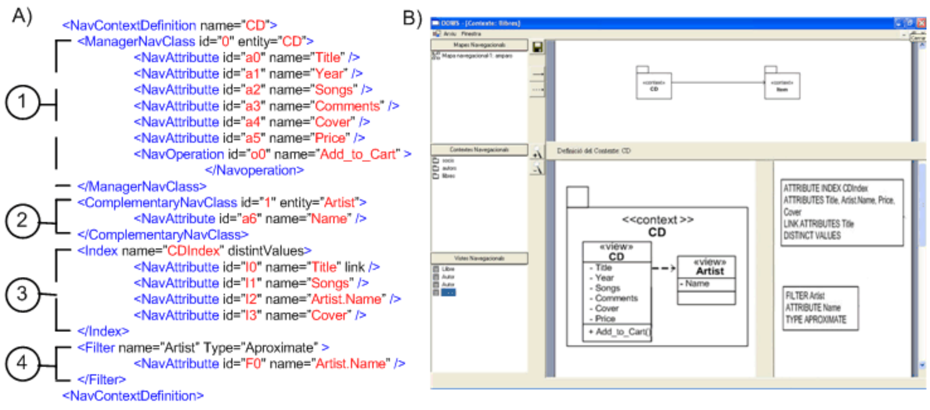
Fig. 11. AGG user interface

Obtaining an OOWS Navigational Model from an AGG Graph. When the AGG system transformation is finished, a graph that represents an OOWS navigational model is obtained. This graph is made up of a set of nodes and edges defined from the OOWS metamodel elements. These elements have been explained in Section 5.



Attributte Name="Cover"
Attributte Name="Title"
Attributtes
Index
Attributte Name="Artist.Name"
Attributte Name="Price"
Filter
Attributte Name="Artist.Name"
NavigationalContext Name="CD"
NavigationalContext Name="Cart"
ManagerClass
ComplementaryClass
Operation Name="Add_to_Cart"
Operations
NavigationalClass Name="CD"
NavigationalClass Name="Artist"
Attributte Name="Name"
NavigationalClass Name="Cart"
Attributte Name="Product Name"
Operation Name="Identification"
Parameter
Attributte Name="Year"
Attributte Name="Cover"
Attributte Name="Title"
Attributte Name="Price"
Attributte Name="Comments"
Attributte Name="Songs"
Attributte Name="Client.Name"
Attributte Name="Client.Address"
Attributte Name="Client.Telephone Number"

Fig. 12. An OOWS specification represented by means of an AGG graph


A)

```
<NavContextDefinition name="CD">
  <ManagerNavClass id="0" entity="CD">
        <NavAttributte id="a0" name="Title" />
        <NavAttributte id="a1" name="Year" />
        <NavAttributte id="a2" name="Songs" />
        <NavAttributte id="a3" name="Comments" />
        <NavAttributte id="a4" name="Cover" />
        <NavAttributte id="a5" name="Price" />
        <NavOperation id="o0" name="Add_to_Cart" >
                 </Navoperation>
  </ManagerNavClass>
  <ComplementaryNavClass id="1" entity="Artist">
        <NavAttribute id="a6" name="Name" />
  </ComplementaryNavClass>
  <Index name="CDIndex" distintValues>
        <NavAttributte id="I0" name="Title" link />
        <NavAttributte id="I1" name="Songs" />
        <NavAttributte id="I2" name="Artist.Name" />
        <NavAttributte id="I3" name="Cover" />
  </Index>
  <Filter name="Artist" Type="Aproximate" >
        <NavAttributte id="F0" name="Artist.Name" />
  </Filter>
<NavContextDefinition>
```

B)

Fig. 13. An OOWS XML specification and the OOWS case tool

Figure 12 shows a partial view of the OOWS graph that has been obtained from the CIM model of the “Web Sale” application after applying the full set of transformation rules. According to this figure, two navigational contexts are defined: the *CD* navigational context and the *Cart* navigational context. On one hand, the *CD* navigational context is made up of a manager class (with some attributes and an operation) and a complementary class (with an attribute). An index and a search filter are also defined in this navigational context. On the other hand, the *Cart* navigational context is made up of the manager class with an attribute and an operation (with its parameters). A link is defined between both contexts.

The OOWS XML document can be loaded by our OOWS case tool. This tool provides us with a graphical representation of navigational specifications. Figure 13B shows the OOWS tool after loading the OOWS XML document obtained from the graph shown in Figure 12. On the upper side we can see the navigational structure section. This figure shows the contexts (*CD* and *Cart*) resulting from the example described in the paper. A link relationship between both contexts is also defined. On the lower side we can see the context definition section where the *CD* navigational context definition is shown. This section shows both the navigational classes that made up the navigational context and the search mechanisms (indexes and filers). In order to see the definition of other navigational context we just need to select it by clicking on the navigational context in the navigational structure section.

7 Generating a Web Application Prototype from the PIM Model

Starting from the navigational model, a web application prototype can be automatically generated by the OOWS case tool. This prototype is made up of a group of connected web pages that define the web application user interface for navigating, visualizing the data and accessing to the web application functionality. Presentation aspects such as colours, sizes or fonts are not considered in the prototype construction. In later stages, presentation guidelines should be applied in order to satisfy client presentation requirements and obtain the final web application. We show an example of how web pages are obtained from the navigational model presented in this work (see Figure 5). Web pages are presented after applying a default presentation guideline.

A web page is created for each navigational context in the navigational map. Each web page is responsible for retrieving the specified information in its navigational context by requesting it from the application tier (which is generated by the OlivaNova tool [4]). Furthermore, for each index defined in a context, an additional web page that provides the user with a list of instances with the specified information

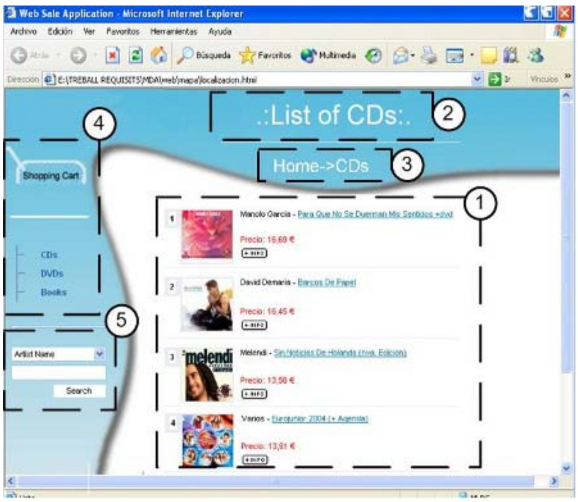
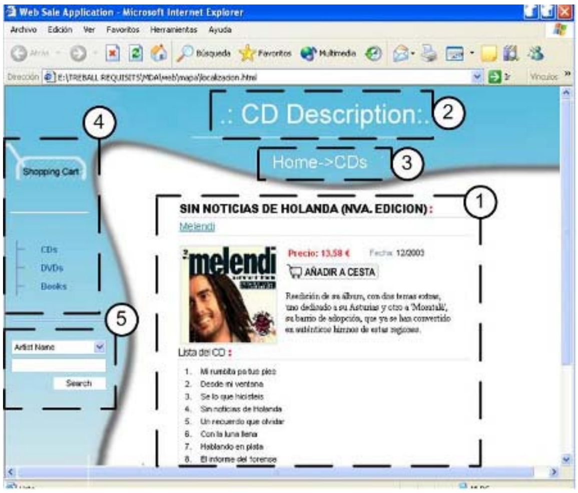

Web Sale Application - Microsoft Internet Explorer
.:List of CDs:.
Home->CDs
Shopping Cart
CDs
DVDs
Books
Artist Name
Search
Manolo García - Para Que No Se Duerman Mis Sentidos +dvd
Precio: 15,60 €
David Demaría - Barcos De Papel
Precio: 15,45 €
Melendi - Sin Noticias De Holanda (nva. Edición)
Precio: 13,50 €
Varios - Eurojunior 2004 (+ Agenda)
Precio: 13,91 €
1
2
3
4
5

Fig. 14. Web page generated from the index defined in the CD navigational context



Reedición de su álbum, con dos temas extras, uno dedicado a su Asturias y otro a 'Montalé', su barrio de adopción, que ya se han convertido en auténticos himnos de estas regiones.

1. Mi rumbita pa tus pies
2. Desde mi ventana
3. Se lo que hicisteis
4. Sin noticias de Holanda
5. Un recuerdo que olvidar
6. Con la luna llena
7. Hablando en plata
8. El informe del forense

Fig. 15. Web Page for the CD context

(in the index definition) is implemented. Figure 14 shows the web page generated from the index of the navigational context CD (see Figure 5B).

Selecting one of these indexed instances (using the title of the CD as the link attribute), the web page shows all the information specified in the context. The web page of Figure 15 presents the CD context. It shows the CD title, the artist's name, the recording year, the songs, the front cover, the price and some commentaries about the CD (see CD context definition in Figure 5B).

The strategy that we apply to generate web pages (see Figures 14 and 15 as an example) divides them into two logical areas:

– The *information* area presents the specific system view defined by a context. This area is located at the right side of the web pages (see box number 1).
– The *navigation* area provides navigation meta-information to the user, in order to improve some aspects of the quality (usability) of the final application [7]:
  • Where the user is (see box number 2). It is stated which is the web page that is being shown to the user currently.
  • How the user reached this context (see box number 3). It is shown the navigational path that has been followed to reach that page.
  • Where the user can go (see box number 4). A link to each navigational page where the user can access appears in this area.
  • Which filters can be used by the user (see boxes number 5). If the context has defined any filter mechanism, it is shown in this logical area.

8 Conclusions

A strategy to automate model-to-model transformation following the MDA approach has been introduced in this work. This strategy has been used to automate the CIM-to-PIM transformation of the OOWS method. In this sense, two goals have been achieved.

- The derivation of the OOWS navigational model (PIM) from the OOWS requirements model (CIM) has been automated by means of graph transformations. First, the CIM model is represented as a graph. Next this graph is transformed into an OOWS graph by means of the AGG tool. Finally, the OOWS graph is translated into an OOWS navigational model by means of an XML translation. This contribution allows us to provide a fully MDA approach that supports automatic model transformation.
- Furthermore, the PIM-to-Code strategy followed by the OOWS method lets us obtain web prototypes automatically from the CIM model. These prototypes are valuable tools to help users understand the captured requirements.

Finally, this proposal has been put into practice successfully in the development of small and medium-size web applications, including the *DSIC Department Web Site* (http://www.dsic.upv.es), the *OOMethod Group Web Site* (http://oomethod.dsic.upv.es) and the web application of the *Development Cooperation Center* (http://www.upv.es/ccd).

References

1. Object Management Group. Unified Modeling Language (UML) Specification Version 2.0 Final Adopted Specification. www.omg.org, 2003.
2. J. Fons, V. Pelechano, M. Albert, and O. Pastor. Development of Web Applications from Web Enhanced Conceptual Schemas. In Workshop on Conceptual Modeling and the Web, ER'03, volume 2813 of Lecture Notes in Computer Science. Springer, 2003.
3. R. Quintero,V. Pelechano, O. Pastor, J. Fons, Aplicación de MDA al Desarrollo de Aplicaciones Web en OOWS, pp. 379 - 388, Jornadas de Ingeniería de Software y Base de Datos (JISBD), VIII, 2003 - November, Alicante (Spain), 84-668-3836-5, 2003.
4. OlivaNova Model Execution System. CARE Technologies (www.care-t.com).
5. P. Valderas. Capturing Web Application Requirements. Spanish. Technical report, DSIC, Technical University of Valencia, February 2005. http://oomethod.dsic.upv.es.
6. The Attributed Graph Grammar System v1.2.4. http://tfs.cs.tu-berlin.de/agg/. 2004
7. L. Olsina: Metodologia Cuantitativa para la Evaluacion y Comparacion de la Calidad de Sitios Web. PhD thesis, Facultad de Ciencias Exactas de la Universidad Nacional de La Plata (1999) In spanish.
8. Object Management Group. Model Driven Architecture (MDA). www.omg.org/mda, 2004.
9. F. Paternò, C. Mancini and S. Meniconi, 1997. "ConcurTaskTrees: a Diagrammatic Notation for Specifying Task Models", In Proceedings of INTERACT'97, Chapman & Hall, 362-369.
10. O. Pastor, J. Gómez, E. Insfran, V. Pelechano. The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26 (2001) 507-534
11. XSL Tranformtations (XSLT) v. 1.0. http://www.w3.org/TR/xslt.
12. G. Rozenberg (ed.). Handbook of Graph Grammars and Computing by Graph Transformation. World Scientific, Singapore (1997)
13. H. Partsch, R. Steinbruggen: Program Transformation Systems. ACM Computing Surveys 15,3 (September 1983), 199–236.

14. E. Insfrán, O. Pastor and R. Wieringa, *Requirements Engineering-Based Conceptual Modelling*. Journal "Requirements Engineering" (RE), March 2002. 7(2): p. 61-72. Springer-Verlag. ISSN: 0947-3602 (printed version) ISSN: 1433-010X (electronic version).
15. D. Schwabe, G. Rossi, and S. Barbosa. Systematic Hypermedia Design with OOHDM. In ACM Conference on Hypertext, Washington, USA, 1996.
16. N. Koch. Software Engineering for Adaptive Hypermedia Applications. PhD thesis, Ludwig-Maximilians-University, Munich, Germany, 2000.
17. O. De Troyer and C. Leune. WSDM: A User-centered Design Method for Web sites. In World Wide Web Conference, 7th International Conference, WWW'97, pages 85,94, 1997.
18. J. Gómez, C. Cachero, O. Pastor. Extending an Object-Oriented Conceptual Modelling Approach to Web Application Design. Kista, Stockholm June 2000. CAiSE'2000, LNCS 1789, Pags 79-93.
19. S. Ceri, P. Fraternali, A. Bongio. Web Modeling Language (WebML): a Modeling Language for Designing Web Sites. In: Proc. of the 9th International World Wide Web Conference, WWW9, Elsevier (2000) 137-157.
20. K.-D. Schewe and B. Thalheim. Conceptual modelling of web information systems. Data and Knowledge Engineering, 2005.
21. S. Meliá, C. Cachero, J. Gómez. Using MDA in Web Software Architectures. 2nd International Workshop on Generative Techniques in the Context of MDA. Anaheim, California. USA. October 2003.
22. H. Albrecht. Model Driven Architecture with OOHDM. International Workshop on Web Engineering (IWWOST) 2004. Munich. Germany. Pp 1-10.
23. Wirfs-Brock, B. Wilkerson, and L. Wiener. *Designing Object–Oriented Software*. Prentice–Hall, 1990.

Accelerating Workflows with Fixed Date Constraints

Martin Bierbaumer, Johann Eder, and Horst Pichler

Institute for Informatics-Systems, University of Klagenfurt, Austria
{bierbaumer, eder, pichler}@uni-klu.ac.at

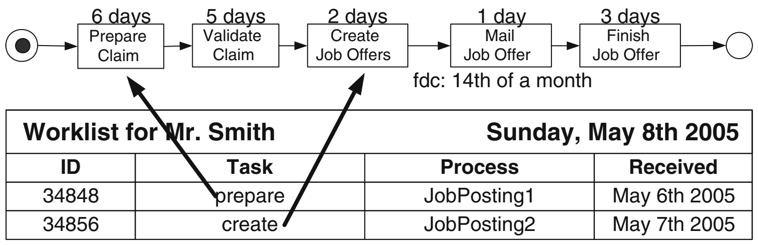
Abstract. Workflow systems execute workflows and assign work items to the work list of participants. As work lists usually hold multiple work items, participants have to decide which work item to handle next. When selecting a specific work item other work items must be postponed, which will in succession delay their appendant workflows. This may lead to disproportionately increased execution durations and turn around times if fixed-date constraints are defined on succeeding tasks. We propose a probabilistic method which assists the participant when deciding which work item to handle next, with the intention to decrease turnaround times and to avoid time-related escalations, by providing information about the delay to expect when postponing tasks.

Keywords: Workflow management systems, time management, time plans, process monitoring and tracking.

1 Introduction

Workflow systems execute workflows and assign tasks or work items to participants according to activities defined in a workflow model, i.e. an activity is assigned if all preceding activities are finished. Participants use a work list to manage the work items assigned to them. Usually, they have to decide which work item to handle next. The decision to work on a particular work item implicitly holds the decision to postpone every other work item in the work list. This may have grave effects on the execution duration of the workflows to which these postponed work items belong. Common policies for this decision problem, like first-in first-out (FIFO) or earliest-deadline-first, may be suboptimal because they do not take into account that a) the postponement of a task may not immediately delay a workflow due to eventually existing buffer times, and b) even a slight postponement may lead to a disproportionately high delay due to fixed-date constraints on succeeding tasks (e.g. 'a task must be finished until the 1st of a month').

Consider the workflow *JobPosting* to announce open positions of a big company in the local newspaper, as visualized in Fig. 1: A department initializes the process by generating a claim. At first the claim will be forwarded to the personnel division where it is *prepared* for further processing. Then the claim is *validated* by the personal manager. After this, a job offer for the open position



Worklist for Mr. Smith			Sunday, May 8th 2005
ID	**Task**	**Process**	**Received**
34848	prepare	JobPosting1	May 6th 2005
34856	create	JobPosting2	May 7th 2005

Fig. 1. Scenario1: job posting workflow

is *created*. Then the offer is *mailed* to the newspaper. Finally the job offer is *finished* (filing, notification of departments, etc.). The expected duration in days is displayed on top of each task. Additionally, as the newspapers special "Job & Career" edition appears only once a month (on each 15th) a corresponding fixed-date constraint (fdc) has been defined on the last activity *mail*, demanding that it must be finished on the 14th of a month. Currently, on Sunday May 8th 2005, Mr. Smith, employed in the companies personnel division, has two work items from two different JobPosting-processes in his work-list. For sake of simplicity, we assume that every day is a working day, Saturdays and Sundays included.

Scenario 1.a) According to the FIFO-policy, suggested by the work-list client, he starts to execute the first work item, the task *prepare* of process *JobPosting1*, which will presumably take 6 days. Although the execution of *prepare* starts immediately, the job offer will, according to the expected task durations, not appear in the May-issue, as the *mail*-task will presumably be finished on May 21st and the process will be finished at May 24th. Unfortunately, the decision to execute *prepare* first implicitly postpones the execution of his second work item, the task *create* of process *JobPosting2*, for 6 days, to May 14th. *create* can be finished on May 16th and the next step *mail* on May 17th. Thus the job offers of the second process will, presumably, also not appear in this month special issue.

Scenario 1.b) If Mr. Smith had chosen to execute the second work-item first, the *mail*-task of process *JobPosting2* could have been finished on time, until May 11th. Thus the FIFO-policy unnecessarily delays the second process for 30 days. Although this decision postpones the first work-item *prepare*, the *JobPosting1* process will not be delayed, as it will also end at May 24th (which is the same end date as in Scenario1). Due to the fixed-date constraint defined on *mail*, enough buffer time exists to compensate the postponement! The scenarios demonstrate that an intelligent and predictive selection of work-items can help to decrease the turn-around times of processes.

In this paper we introduce a method which exploits knowledge about the workflow structure and time properties, to assist workflow participants when deciding which work item to handle next, in order to decrease turnaround times and to avoid time-related escalations, by providing information about the delay to expect when selecting or postponing tasks.

The paper is structured as follows: In Section 2 we present related work. Section 3 introduces the probabilistic timed workflow model. In Section 4 we examine the relationship between delay and postponement and introduce delay tables. In Section 5 we add the notion of probabilistic delay tables in order to deal with conditional workflow execution, including a presentation of structural workflow transformations [9]. Section 6 explains how to interpret probabilistic delay tables by using delay time histograms. Finally Section 7 concludes the paper with a brief outlook on our future research topics.

2 Related Work

When dealing with time management it seems to be straightforward to apply existing and extensively examined techniques from related areas (like project management) on workflows. Unfortunately some complex issues originating from workflow modelling and instantiation complicate this intention [2,13]. For instance workflow systems typically do not compute schedules due to the partial knowledge they have about the execution of their processes and the impossibility to know the actual flow at decision points at process instantiation time.

Several publications introduced methods which cope with these workflow specific problems, e.g. [4,5,7,11,12]. They propose the calculation of time plans which define execution intervals for each activity in a workflow, such that deadlines will not be violated. The concepts presented are inspired by PERT and CPM charts which are commonly used in project management. But all of them suffer from the uncertainty about time and branching information which arise when tasks are executed conditionally. Some newer publications already utilize stochastic information which are used for performance analysis [1,10] or scheduling issues [3]. We dealt with this problem by introducing probabilistic time management as described in [6].

How to handle different types of time constraints is explained in [7,11]. Although most of the above mentioned publications deal with similar time related problems, as far as we know, currently no publication exists which examines the relation between postponement and delay. The approach presented in this paper follows the basic motivation of [8]: to assist the workflow participant with supportive work-list tools.

3 Probabilistic Timed Workflow Model

We define a full-blocked workflow model that we use in the rest of this paper. Workflows are represented as directed acyclic graphs (see Figures 1, 3 and 5). Edges determine the execution sequence of nodes, thus a successor can start if its predecessor(s) are finished. A node is identified by a unique name and can be of type *activity* (represented by boxes), which corresponds to individual tasks of a business process, or a control node of type *start*, *end*, *and-split|join* or *or-split|join*. Conditional branches are augmented with statistically weighted branching probabilities (defined by estimations or empirically generated values from the workflow log). Routes after an *and-split* will be processed concurrently.

They will be synchronized at the according *and-join*, which does not proceed until all predecessor nodes are finished. After an *or-split* only one route, depending on a run-time evaluated condition, will be selected. The corresponding *or-join* proceeds if one predecessor node is finished (the one on the route selected at the or-split). A workflow model is *full-blocked* if the graph is restricted to proper nesting of splits and joins. Therefore and-structures and or-structures may be nested, but must not overlap (according to the conformance-class declaration in [14]).

N designates the node named N and $N.type$ yields the type of the node. $M \rightarrow N$ designates an edge between two nodes M and N, and $p_{M \rightarrow N}$ yields the branching probability for edges going out from or-splits. $N.Pred$ defines the set of adjacent predecessor nodes and $N.Succ$ defines the set of adjacent successor nodes.

Additionally, *time properties* have to be defined during the modelling process of the workflow. Each node N is augmented with the expected *duration* $N.d$ in an arbitrary time unit like hours or days. Control nodes like *start* or *end* usually have a duration of 0. The workflows overall execution duration is limited by a workflow *deadline* δ, which is defined as a time value relative to the start of the workflow (note that the definition of δ is compulsory!).

Additionally the execution of a node can be constrained to certain dates by a *fixed-date constraint*. Although it is basically possible to assign a fixed-date constraint to either the start or the end of a node, we restrict our explanations to the end of a node (to avoid the need of describing analogous concepts). A fixed-date constraint $fdc(N, F)$ expresses that the end of node N can only occur on certain dates specified by the *fixed-date object* F, where $F.valid(Date)$ returns true if the arbitrary date is valid for F. $F.next(Date)$ and $F.prev(Date)$ return the next and previous valid dates after $Date$. Assume that a fixed-date object f may for example be defined as a list of valid dates f = *(12th of March, 17th of June, 25th of September)* or as an expression like f = *every 3rd sunday starting with 6th of September*.

4 Calculation of Delay

4.1 Earliest Possible End

Based on the workflow structure and time properties, it is possible to calculate the *earliest possible end (EPE)* for each node, relative to the start time of the workflow. It depicts the earliest possible point in time an activity may end, defined by the sum of durations of preceding nodes. The calculation of EPEs starts from the current node Cur with the current time now. Additionally, as and-joins demand that every preceding node must be finished before the execution can be continued, the EPE of an and-join is determined by the maximum EPE of all adjacent preceding nodes. For this we define the operation $maxPredEPE(N)$ for a node N, which yields the maximum EPE of all predecessors if N is an and-join or the EPE of the single predecessor otherwise. The EPE of every node, which

Algorithm 1 Calculation of Earliest Possible End Times

```
1: $Cur.epe := now + Cur.d$
2: for every $N$ succeeding $Cur$ in forward top. order do
3:     $N.epe := maxPredEPE(N) + N.d$
4:     if if $fdc(N, f) \in FDC$ then
5:         $N.epe := f.next(N.epe)$
6:     end if
7: end for
```

is a transitive successor of the current node *Cur*, is calculated as described in algorithm 1. How to deal with EPEs for or-joins is explained in section 5.

Consider the workflow from scenario 1 with $Cur = prepare$ and $now = may8$ (depicting the current time: 8th of May 2005) with the EPEs: $prepare.epe = may14$, $validate.epe = may19$, $create.epe = may21$ and $mail.epe = may22$. According to the fixed-date constraint defined on *mail*, the EPE must be shifted to the next valid date adhering to $fdc(mail,$14th of a month), yielding $mail.epe = jun14$. Then we calculate $finish.epe = jun17$. Finally, as control nodes like *End* have a duration of 0, the earliest possible end of the workflow is $End.epe = jun17$.

There are two special cases to consider: a) As f may define a finite set of fixed dates, it may yield no valid next date at *f.next(N.epe)*. Then the user or an administrator must be informed that a future deadline or fixed-date violation will (presumably) occur, and a further delay due to a postponement of *Cur* has to be avoided. b) Due to and-structures many activities may currently be executed in parallel. Thus more than one current activity *Cur* may exist. In this case all current activities must be initialized as in line (1). The algorithm still works, as due to the full-blocked structure of the workflow graph all parallel routes will be joined at a succeeding and-join. Additionally, some participants may already have started to execute their current tasks. In this case the duration of the node must be adapted by applying *Cur.d = Cur.d - (now - Cur.start)*, where *start* denotes the actual start time of the nodes execution.

4.2 Delay Tables

A *postponement* designates the shift of the execution of an activity, which is ready to be executed by a participant, to a later point in time. A *delay* is generated if due to a postponement the earliest possible end of the workflow (= *End.epe*) would be increased.

In our running example with $now = may8$ and $Cur = prepare$ we additionally define an overall workflow deadline of $\delta = 90$, therefore the workflow (or the node *End*) must be finished until $now + \delta = aug6$. The examination of dependencies between delay and postponement must start at the last node. One can see, that the workflow can not end before $End.epe = jun17$ and must not end after $now + \delta = aug6$. Thus, the following can be stated (for *End*): if it is not postponed the workflow will not be delayed and end at $jun17$, if it is postponed by 1 day the workflow will be delayed by 1 day and end at $jun18$, and so on. Finally we can state, that if it is postponed by 50 days the workflows will be delayed by 50 days and end at *aug*6. A further postponement will lead to a violation of the workflow deadline $now + \delta = aug6$.

Based on this knowledge it is possible to define a relation between a *delay deadline (d)* and a *delay time (t)*, where t is the delay of the workflow to expect, if the end of the node is postponed to d. This relation is captured in the *delay table (DT)*. To query delay tables we introduce the operation *delay time selection*, which yields the delay time for a given delay deadline.

Definition 1 (Delay table, DT). *A* delay table *N.dt for a node N is a set of tuples (d, t) describing the delay time t to expect if N ends at the delay deadline d. Furthermore a delay table is called* valid *if each t and each d is unique in the set. The operation $y = selectT(dt, x)$ applied on a valid delay table selects the delay time y for a delay deadline x, such that:*

$$y = \begin{cases} z \text{ if } \forall(d,t) \in dt, \text{ where } x \geq d\text{: } z \leq t \\ \infty \text{ if } \forall(d,t) \in dt\text{: } x > d \end{cases}$$

The valid DT of *End* is *End.dt*=$\{$*(jun17,0),(jun18,1),(jun19,2), +++,(aug6, 50)*$\}$. The *+++* between two tuples (d_1, t_1),*+++*,(d_n, t_n) symbolize that the set holds additional tuples (d_i, t_i), such that $d_i = d_{i-1} + 1$ and $t_i = t_{i-1} + 1$ for $1 > i > n$. The operation *selectT* yields the delay time for a given delay deadline. If no tuple with the requested delay deadline exists, the delay time of the tuple with the next greater delay deadline is selected: e.g. *selectT(End.dt, jun21) = 4* states that the workflow will be delayed by 4 days if the node *End* ends at *jun*21. The explicit definition of a result of ∞ is necessary as a point in time may be selected which is greater than the maximum delay time in the DT. If postponed to that point in time a workflow deadline violation would occur: e.g. *selectT(End.dt, sep21)* $= \infty$ states that if it ends at *sep*21 the workflow deadline will be violated.

As the relation between d and t in *End.dt* is linear, we call it a *linear DT*. To calculate the always linear DT of an end node we use the following operation: *End.dt = linearDT(End.epe,now + δ)*.

Definition 2 (Calculate linear DT). *A linear DT dt, delimited by a lower bound lb and an upper bound ub, is generated as follows: $dt = linearDT(lb, ub) = \{(d,t) \mid \forall d : lb \leq d \leq ub, t = d - lb\}$, where $|dt| = ub - lb + 1$.*

For nodes, which are to be executed in a sequence, the DT of a predecessor node is calculated by subtracting the duration of the successor from the DT of the successor:

Definition 3 (Subtract scalar from DT). *The scalar k is subtracted from a DT dt_1 resulting in a DT dt_2 as follows: $dt_2 = dt_1 - k = \{(d-k, t) \mid (d,t) \in dt_1\}$.*

Therefore *finish.dt = End.dt - 0* and *mail.dt = finish.dt - finish.d* (see also Fig. 2). As a fixed-date constraint *fdc(mail,f)* exists, the end of *mail* may only occur on valid dates, defined by the fixed-date object *f=14th of a month*. Thus the delay deadlines must be adjusted using the *previous* function of the fixed-date object.

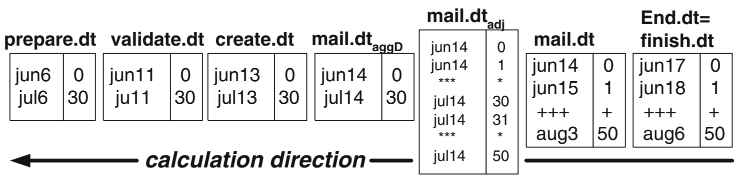


Fig. 2. Delay tables for Scenario1

Definition 4 (Adjust delay table to fixed-date object). *A delay table dt is adjusted to a fixed date object f, resulting in a delay table dt′, as follows:* $dt' = adjust(dt, f) = \{(f.prev(d), t) \mid (d, t) \in dt\}$.

The operation yields the $mail.dt_{adj}$ = *(adjust(mail.dt,f))=*{*(jun14,0),(jun14,1), *** ,(july14,30),(july14,31), *** , (july14,50)*}. The ***** between two tuples (d_1, t_1),*****,(d_n, t_n) symbolize that the set holds additional tuples (d_i, t_i), such that $d_i = d_1$ and $t_i = t_{i-1} + 1$ for $1 > i > n$. As one can see this operation results in an invalid DT with multiple identical delay deadlines (but different delay times). In order to receive a valid DT we have to remove all tuples with identical delay deadlines, but the one with smallest delay time. To achieve this the operation *delay deadline aggregation* must be applied on $mail.dt_{adj}$.

Definition 5 (Delay deadline aggregation). *The operation* $aggD(dt_1)$ *yields an aggregated delay table* dt_2*, such that* $dt_2 = aggD(dt_1) = \{(x, minT(dt_1, x)) \mid (x, t) \in dt_1\}$*; where* $y = minT(dt_1, x)$ *selects the minimum delay time y for a delay deadline x of the delay table* dt_1*, such that* $\forall(x, t_1) \in dt_1$*:* $y \leq t_1$ *must hold.*

Now we apply *aggD* on the intermediary DT $mail.dt_{adj}$: $mail.dt_{aggD}$ = $aggD(mail.dt_{adj})$ = {*(jun14,0),(jul14,30)*}, which is interpreted as: if *mail* ends at a $date \leq jun14$, the delay will be 0; if it ends a date $jun14 < date \leq jul14$ the delay will be 30. And the next possible end, according to the fixed-date constraint, is $aug14$. But if *mail* would end at $aug14$, the workflow deadline would be violated. Thus the last allowed delay deadline (or end time) is *jul14*. The operation which combines *aggD* and *adjust* is defined as follows:

Definition 6 (Apply fixed-date object on delay table). *The fixed-date object f is applied on a delay table* dt_1 *yielding a delay table* dt_2*, such that* $dt_2 = applyFDC(dt_1, f) = aggD(adjust(dt_1, f))$.

Afterwards we proceed by subsequently subtracting the durations *create.d=2* and *validate.d=5*, finally yielding *prepare.dt=*{*(jun6,0),(jul6,30)*}. The DT of the current activity *prepare* can be used to answer participant-questions like "By how many days will the workflow be delayed if the end of *prepare* is postponed to *jun25*?". The answer is *selectT(prepare.dt, jun25) = 30*: "The delay will be presumably be 30 days!". And for *selectT(prepare.dt, aug8)* $= \infty$ the answer would be: "A deadline violation will occur!". The DT may also be used to inform the participant (Mr. Smith) that he may postpone the end of *prepare* to $jun6$ without delaying the workflow, which is defined by the tuple with a delay time of 0.

4.3 Calculation of DTs for And-Splits

We adapt the example, as visualized in Fig. 3: The company decides that the management's staff unit must be informed about every claim. Additionally a fixed-date constraint is attached to the new activity *check* (which takes 1 day), as the staff unit team only meets every 3d monday (starting with may 9th) to discuss and check new claims. Assume that the semantics of *finish* is now: finally the job offer and the claim are finished together (filing, notification of departments, etc.). Again *Cur=prepare, now=may8* and *δ=90.* At first the EPEs must be calculated according to the algorithm presented in Section 4.1: *Start.epe=may8, prepare.epe=may14, AS.epe=may14, validate.epe=may19, create.epe=may21, mail.epe=jun14, check=may30, AJ.epe=jun14, finish.epe=jun17* and *End.epe=jun17.* The EPE of the new activity *check* has been adjusted to the next valid date $may30$ (= $may9$ + 3weeks) according to its fixed-date constraint. And the EPE of the and-split AJ is equal to the maximum EPE of its predecessors (*mail* and *check*), which is *mail.epe.*

The calculation of DTs starts with the initialization of the last node *End.dt = linearDT(End.epe,now + δ)*, followed by the subsequent calculation of $finish.dt$ and *AJ.dt* (see also Fig.4). The DTs of nodes which are predecessors of an and-split are calculated like nodes in a sequence: *mail.dt = AJ.dt - AJ.d* and *check.dt = AJ.dt - AJ.d.* Since *AJ.d=0*, their DTs are equal to *AJ.dt.* For the upper parallel route we refer to the calculations of Scenario1, as the DTs are equal. For the single node *check* of the second route we have to apply its fixed-date constraint *applyFDC(check.pdt,every 3d monday starting with may9))*, resulting in $check.dt_{fdc}$ (to save some space we did not show the intermediate results $check.dt_{aggD}$ and $check.dt_{adjust}$).

Before combining DTs of and-split successors, their durations must be subtracted (resulting in an intermediate dt'): *validate.dt'={(jun6,0),(jul6,30)}* and *check.dt' = {(may29,0),(jun19,6),(jul10,27)}*. To combine two DTs the operation *delay table conjunction*, which is a combination of the operations *merge* and $aggT$, has to be applied.

Definition 7 (Merging two delay tables). *The operation* merge(dt_1,dt_2) *applied on two delay tables* dt_1 *and* dt_2*, results in a delay table* dt *as follows:* dt = merge(dt_1, dt_2) = {(d,t) | d ∈ D, selectT(dt_1,d) $\neq \infty$, selectT(dt_2,d) $\neq \infty$,

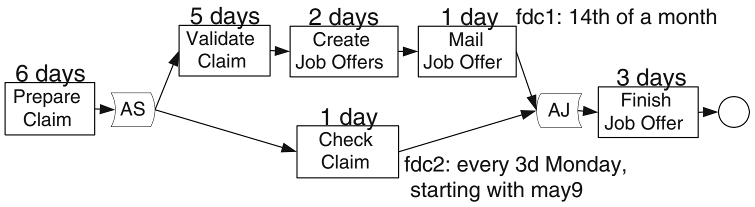
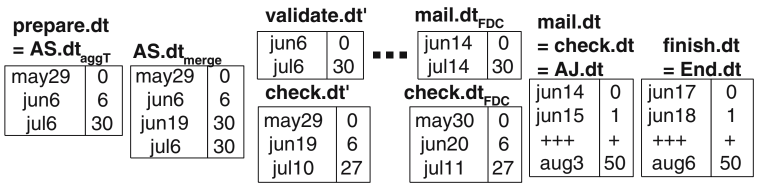


Fig. 3. Scenario2: and-structure



prepare.dt = AS.dt$_{aggT}$

may29	0
jun6	6
jul6	30

AS.dt$_{merge}$

may29	0
jun6	6
jun19	30
jul6	30

validate.dt'

jun6	0
jul6	30

check.dt'

may29	0
jun19	6
jul10	27

■ ■ ■

mail.dt$_{FDC}$

jun14	0
jul14	30

check.dt$_{FDC}$

may30	0
jun20	6
jul11	27

mail.dt = check.dt = AJ.dt

jun14	0
jun15	1
+++	+
aug3	50

finish.dt = End.dt

jun17	0
jun18	1
+++	+
aug6	50

Fig. 4. Delay tables for Scenario2

t = max(selectT(dt$_1$,d),selectT(dt$_2$,d))}, *where* D = {d$_1$ | (d$_1$,t$_1$) $\in$ dt$_1$ } $\cap$ {d$_2$ | (d$_2$,t$_2$) $\in$ dt$_2$}.

The merge-operation *AS.dt = merge(validate.dt',check.dt'* selects the greatest delay time (as all routes are processed in parallel) for every possible delay deadline, which is accomplished as follows: first a set D which holds all delay deadlines of both sets is generated, which is for our example *D={may29,jun6,jun19,jul6, jul10}*. Then for every delay deadline in D the *selectT*-operation is applied on both DTs and the maximum of the results is selected, which is 0 for $may29$, 6 for $jun6$, 30 for $jun19$ and 30 for $jul6$: *AS.dt$_{merge}$ = {(may29,0),(jun6,6),(jun19, 30), (jul6,30)}*. The delay deadline $jul10$ has been filtered out by the condition *maxT(dt$_1$,d)* $\neq \infty$ as *maxT(validate.dt,jul10)* $= \infty$. The intermediate DT is invalid as it holds two tuples with equal delay times. To receive a valid DT it is necessary to remove the tuple with the earlier delay deadline by applying the delay time aggregation.

Definition 8 (Delay time aggregation). *The operation $aggT(dt_1)$ yields an aggregated delay table dt_2, such that $dt_2 = aggT(dt_1) = \{(maxD(dt_1, x), t) \mid (x, t) \in dt_1\}$; where $y = maxD(dt_1, x)$ selects the maximum delay deadline y for a delay time x of the delay table dt_1, such that $\forall (d_1, x) \in dt_1 : y \geq d_1$.*

According to this *AS.dt$_{aggT}$* $=$ $aggT$(*AJ.dt$_{merge}$*) $=$ *{(may29,0),(jun9,6), (jul9,30)}*. And finally, due to *AS.d = 0*, the DT of the current activity *prepare* is *prepare.dt = AS.dt$_{aggT}$ – 0*. Based on the operations $merge$ and $aggT$ we define the *delay table conjunction*.

Definition 9 (Delay table conjunction). *The delay table conjunction $dt = dt_1 \wedge dt_2$ applied on two delay tables d_1 and d_2 yields a delay table dt, such that $dt = aggT(merge(dt_1, dt_2))$.*

Proposition: The conjunction is commutative and associative as $dt_A \wedge dt_B = dt_B \wedge dt_A$ and $(dt_A \wedge (dt_B \wedge dt_C)) = ((dt_A \wedge dt_B) \wedge dt_C)$. Thus it can be extended to any number of DTs: $dt_1 \wedge ... \wedge dt_n = \bigwedge_{i=1}^{n} dt_i$.

5 Calculation of Probabilistic Delay

One fundamental core problem has not been addressed so far: in workflows with or-splits many different execution routes (also called *instance types*) are possible.

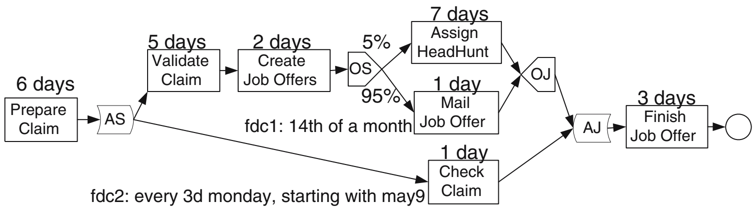


Fig. 5. Scenario3: or-structure

Consider the workflow in Fig. 5: The company decides to assign a head hunter for open high-management positions, instead of advertising in the newspaper. Selection of and negotiation with a head hunter takes about 7 days. According to past experiences about 5% of all job offers address high-management positions. No fixed-date constraint is assigned to this task. In this graph two instance types can be identified, as two different execution sequences of activities and nodes are possible: the first one via activity *assign* and the second one via activity *mail*. Note that different instance types are only produced if the graph contains or-structures (after and-splits all nodes will be executed unconditionally).

5.1 Unfolding the Graph

When calculating the EPEs a problem arises at or-joins: *OJ.epe* can not be determined unambiguously, as the EPE depends on the path executed prior to *OJ*. Which one this will be, is unknown at the current node *Cur*. The only thing we can tell is, that it is likely that the route via *assign* will be executed with a probability of 5% and the route via *mail* with a probability of 95%. As the algorithm to calculate EPEs demands exactly one EPE per node, we developed the following solution.

At first we *unfold* the workflow, in order to split up different instance types. [9] developed a *backward unfolding procedure*, which successively applies basic transformation operations on an originally block-structured graph. A transformation operation results in a new workflow graph, which is semantically equal to the old one, based on the notion of equivalence of tasks and identical execution order. The result of the backward unfolding procedure is a graph where every or-join has been moved to the end.

Fig. 6 shows the unfolded version of the graph from Fig. 5. Three basic transformation operations have been applied to move the or-join to the end of the workflow: first the or-join has been moved over the and-join, which resulted in a duplication of nodes and edges succeeding the or-join. The second and third operation moved the node over the activity *finish* and end node towards the end of the workflow. It is important to notice that the possible order of node-execution is equal to the one in the original graph. A backward unfolded graph will have as many end-nodes as possible routes through the graph exist, which is one for each instance type. Furthermore, as synchronizing or-joins have been moved to the end, it enables us to calculate exactly one earliest possible end

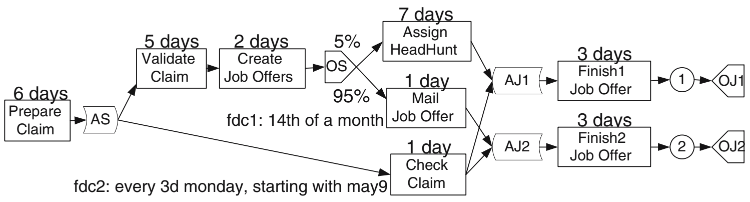


Fig. 6. Unfolded workflow graph

Algorithm 2 Calculation of execution probabilities

```
1: $Cur.x = 1$
2: for every successor $N$ of $Cur$ in forward top. order do
3:    if $P \in N.Pred$ and $P.t$ = or-split then
4:       $N.x = P.x * p_{P \to N}$
5:    else if $N.type$ = and-join then
6:       $N.x$ = minimum of all $P.x$, $P \in Pred_N$
7:    else
8:       $N.x = P.x, P \in Pred_N$
9:    end if
10: end for
```

time for each node, by applying the algorithm of section 4.1. For further details on graph transformations see [9].

5.2 Calculation of Execution Probabilities

As mentioned above, the calculation of DTs starts at the end node of a graph, but in the unfolded graph multiple end nodes have to be considered. Therefore the initial DTs of end-nodes must be weighted according to their execution probability, in relation to the current activity *Cur*. This can be accomplished by calculating the execution probability dependent on predecessor execution probabilities and the type of nodes. For this we have to traverse the graph in a forward topological order starting with the current node (see algorithm 2). As the current node will be executed in any case, its execution probability is initialized with $Cur.x = 1$. The calculation finally yields $End1.x = 0.05$ and $End2.x = 0.95$, which means, that starting from the current node *Cur*, the end-node *End*1 will be executed with a probability of 5% and the end-node *End*2 with a probability of 95%. The sum of probabilities of all end nodes must be 1.

5.3 Probabilistic Delay Tables

With the unfolded graph and execution probabilities for end-nodes it is possible to calculate a set of *probabilistic delay tables (PDTs)* for the current node, which can be queried in several ways. Each node *N* of an unfolded graph can hold multiple weighted DTs, one for each end-node that is reachable from *N* (or each possible instance type starting from *N*), which are stored in the set *N.PDT*.

Definition 10 (Set of Probabilistic Delay Tables, PDT). *The set of* probabilistic delay tables *N.PDT of a node N is a set of tuples* (e, p, dt)*, where dt is a delay table, e is the end-node from which dt originated and p is the execution-probability of the end-node.*

For the PDT-calculation algorithm the operations defined on DTs must be adapted to set-based PDT-operations. Details will be explained in the subsequent section, along with the calculation of PTDs for our running example:

Definition 11. Operations on PDTs

- *Subtract scalar: PDT - k* $= \{$*(e,p,dt-k)* $|$ *(e,p,dt)* $\in$ *PDT*$\}$
- *Apply fixed-date constraint: applyFDC(PDT,f)* $= \{$*(e,p,applyFDC(dt))* $|$ *(e,p,dt)* $\in$ *PDT*$\}$
- *Conjunction:* $PDT_1 \wedge PDT_2 = \{(e,p,dt_1 \wedge dt_2) \mid (e,p,dt_1) \in PDT_1, (e,p,dt_2) \in PDT_2\}$.

Proposition: The conjunction is commutative and associative as $PDT_A \wedge PDT_B = PDT_B \wedge PDT_A$ and $(PDT_A \wedge (PDT_B \wedge PDT_C)) = ((PDT_A \wedge PDT_B) \wedge PDT_C)$. Thus it can be extended to any number of PDTs: $PDT_1 \wedge ... \wedge PDT_n = \bigwedge_{i=1}^{n} PDT_i$.

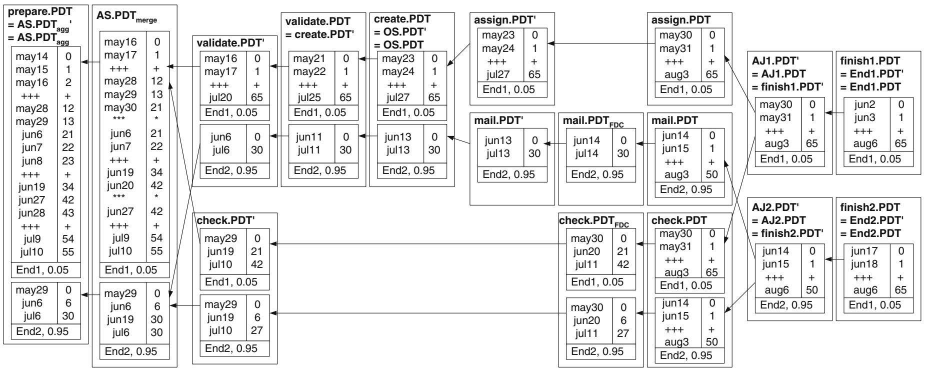
5.4 Calculation of PDTs

Before calculating the PDTs, the graph must be unfolded and or-joins must be deleted, followed by the forward calculation of execution probabilities and EPEs, resulting in: *End1.epe=jun2, End2.epe=jun17, End1.x=0.05* and *End2.x=0.95.* The calculation of *PDTs* is again performed in a backward topological order, where the PDT of each node is determined according to its node type (see algorithm 3 and Fig. 7). Note, that we use the intermediary $N.PDT'$ for each node N, which contains the PDT after fixed-date adjustment and subtraction of node duration.

The algorithm starts with the initialization of all end-nodes End_1 and End_2 as defined in line 2 of the algorithm: $End_i.PDT = \{ (End_i, p_i, linearDT(End_i.epe, now + \delta))\}$. The PDT of an end-node contains exactly one tuple with a reference to the originating end-node End_i (in this case the node itself), the execution probability of this end-node $End_i.x$, and the initial linear delay table, calculated as $linearDT(End_i.epe, now + \delta)$. Since no fixed-date constraint on End_i exists (line 9), the intermediary PDT is calculated as $End_i.PDT' = End_i.PDT - End_i.d$ (line 12). Due to $End_i.d = 0$, this results in $End_i.PDT' = End_i.PDT$.

In the next iteration the PDT of the preceding node is determined. Since $finish_i$ is no end-node and no and-split, the PDT is calculated as union of all successor-*PDT'* (line 7). As $finish_i$ has only one successor, the PDTs are equal: $finish_i.PDT = End_i.PDT'$. Then the duration of $finish_i.d = 3$ is subtracted, yielding $finish_i.PDT'$ (line 12).

The calculation of PDTs for nodes between $finish_i$ and *OS* are calculated analogously. As for each node only one possible route to an end-node has to be considered the *union*-operation always results in a set with one tuple. At *mail*



prepare.PDT = AS.PDT$_{agg}$' = AS.PDT$_{agg}$
may14 0, may15 1, may16 2, +++ +, may28 12, may29 13, jun6 21, jun7 22, jun8 23, +++ +, jun19 34, jun27 42, jun28 43, +++ +, jul9 54, jul10 55
End1, 0.05
may29 0, jun6 6, jul6 30
End2, 0.95
AS.PDT$_{merge}$
may16 0, may17 1, +++ +, may28 12, may29 13, may30 21, *** *, jun6 21, jun7 22, +++ +, jun19 34, jun20 42, *** *, jun27 42, +++ +, jul9 54, jul10 55
End1, 0.05
may29 0, jun6 6, jun19 30, jul6 30
End2, 0.95
validate.PDT'
may16 0, may17 1, +++ +, jul20 65
End1, 0.05
jun6 0, jul6 30
End2, 0.95
check.PDT'
may29 0, jun19 21, jul10 42
End1, 0.05
may29 0, jun19 6, jul10 27
End2, 0.95
validate.PDT = create.PDT'
may21 0, may22 1, +++ +, jul25 65
End1, 0.05
jun11 0, jul11 30
End2, 0.95
create.PDT = OS.PDT' = OS.PDT
may23 0, may24 1, +++ +, jul27 65
End1, 0.05
jun13 0, jul13 30
End2, 0.95
assign.PDT'
may23 0, may24 1, +++ +, jul27 65
End1, 0.05
mail.PDT'
jun13 0, jul13 30
End2, 0.95
mail.PDT$_{FDC}$
jun14 0, jul14 30
End2, 0.95
check.PDT$_{FDC}$
may30 0, jun20 21, jul11 42
End1, 0.05
may30 0, jun20 6, jul11 27
End2, 0.95
assign.PDT
may30 0, may31 1, +++ +, aug3 65
End1, 0.05
mail.PDT
jun14 0, jun15 1, +++ +, aug3 50
End2, 0.95
check.PDT
may30 0, may31 1, +++ +, aug3 65
End1, 0.05
jun14 0, jun15 1, +++ +, aug3 50
End2, 0.95
AJ1.PDT' = AJ1.PDT = finish1.PDT'
may30 0, may31 1, +++ +, aug3 65
End1, 0.05
finish1.PDT = End1.PDT' = End1.PDT
jun2 0, jun3 1, +++ +, aug6 65
End1, 0.05
AJ2.PDT' = AJ2.PDT = finish2.PDT'
jun14 0, jun15 1, +++ +, aug6 50
End2, 0.95
finish2.PDT = End2.PDT' = End2.PDT
jun17 0, jun18 1, +++ +, aug6 65
End1, 0.05

Fig. 7. Calculation of probabilistic delay tables for Scenario3

Algorithm 3 PDT calculation

```
 1: for all nodes $N$ in a backward topological order do
 2:    if $N.type = end$ then
 3:       $N.PDT=\{(N,\ N.x,\ linearDT(N.epe,\ now + \delta))\}$
 4:    else if $N.type = and\text{-}split$ then
 5:       $N.PDT = \bigwedge S.PDT', \forall S \in N.Succ$
 6:    else
 7:       $N.PDT = \bigcup S.PDT', \forall S \in N.Succ$
 8:    end if
 9:    if $\exists fdc(N, f) \in FDC$ then
10:       $N.PDT' = applyFDC(N.PDT, f) - N.d$
11:    else
12:       $N.PDT' = N.PDT - N.d$
13:    end if
14:    if $N = Cur$ then
15:       return
16:    end if
17: end for
```

the fixed-date constraint has to be applied before subtracting the duration (line 10), yielding the same DT as in scenario 2.

So far, each determined PDT holds exactly one tuple: e.g. *assign.PDT'* = $\{$*(e,p,dt)*$\}$ states that the delay information stored in *dt* originates from the end-node $e{=}End_1$, which has an execution probability of *p=0.05.* The or-split *OS* has two successors, therefore *check.PDT* = *assign.PDT'* $\cup$ *mail.PDT'* (line 7). This results in a set with two tuples holding information about two different delay tables which originate from two different end-nodes (with different execution probabilities). Thus we can state, that the number of tuples in the PDT of a node is equal to the number of instance types containing this node. The same can analogously be stated for the node *check*, as it also has two successors leading to two different end-nodes. We proceed with the calculation of *validate.PDT'* (subtract duration) and *check.PDT'* (apply fdc and subtract duration)

At the and-split a PDT-conjunction has to be applied on successor-PDTs: *validate.PDT'* $\wedge$ *check.PDT'*. The PDT-conjunction basically applies a regular DT-conjunction on all DTs with identical end-node references (they belong to the same instance type). Therefore it consists of two steps: 1) first it merges all DTs, which originate from the same end-node, into one DT_{merge} (see PDT_{merge} in Fig. 7). 2) Subsequently, to remove duplicate delay times, it aggregates each intermediate DT_{merge} by applying an aggregation on each DT. Note that PDT_{merge} and PDT_{agg} are displayed for illustration issues only, as according to its definition the PDT-conjunction does not produce intermediate results. Finally, the PDT for the current activity *prepare* is calculated.

6 Interpretation and Application

PDTs are used to generate information about the delay to expect for a workflows if a participant postpones the end of his current task to a given date. The answer to the question "What is the expected delay, if the end of *prepare* is postponed to $june1$?" can no longer yield an ambiguous result, as the PDT of prepare holds two DTs with different weights. Therefore we introduce *(cumulated) delay time histograms.*

Definition 12 (Delay Time Histograms). *The cumulated delay time histogram* N.CDTH$_x$ *for a node* N *and a date* x*, based on the delay time histogram* N.DTH$_x$ = $\{(\Sigma$p,selectT(dt,x)) | (e,p,dt) $\in$ N.PDT$\}$*, is a set of tuples* (c$_i$,t$_i$) *such that* $c_i = \sum_{t_j \le t_i} p_j$, where (p$_i$,t$_i$) $\in$ I.

The DTH contains one tuple with delay time and probability for each DT in the PDT, selected for a given delay deadline *x*: *prepare.DTH*$_{jun1}$ = $\{$*(0.05,21), (0.95,6)*$\}$. The expression Σp aggregates tuples with equal delay times by adding up their probabilities. For the CDTH the probability values of the DTH are cumulated in increasing order of delay times: *prepare.CDTH*$_{jun1}$ = $\{$*(0.95,6), (1.0,21)*$\}$. It is interpreted as "With a probability of 95% the maximum delay will be 6 days, and with a probability of 100% the maximum delay will be 21 days!". For huge PDTs containing hundreds of weighted DTs, answers like the ones above will be of no use for the participant. Therefore we introduce the selection for a defined certainty (minimum probability).

Definition 13 (Selection of probabilistic delay time). t = selectPT(N,x,c) *selects the maximum delay time* t *for a delay deadline* x *and a given minimum probability* c*, such that* $\forall$(c$_i$,t$_i$) $\in$ N.CDTH$_x$*, where* $c_i \le c$*:* $t_i \ge t$.

Thus *selectPT(prepare,jun1,0.9) = 6*: "The maximum delay will be 6 days (with a 90% certainty)!". As the workflow participant should not be confused with probability values, the minimum certainty should be configured as a fixed work-list parameter. The selectPT-operation can now be used to generate a presorted work list, which proposes an execution order of work items where the overall sum of delay times is significantly reduced, compared to for instance a FIFO-strategy. The examination of these algorithms is subject of ongoing work. Nevertheless a necessary prerequisite is, that the workflow system features a time management component, which performs the above explained calculation algorithms based on empirical data from the workflow log. Additionally the work-list must be enabled to invoke this algorithms on demand and to represent the results accordingly.

7 Conclusions and Future Work

We proposed a method which aims at assisting workflow-participants in their decisions to select work-items to be executed next. This method is based on delay tables, containing probabilistic information about the delay to expect when the execution of a task is postponed to a certain date. They are calculated by utilizing the structure of the workflow, augmented with empirical information about expected execution-durations and branching-probabilities. A suggested execution sequence will decrease turnaround times, avoid time-related escalations and subsequently save costs.

Currently, we investigate how to include duration-distributions for activities, as scalar duration values are to imprecise in a realistic administrative workflow scenario. Another research objective is to cope with the complexity explosion due to unfold-operation, by adapting the algorithm in order to work with still folded (and non-full-blocked) graphs. Furthermore we proceed with our investigations on optimization algorithms for presorted work lists. The integration of probabilistic time management into workflow environments is subject of ongoing research.

References

1. W. M. P. van der Aalst and H. A. Reijers. Analysis of discrete-time stochastic petrinets. In *Statistica Neerlandica, Journal of the Netherlands Society for Statistics and Operations Research*, Volume 58 Issue 2, 2003.
2. C. Bussler. Workflow Instance Scheduling with Project Management Tools. In *9th Workshop DEXA'98.* IEEE Computer Society Press, 1998.
3. G. Baggio and J. Wainer and C. A. Ellis. Applying Scheduling Techniques to Minimize the Number of Late Jobs in Workflow Systems. In *Proc. of the 2004 ACM Symposium on Applied Computing (SAC).* ACM Press, 2004.
4. C. Bettini, X. X. Wang, and S. Jajodia. Temporal reasoning in workflow systems. *Distributed and Parallel Databases*, 11(3), May 2002.
5. C. Combi and G. Pozzi. Temporal conceptual modelling of workflows. In *Proc. of the Int. Conf. on Conceptual Modeling (ER 2003).* LNCS 2813. Springer, 2003.
6. J. Eder and H. Pichler. Duration Histograms for Workflow Systems In Proc. of the Conf. on Engineering Information Systems in the Internet Context 2002, Kluwer Academic Publishers, 2002.
7. J. Eder and E. Panagos. Managing Time in Workflow Systems. In *Workflow Handbook 2001.* Future Strategies INC. in association with Workflow Management Coalition, 2000.

8. J. Eder, W. Gruber, M. Ninaus, and H. Pichler Personal Scheduling for Workflow Systems. LNCS 2678, Springer Verlag, 2003.
9. W. Gruber Modeling and Transformation of Workflows with Temporal Constraints. Akademische Verlagsgesellschaft, Berlin, 2004.
10. M. Gillmann, G. Weikum, and W. Wonner. Workflow management with service quality guarantees. In *Proc. of the 2002 ACM SIGMOD Int. Conf. on Management of Data*. ACM Press, 2002.
11. O. Marjanovic, M. Orlowska. On modeling and verification of temporal constraints in production workflows. *Knowledge and Information Syst.*, 1(2), 1999.
12. E. Panagos and M. Rabinovich. Predictive workflow management. In *Proc. of the 3rd Int. Workshop on Next Generation Information Technologies and Systems*, Neve Ilan, Israel, June 1997.
13. S.Sadiq, O. Marjanovic, and M. E. Orlowska. Managing Change and Time in Dynamic Workflow Processes. In *International Journal of Cooperative Information Systems*. Vol. 9, Nos. 1 & 2. March - June 2000.
14. Interface 1: Process Definition Interchange - Process Model. *Workflow Management Coalition Specification*. Document number TC00-1016-P.

Workflow Data Patterns: Identification, Representation and Tool Support*

Nick Russell[1], Arthur H.M. ter Hofstede[1], David Edmond[1], and Wil M.P. van der Aalst[1,2]

[1] School of Information Systems, Queensland University of Technology, GPO Box 2434, Brisbane QLD 4001, Australia
{n.russell, a.terhofstede, d.edmond}@qut.edu.au

[2] Department of Technology Management, Eindhoven University of Technology, P.O.Box 513, NL-5600 MB Eindhoven, The Netherlands
w.m.p.v.d.aalst@tm.tue.nl

Abstract. Workflow systems seek to provide an implementation vehicle for complex, recurring business processes. Notwithstanding this common objective, there are a variety of distinct features offered by commercial workflow management systems. These differences result in significant variations in the ability of distinct tools to represent and implement the plethora of requirements that may arise in contemporary business processes. Many of these requirements recur quite frequently during the requirements analysis activity for workflow systems and abstractions of these requirements serve as a useful means of identifying the key components of workflow languages. In this paper, we describe a series of *workflow data patterns* that aim to capture the various ways in which data is represented and utilised in workflows. By delineating these patterns in a form that is independent of specific workflow technologies and modelling languages, we are able to provide a comprehensive treatment of the workflow data perspective and we subsequently use these patterns as the basis for a detailed comparison of a number of commercially available workflow management systems, workflow standards and web-service composition languages.

1 Introduction

There are a series of concepts that apply to the representation and utilisation of data within workflow systems. These concepts not only define the manner in which data in its various forms can be employed within a business process and the range of informational concepts that a workflow engine is able to capture but also characterise the interaction of data elements with other workflow and environmental constructs.

* This work was partially supported by the Australian Research Council under the Discovery Grant "*Expressiveness Comparison and Interchange Facilitation between Business Process Execution Languages*".

Detailed examination of a number of workflow tools and business process modelling paradigms suggests that the way in which data is structured and utilised within these tools has a number of common characteristics. Indeed these characteristics bear striking similarities to the *Workflow Patterns* [2] and *Design Patterns* [5] initiatives in terms of their generic applicability, except in this case they refer specifically to the data perspective [7] of workflow systems and the manner in which it interrelates with other workflow perspectives.

The use of a patterns-based approach for illustrating data-related concepts in workflow systems offers the potential to describe these constructs in a language-independent way. This ensures that the patterns identified have broad applicability across a wide variety of workflow implementations. It provides the basis for comparison of data-related capabilities between distinct products and offers a means of identifying potential areas of new functionality in workflow systems. In this paper we focus on workflow technology but the results identified apply to any process-aware information system (PAIS).

1.1 Background and Related Work

Interest in workflow systems has grown dramatically over the past decade and this has fuelled the development of a multitude of both commercial workflow engines and research prototypes each with unique features and capabilities. Despite attempts by industry bodies such as the Workflow Management Coalition (`www.wfmc.org`) and the Object Management Group (`www.omg.org`) to provide standards for workflow management systems, there is limited adoption by commercial vendors. Perhaps the most notable shortcoming in this area is the absence of a common formalism for workflow modelling. A number of possible candidates have been considered from the areas of process modelling and general systems design including Petri-Nets [1], Event-Driven Process Chains (EPCs) [11] and UML Activity Diagrams [4] although none of these have achieved broad usage.

Data modelling in the context of workflow systems is an area that has received particularly scant attention. Many of the significant workflow modelling proposals and prototypes take a uniform approach to data representation and utilisation. In ADEPT [9], all data elements are assumed to be represented by global workflow variables. MENTOR [13] proposes that data flow is modelled via Activity Diagrams linked to the control-flow perspective represented in the form of State Charts. WASA [8] assumes that data is characterised in the form of data containers which are passed between workflow activities although the later WASA2 project [12] supports a more varied range of data objects by leveraging underlying CORBA services. INCAs [3] is one of the few initiatives which proposes a broader range of data representations and interaction facilities.

One of the major impediments to a generic modelling technique is the broad range of offerings that fall under the "workflow umbrella" [6] – ranging from unstructured groupware support products through to transaction-oriented production workflows – and the inherent difficulty of establishing a conceptual framework that is both suitable and meaningful across the entire range of offerings. The recent *Workflow Patterns* initiative [2] has taken an empirical approach to identifying the major control-flow constructs that are inherent in workflow systems through a broad survey of process modelling languages and software offerings. The outcomes of this research were a set of twenty patterns characterising commonly utilised process control structures in workflow systems together with validation of their applicability through a detailed survey of thirteen commercial workflow products and two research prototypes. It is interesting to note that this work has directly influenced tool selection processes, commercial and open-source workflow systems, and workflow standards (see `www.workflowpatterns.com` for details). This paper adopts an approach similar to that in [2] although in this case, the focus is on the data perspective. One significant advantage of a patterns-based approach is that it provides a basis for comparison between software offerings without requiring that they share the same conceptual underpinnings. This paper aims to extend the previous work on *Workflow Control Patterns* to the data perspective. It identifies 40 data patterns that recur in workflow systems and describes a selection of these in detail[1]. It also examines their use across six major workflow products, standards and web service composition languages.

2 Workflow and Data Concepts

2.1 Workflow Structure

Before we describe the data perspective in detail, we first present a standard set of definitions for the various components of a workflow system that we will utilise throughout this paper. A *workflow* or *workflow model* is a description of a business process in sufficient detail that it is able to be directly executed by a *workflow management system*. A *workflow model* is composed of a number of *tasks* which are connected in the form of a directed graph. An executing instance of a workflow model is called a *case* or *process instance*. There may be multiple cases of a particular workflow model running simultaneously, however each of these is assumed to have an independent existence and they typically execute without reference to each other. Each invocation of a task that executes is termed a *task instance*. A task instance may initiate one or several task instances when it completes. This is illustrated by an arrow from the completing task to the task being initiated e.g. in Figure 1, task instance B is initiated when task instance A completes. A *task* corresponds to a single unit of work. Four distinct types of task are denoted: *atomic*, *block*, *multiple-instance* and *multiple-instance block*. We use the generic term *components* of a workflow to refer to all of the tasks that comprise a given workflow model. An *atomic task* is one which has a simple, self-contained definition (i.e. one that is not described in terms of other workflow tasks) and only one instance of the task executes when it is initiated. A *block task* is a complex action which has its implementation described in terms of a *sub-workflow*. When a *block task* is started, it passes control to the first task(s) in its corresponding *sub-workflow*. This *sub-workflow* executes to completion and at its conclusion, it passes control back to the *block*

[1] Readers seeking a comprehensive description of all 40 patterns are referred to [10].

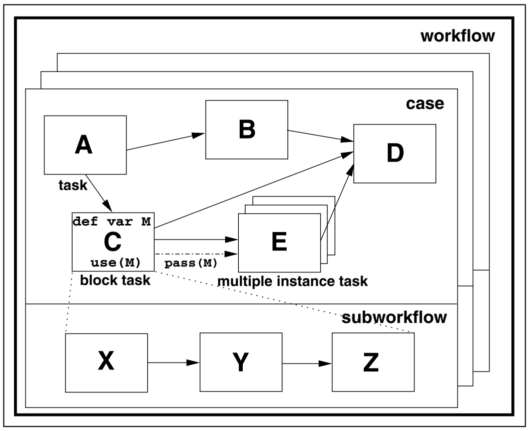


Fig. 1. Components of a workflow

task. E.g. block task C is defined in terms of the sub-workflow comprising tasks, X, Y and Z. A *multiple-instance task* is a task that may have multiple distinct execution instances running concurrently within the same workflow case. Each of these instances executes independently. Only when a nominated number of these instances have completed is the task following the multiple instance task initiated. A *multiple-instance block task* is a combination of the two previous constructs and denotes a task that may have multiple distinct execution instances each of which is block structured in nature (i.e. has a corresponding *sub-workflow*). The control flow between tasks occurs via the *control channel* which is indicated by a solid arrow between tasks. There may also be a distinct *data channel* between workflow tasks which provides a means of communicating *data elements* between two connected tasks which is illustrated with a broken (dash-dot) line. The control and data channels may be combined. Where data elements are passed along a channel between tasks, this is illustrated by the `pass()` relation, e.g. in Figure 1 data element M is passed from task instance C to E. The definition of data elements within the workflow is illustrated by the `def var` *`variable-name`* phrase. Depending on where this appears, the variable may have task, block, case or workflow scope indicating the level at which the data element is bound. The places where a given data element can be accessed are illustrated by the `use()` phrase. In the case of workflow, case and block level data elements, these may be passed between tasks by reference (i.e. the location rather than the value of the data element is passed). This is indicated through the use of the `&` symbol e.g. the `pass(&M)` phrase indicates that the data element M is being passed by reference rather than value.

2.2 Data Characterisation

The patterns presented in this paper are intended to be language independent and do not assume a concrete syntax. In the absence of an agreed workflow model, the aim is to define them in a form that ensures they are applicable to the broadest possible range of workflow systems. As such, we use informal diagrams throughout this paper for illustrating workflow execution instances. The aim of these diagrams is to illustrate the scope of workflow data and the manner in which it is passed between various workflow components. They do not have a formal semantics for the control-flow perspective.

From a data perspective, there are a series of characteristics that occur repeatedly in different workflow modelling paradigms. These can be divided into four distinct groups: *data visibility*, which relate to the definition and scope of data elements and the manner in which they can be utilised by various components of a workflow process; *data interaction*, which focus on the manner in which data is communicated between active elements within and outside of a workflow; *data transfer*, which consider the means by which the actual transfer of data elements occurs between workflow components and describe the various mechanisms by which data elements can be passed across the interface of a workflow component; and *data-based routing*, which characterise the manner in which data elements can influence the operation of other aspects of the workflow, particularly the control flow perspective.

These characteristics are examined in detail in Section 3 and between them, they form the basis for the identification of 40 patterns relevant to the data perspective of PAIS. As validation of the broad applicability of the patterns identified, their occurrence is examined in three major workflow engines (Staffware, WebSphere MQ and COSA), a case handling system (FLOWer), a web service composition language (BPEL4WS) and a workflow standard (XPDL). The results of these evaluations are presented in Section 4.

3 Data Patterns

3.1 Data Visibility

Within the context of a workflow engine, there are a variety of distinct ways in which data elements can be defined and utilised. Typically individual data elements are bound to a specific workflow construct (e.g. a task or a block) and this binding defines the scope in which the data element can be accessed. More generally, it also influences the way in which the data element may be used e.g. to capture *production* information, to manage *control* data or for *communication* with the external environment. Eight data visibility patterns have been identified. The second of these, Block Data, is discussed subsequently in further depth. Lines 1 to 8 of Table 1 provide a complete listing of the data visibility patterns.

Pattern 2 (Block Data)

Description. Block tasks (i.e. tasks which can be described in terms of a corresponding sub-workflow) are able to define data elements which are accessible by each of the components of the corresponding sub-workflow.

Example. All components of the sub-workflow which define the *Assess Investment Risk* block task can utilise the *security details* data element.

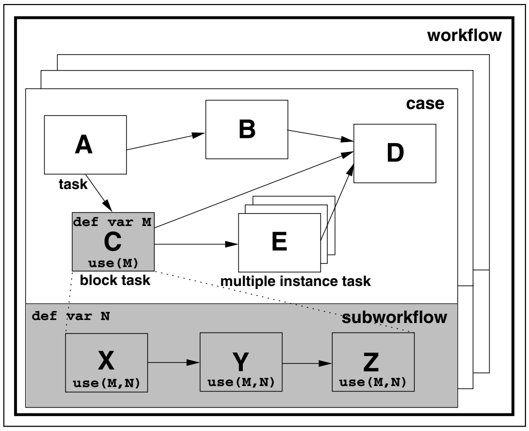


Fig. 2. Block level data visibility

Motivation. The manner in which a block task is implemented is usually defined via its decomposition into a sub-workflow. It is desirable that data elements available in the context of the undecomposed block task are available to all of the components that make up the corresponding sub-workflow. Similarly, it is useful if there is the ability to define new data elements within the context of the sub-workflow that can be utilised by each of the components during execution.

Figure 2 illustrates both of these scenarios, data element M is declared at the level of the block task C and is accessible both within the block task instance and throughout each of the task instances (X, Y and Z) in the corresponding sub-workflow. Similarly data element N is declared within the context of the sub-workflow itself and is available to all task instances in the sub-workflow. Depending on the underlying workflow system, it may also be accessible at the level of the corresponding block task.

Implementation. The concept of block data is widely supported by workflow systems and all but one of the offerings examined in this survey which supported the notion of sub-workflows[2] implemented it in some form. Staffware allows sub-workflows to specify their own data elements and also provides facilities for parent processes to pass data elements to sub-workflows as formal parameters. In WebSphere MQ, sub-workflows can specify additional data elements in the data container that is used for passing data between task instances within the sub-workflow and restrict their scope to the sub-workflow. FLOWer and COSA also provide facilities for specifying data elements within a sub-workflow.

Issues. A major consideration in regard to block-structured tasks within a workflow is the handling of block data visibility where cascading block decompositions are supported and data elements are implicitly inherited by sub-workflows. As an example, in the preceding diagram block data sharing would enable a data element declared within the context of task C to be utilised by task X, but

[2] BPEL4WS which does not directly support sub-workflows is the only exception.

if X were also a block task would this data element also be accessible to task instances in the sub-workflow corresponding to X?

Solutions. One approach to dealing with this issue adopted by workflow tools such as Staffware is to only allow one level of block data inheritance by default i.e. data elements declared in task instance C are implicitly available to X, Y and Z but not to further sub-workflow decompositions. Where further cascading of data elements is required, then this must be specifically catered for. COSA allows a sub-workflow to access all data elements in a parent process and provides for arbitrary levels of cascading[3], however updates to data elements in sub-workflows are not automatically propagated back to the parent task.

3.2 Data Interaction

Data interaction patterns capture the various ways in which data elements can be passed between components in a workflow process and how the characteristics of the individual components can influence the manner in which the trafficking of data elements occurs. Of particular interest is the distinction between the communication of data between components within a workflow engine as against the data-oriented interaction of a workflow component with some form of information resource or service that operates outside of the context of the workflow engine (i.e. in the external environment).

In total, 18 data interaction patterns have been identified – six of these relate to data interaction between internal workflow components and the remaining twelve describe the various situations where data interaction can occur between a workflow component and the external environment. The complete listing of these patterns is presented in lines 9 to 26 of Table 1. In this section, we describe two internal and one external data interaction patterns in detail.

Pattern 9 (Data Interaction between Tasks)

Description. The ability to communicate data elements between one task instance and another within the same case.

Example. The *Determine Fuel Consumption* task requires the *coordinates* determined by the *Identify Shortest Route* task before it can proceed.

Motivation. The passing of data elements between tasks is a fundamental aspect of workflow systems. In many situations, individual tasks execute in their own distinct address space and do not share significant amounts of data on a global basis. This necessitates the ability to move commonly used data elements between distinct tasks as required.

Implementation. All workflow engines examined support the notion of passing parameters from one task to another however, this may occur in a number of distinct ways depending on the relationship between the data perspective and control flow perspective within the workflow. There are three main approaches as illustrated in Figure 3.

[3] Although more than four levels of nesting are not recommended.

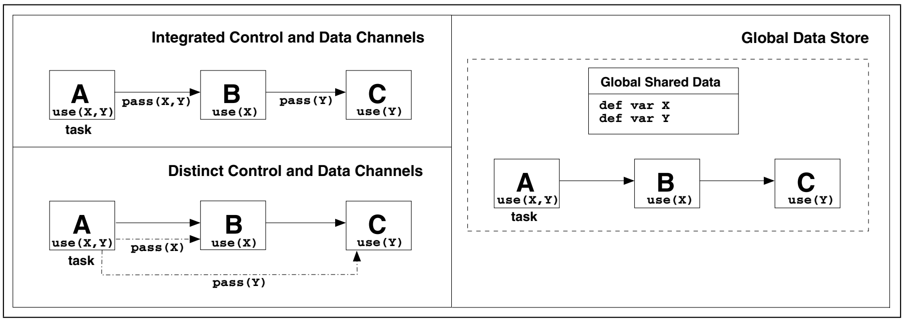


Fig. 3. Approaches to data interaction between tasks

- *Integrated control and data channels* – where both control flow and data are passed simultaneously between tasks utilising the same channel. In the example, task B receives the data elements X and Y at exactly the same time that control is passed to it. Whilst conceptually simple, one of the disadvantages of this approach to data passing is that it requires all data elements that may be used some time later in the workflow process to be passed with the thread of control regardless of whether the next task will use them or not. E.g. task B does not use data element Y but it is passed to it because task C will subsequently require access to it.
- *Distinct data channels* – in which data is passed between workflow tasks via explicit data channels which are distinct from the process control links within the workflow design. Under this approach, the coordination of data and control passing is usually not specifically identified. It is generally assumed that when control is passed to a task that has incoming data channels, the data elements specified on these channels will be available at task commencement.
- *Global data store* – where tasks share the same data elements (typically via access to globally shared data) and no explicit data passing is required. This approach to data sharing is based on tasks having shared *a priori* knowledge of the naming and location of common data elements. It also assumes that the implementation is able to deal with potential concurrency issues that may arise where several task instances seek to access the same data element.

Most of the offerings examined adopt the third strategy. Staffware, FLOWer, COSA and XPDL all facilitate the passing of data through case-level data repositories accessible by all tasks. BPEL4WS utilises a combination of the first and third approaches. Variables can be bound to scopes within a process definition which may encompass a number of tasks, but there is also the ability for messages to be passed between tasks when control passes from one task to another. WebSphere MQ adopts the second mechanism with data elements being passed between tasks in the form of data containers via distinct data channels.

Issues. Where there is no data passing between tasks and a common data store is utilised by several tasks for communicating data elements, there is the potential for concurrency problems to arise, particularly if the case involves parallel execution paths. This may lead to inconsistent results depending on the task execution sequence that is taken.

Solutions. Concurrency control is handled in a variety of different ways by the offerings examined in Section 4. FLOWer avoids the problem by only allowing one active user or process that can update data elements in a case at any time (although other processes and users can access data elements for reading). BPEL4WS supports serialisable scopes which allow compensation handlers to be defined for groups of tasks that access the same data elements. A compensation handler is a procedure that aims to undo or compensate for the effects of the failure of a task on other tasks that may rely on it or on data that it has affected. Staffware provides the option to utilise an external transaction manager (Tuxedo) within the context of the workflow cases that it facilitates.

Pattern 12 (Data Interaction – to Multiple Instance Task)

Description. The ability to pass data elements from a preceding task instance to a subsequent task which is able to support multiple execution instances. This may involve passing the data elements to all instances of the multiple instance task or distributing them on a selective basis.

Examples. The *New Albums List* is passed to the *Review Album* task and one task instance is started for each entry on the list. Each of the *Review Album* task instances is allocated a distinct entry from the *New Albums List* to review.

Motivation. Where a task is capable of being invoked multiple times, a means is required of controlling which data elements are passed to each of the execution instances. This may involve ensuring that each task instance receives all of the data elements passed to it (possibly on a shared basis) or distributing the data elements across each of the execution instances on some predefined basis.

Implementation. There are three potential approaches to passing data elements to multiple instance tasks as illustrated in Figure 4. As a general rule, it is possible either to pass a data element to all task instances or to distribute one item from it (assuming it is a composite data element such as an array or a set) to each task instance. Indeed the number of task instances that are initiated may be based on the number of individual items in the composite data element. The specific approaches are as follows:

– *Instance-specific data passed by value* – this involves the distribution of a data element passed by value to task instances on the basis of one item of the data element per task instance (in the example shown, task instance B_1 receives M[1], B_2 receives M[2] and so on). As the data element is passed by value, each task instance receives a copy of the item passed to it in its own address space. At the conclusion of each of the task instances, the data element is reassembled from the distributed items and passed to the subsequent task instance.
– *Instance-specific data passed by reference* – this scenario is similar to that described above except that the task instances are passed a reference to a

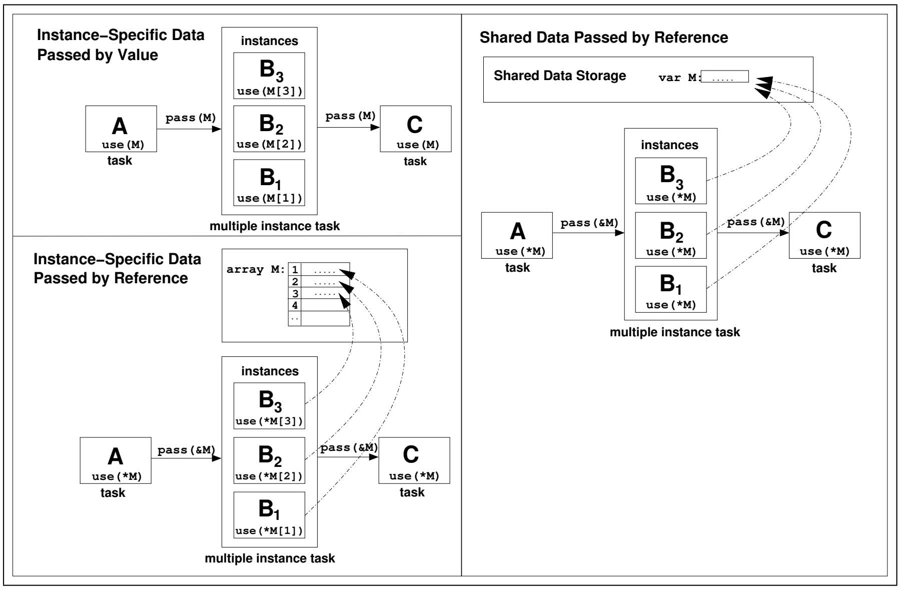


Fig. 4. Data interaction approaches for multiple instance tasks

specific item in the data element rather than the value of the item. This approach obviates the need to reassemble the data element at the conclusion of the task instances.

– *Shared data passed by reference* – in this situation all task instances are passed a reference to the same data element. Whilst this allows all task instances to access the same data element, it does not address the issue of concurrency control should one of the task instances amend the value of the data element (or indeed if it is altered by some other workflow component).

FLOWer provides facilities for instance-specific data to be passed by reference whereby an array can be passed to a designated multiple instance task and specific sub-components of it can be mapped to individual task instances. It also allows for shared data elements to be passed by reference to all task instances.

Issues. Where a task is able to execute multiple times but not all instances are created at the same point, an issue that arises is whether the values of data elements are set for all execution instances at the time at which the multiple instance task is initiated or whether they can be fixed after this occurs but prior to the actual invocation of the task instance to which they relate.

Solutions. In FLOWer, the Dynamic Plan construct allows the data for individual task instances to be specified at any time prior to the actual invocation of the task. The passing of data elements to specific task instances is handled via Mapping Array data structures. These can be extended at any time during the execution of a Dynamic Plan, allowing for new task instances to be created "on the fly" and the data corresponding to them to be specified at the latest possible time.

Pattern 16 (Data Interaction – Environment to Task – Pull-Oriented)

Description. The ability of a workflow task to request data elements from resources or services in the operational environment.

Example. The *Determine Cost* task must request *cattle price* data from the *Cattle Market System* before it can proceed.

Motivation. Workflow tasks require the means to proactively seek the latest information from known data sources in the operating environment during their execution. This may involve accessing the data from a known repository or invoking an external service in order to gain access to the required data elements.

Implementation. Distinct workflow engines support this pattern in a variety of ways however these approaches divide into two categories: *explicit integration mechanisms*, where the workflow system provides specific constructs for accessing data in the external environment and *implicit integration mechanisms*, where access to external data occurs at the level of the programmatic implementations that make up tasks in the workflow process and is not directly supported by the workflow engine. Interaction with external data sources typically utilises interprocess communication (IPC) facilities provided by the operating system facilities such as message queues or remote procedure calls, or enterprise application integration (EAI) mechanisms such as DCOM, CORBA or JMS.

Staffware provides two distinct constructs that support this objective. Automatic Steps allow external systems to be called (e.g. databases or enterprise applications) and specific data items to be requested. Scripts allow external programs to be called either directly at system level or via system interfaces such as DDE (dynamic data exchange) to access required data elements. FLOWer utilises Mapping Objects to extract data elements from external databases. COSA has a number of Tool Agent facilities for requesting data from external applications. XPDL and BPEL4WS provide facilities for the synchronous request of data from other web services. In contrast, WebSphere MQ does not provide any facilities for external integration and requires the underlying programs that implement workflow tasks to provide these capabilities where they are needed.

Issues. One difficulty with this style of interaction is that it can block progress of the requesting case if the external application has a long delivery time for the required information or is temporarily unavailable.

Solutions. The only potential solution to this problem is for the requesting case not to wait for the requested data (or continue execution after a nominated timeout) and to implement some form of asynchronous notification of the required information. The disadvantage of this approach is that it complicates the overall interaction by requiring the external application to return the required information via an alternate path, necessitating the workflow to provide notification facilities.

3.3 Data Transfer Patterns

Data transfer patterns focus on the manner in which the actual transfer of data elements occurs between one workflow component and another. These patterns serve as an extension to those presented in Section 3.2 and aim to capture the various mechanisms by which data elements can be passed across the interface of a workflow component.

The specific style of data passing that is used in a given scenario depends on a number of factors including whether the two components share a common address space for data elements, whether it is intended that a distinct copy of an element is passed as against a reference to it and whether the component receiving the data element can expect to have exclusive access to it. These variations give rise to seven distinct patterns as listed in lines 27 to 33 of Table 1. In this section, we describe one pattern in detail – Data transfer by value - incoming.

Pattern 27 (Data Transfer by Value – Incoming)

Description. The ability of a workflow component to receive incoming data elements by value relieving it from the need to have shared names or common address space with the component(s) from which it receives them.

Example At commencement, the *Identify Successful Applicant* task receives values for the *required role* and *salary* data elements.

Motivation. Under this scenario, data elements are passed as values between communicating workflow components. There is no necessity for each workflow component to utilise a common naming strategy for the data elements or for components to share access to a common data store in which the data elements reside. This enables individual components to be written in isolation without specific knowledge of the manner in which data elements will be passed to them or the context in which they will be utilised.

Implementation. This approach to data passing is commonly used for communicating data elements between tasks that do not share a common data store or wish to share task-level (or block-level) data items. The transfer of data between workflow components is typically based on the specification of mappings between them identifying source and target data locations. In this situation, there is no necessity for common naming or structure of data elements as it is only the data values that are actually transported between interacting components.

WebSphere MQ utilises this approach to data passing in conjunction with distinct data channels. Data elements from the originating workflow task instance are coalesced into a data container. A mapping is defined from this data container to a distinct data container which is transported via the connecting data channel between the communicating tasks. A second mapping is then defined from the data container on the data channel to a data container in the receiving task. BPEL4WS provides the option to pass data elements between activities using messages – an approach which relies on the transfer of data between workflow components by value. XPDL provides more limited support for data transfer by value between a block task and sub-workflow. As all data elements are case level, there is no explicit data passing between tasks.

3.4 Data-Based Routing

Whereas other groups of patterns focus on characteristics of data elements in isolation from other workflow perspectives (i.e. control, resource, organisational etc.), data-based routing patterns aim to capture the various ways in which data elements can interact with other perspectives and influence the overall operation of the workflow. Constructs such as pre-conditions, post-conditions, triggers and splits are characterised by these patterns and seven of them have been identified as listed in lines 34 to 40 of Table 1. We do not discuss these patterns in detail here and interested readers are referred to [10] for more details.

4 Evaluation of Existing Workflow Products

This section presents the results of a detailed evaluation of support for the 40 workflow data patterns by six workflow systems, standards and web service composition languages. A broad range of offerings were chosen for this review in order to validate the applicability of each of the patterns to the various types of tools that fall under the "workflow umbrella" [6]. Specific tools and languages evaluated were Staffware Process Suite v9, WebSphere MQ Workflow 3.4, FLOWer 3.0, COSA 4.2, XPDL 1.0 and BPEL4WS 1.1. A three point assessment scale is used with "+" indicating direct support for the pattern, "+/–" indicating partial support and "–" indicating that the pattern is not implemented. Specific rating criteria have been devised and are detailed in [10].

Lines 1 to 8 indicate the various levels of data construct visibility supported within the tools. As a general rule, it can be seen that individual products tend to favour either a task-level approach to managing production data and pass data elements between task instances or they use a shared data store at case level. The only exception to this being COSA which fully supports data at both levels. A similar result can be observed for workflow and environment data with most workflow products fully supporting one or the other (Staffware being the exception here). The implication of this generally being that globally accessible data can either be stored in the workflow product or outside of it (i.e. in a database). XPDL and BPEL4WS are the exceptions although this outcome seems to relate more to the fact that there is minimal consideration for global data facilities within these specifications. Lines 9 to 14 list the results for internal data passing. All offerings supported task-to-task interaction and block task-to-sub-workflow interaction[4]. The notable omissions here were the general lack of support for handling data passing to multiple instance tasks (FLOWer being the exception) and the lack of integrated support for data passing between cases. Lines 15 to 26 indicate the ability of the workflow products to integrate with data sources and applications in the operating environment. WebSphere MQ, FLOWer and COSA demonstrate a broad range of capabilities in this area. XPDL and BPEL4WS clearly have limited potential for achieving external integration other than with web services. Lines 27 to 33 illustrate the mechanisms

[4] BPEL4WS being the exception given its lack of support for sub-workflows.

Table 1. Support for Data Patterns in Workflow Systems

Nr	Pattern	Staffware	WebSphere	FLOWer	COSA	XPDL	BPEL4WS
1	Task Data	–	+/–	+/–	+	–	+/–
2	Block Data	+	+	+	+	+	–
3	Scope Data	–	–	+/–	–	–	+
4	Folder Data	–	–	–	+	–	–
5	Multiple Instance Data	+/–	+	+	+	+	–
6	Case Data	+/–	+	+	+	+	+
7	Workflow Data	+	+	–	+/–	+/–	–
8	Environment Data	+	+/–	+	+	–	+
9	Data Interaction between Tasks	+	+	+	+	+	+
10	Data Interaction – Block Task to Sub-workflow	+	+	+/–	+/–	+	–
11	Data Interaction – Sub-workflow to Block Task	+	+	+/–	+/–	+	–
12	Data Interaction – to Multiple Instance Task	–	–	+	–	–	–
13	Data Interaction – from Multiple Instance Task	–	–	+	–	–	–
14	Data Interaction – Case to Case	+/–	+/–	+/–	+	+/–	+/–
15	Data Interaction – Task to Env. – Push	+	+/–	+	+	+	+
16	Data Interaction – Env. to Task – Pull	+	+/–	+	+	+	+
17	Data Interaction – Env. to Task – Push	+/–	+/–	+/–	+	–	+/–
18	Data Interaction – Task to Env. – Pull	+/–	+/–	+/–	+	–	+/–
19	Data Interaction – Case to Env. – Push	–	–	+	–	–	–
20	Data Interaction – Env. to Case – Pull	–	–	+	–	–	–
21	Data Interaction – Env. to Case – Push	+/–	+/–	+	+	–	–
22	Data Interaction – Case to Env. – Pull	–	–	+	+	–	–
23	Data Interaction – Workflow to Env. – Push	–	+/–	–	–	–	–
24	Data Interaction – Env. to Workflow – Pull	+/–	–	–	–	–	–
25	Data Interaction – Env. to Workflow – Push	–	+/–	–	–	–	–
26	Data Interaction – Workflow to Env. – Pull	+	+	–	+	–	–
27	Data Transfer by Value – Incoming	–	+	–	+/–	+/–	+
28	Data Transfer by Value – Outgoing	–	+	–	+/–	+/–	+
29	Data Transfer – Copy In/Copy Out	–	–	+/–	–	+/–	–
30	Data Transfer by Reference – Unlocked	+	–	+	+	+	+
31	Data Transfer by Reference – Locked	–	–	+/–	–	–	+/–
32	Data Transformation – Input	+/–	–	+/–	–	–	–
33	Data Transformation – Output	+/–	–	+/–	–	–	–
34	Task Precondition – Data Existence	+	–	+	+	–	+/–
35	Task Precondition – Data Value	+	–	+	+	+	+
36	Task Postcondition – Data Existence	+/–	+	+	–	–	–
37	Task Postcondition – Data Value	+/–	+	+	–	–	–
38	Event-based Task Trigger	+	+/–	+	+	–	+
39	Data-based Task Trigger	–	–	+	+	–	+/–
40	Data-based Routing	+/–	+	+/–	+	+	+

used by individual workflow engines for passing data between components. Generally this occurs by value or by reference. There are two areas where there is clear opportunity for improvement. First, support for concurrency management where data is being passed between components – only FLOWer and BPEL4WS offered some form of solution to this problem. Second, the transformation of data elements being passed between components – only Staffware provides a fully functional capability for dealing with potential data mismatches between sending and receiving components although its applicability is limited. Lines 34 to 40 indicate the ability of the data perspective to influence the control perspective within each product. FLOWer demonstrates outstanding capability in this area and Staffware, WebSphere MQ and COSA also have relatively good integration of the data perspective with control flow although each of them lack some degree of task pre and postcondition support. Similar comments apply to XPDL which has significantly more modest capabilities in this area and completely lacks any form of trigger support. BPEL4WS would also benefit from better pre and postcondition support and lacks data-based triggering.

Through these evaluations a number of insights have been gained in relation to the current level of data support in workflow systems. Current workflow modelling techniques centre on the capture of control-flow and offer minimal support for documenting data requirements. This difficulty extends into workflow design tools which typically provide fragmented facilities for incorporating data requirements in workflows. There is little support for multiple instance tasks in current tools (which provide for true task parallelism) and where it exists, the level of data support is minimal. There also appears to have been little learnt from the database field and the evaluations revealed limited support for data persistence and concurrency handling in the tools examined.

5 Conclusion

This paper has identified 40 new *workflow data patterns* which describe the manner in which data is defined and utilised in workflow systems. The main contribution of this work is that it is the first systematic attempt to provide a taxonomy of data usage in workflow systems in a technology-independent manner. Validation of the applicability of these patterns has been achieved through a detailed review of six workflow systems, standards and web service composition languages. The results of this review indicate that the data patterns identified are applicable not only to workflow systems but that they are also of relevance to process-aware information systems more generally.

Evaluation of pattern support in current tools gives a valuable insight into the operation of workflow systems and the data patterns identified have a number of practical uses. First, they provide an effective foundation for training workflow designers and developers. Second, they provide a means of assessing tool capabilities and are particularly useful in tool evaluation and selection exercises (e.g. tender evaluations). Finally, they offer the basis for vendors to identify functionality gaps and potential areas for enhancement.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(3):5–51, 2003.
3. D. Barbara, S. Mehrotra, and M. Rusinkiewicz. INCAs: Managing Dynamic Workflows in Distributed Environments. *Journal of Database Management*, 7(1):5–15, 1996.
4. M. Dumas and A. ter Hofstede. UML Activity Diagrams as a Workflow Specification Language. In M. Gogolla and C. Kobryn, editors, *Proceedings of the Fourth International Conference on the Unified Modeling Language (UML 2001)*, LNCS 2185, pages 76–90, Toronto, Canada, 2001. Springer.
5. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Boston, USA, 1995.
6. D. Georgakopoulos, M.F. Hornick, and A.P. Sheth. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119–153, 1995.
7. S. Jablonski and C. Bussler. *Workflow Management: Modeling Concepts, Architecture and Implementation.* Thomson Computer Press, London, UK, 1996.
8. C.B. Medeiros, G. Vossen, and M. Weske. WASA: A Workflow-Based Architecture to Support Scientific Database Applications. In N. Revell and A.M. Tjoa, editors, *Proceedings of the 6th International Workshop and Conference on Database and Expert Systems Applications (DEXA)*, pages 574–583, London, UK, 1995. Springer.
9. M. Reichert and P. Dadam. ADEPTflex - Supporting Dynamic Changes of Workflows Without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
10. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns (Revised Version). Technical Report FIT-TR-2004-01, Queensland University of Technology, Brisbane, Australia, 2004. `http://www.bpmcenter.org`.
11. A.-W. Scheer. *ARIS - Business Process Modelling.* Springer, Berlin, Germany, 2000.
12. G. Vossen and M. Weske. The WASA2 Object-Oriented Workflow Management System. In A. Delis, C. Faloutsos, and S. Ghandeharizadeh, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 1999)*, pages 587–589, Philadelphia, Pennsylvania, USA, 1999. ACM Press.
13. D. Wodtke, J. Weissenfels, G. Weikum, and A. Kotz-Dittrich. The Mentor Project: Steps Towards Enterprise-Wide Workflow Management. In S.Y.W. Su, editor, *Proceedings of the 12th International Conference on Data Engineering (ICDE 1996)*, pages 556–565, New Orleans, Louisiana, USA, 1996. IEEE Computer Society.

Actor-Oriented Design of Scientific Workflows*

Shawn Bowers[1] and Bertram Ludäscher[2]

[1] UC Davis Genome Center
[2] Department of Computer Science, University of California, Davis
{sbowers, ludaesch}@ucdavis.edu

Abstract. Scientific workflows are becoming increasingly important as a unifying mechanism for interlinking scientific data management, analysis, simulation, and visualization tasks. Scientific workflow systems are problem-solving environments, supporting scientists in the creation and execution of scientific workflows. While current systems permit the creation of executable workflows, conceptual modeling and design of scientific workflows has largely been neglected. Unlike business workflows, scientific workflows are typically highly data-centric naturally leading to dataflow-oriented modeling approaches. We first develop a formal model for scientific workflows based on an actor-oriented modeling and design approach, originally developed for studying models of complex concurrent systems. Actor-oriented modeling separates two modeling concerns: component *communication* (dataflow) and overall workflow *coordination* (orchestration). We then extend our framework by introducing a novel *hybrid* type system, separating further the concerns of conventional data modeling (*structural data type*) and conceptual modeling (*semantic type*). In our approach, semantic and structural mismatches can be handled independently or simultaneously, and via different types of *adapters*, giving rise to new methods of scientific workflow design.

1 Introduction

Scientific workflows are quickly becoming recognized as an important unifying mechanism to combine scientific data management, analysis, simulation, and visualization tasks. Scientific workflows often exhibit particular traits, e.g., they can be data-intensive, compute-intensive, analysis-intensive, and visualization-intensive, thus covering a wide range of applications from low-level "plumbing workflows" of interest to Grid engineers, to high-level "knowledge discovery workflows" for scientists [11]. Consequently, workflows steps can have very different granularities and may be implemented as shell scripts, web services, local application calls, or as complex subworkflows.

A *scientific workflow system* is a problem-solving environment that aims at simplifying the task of "gluing" these steps together to form executable data management and analysis pipelines. While current systems permit the creation of executable workflows, conceptual modeling and design of scientific workflows has been largely neglected. Unlike business workflows, scientific workflows are typically highly data-centric, naturally leading to dataflow-oriented modeling approaches, while business workflow modeling

* This work supported in part by NSF/ITR 0225673 (GEON), NSF/ITR 0225676 (SEEK), NIH/NCRR 1R24 RR019701-01 (BIRN-CC), and DOE DE-FC02-01ER25486 (SDM).

is dominated by control, event, and task-oriented approaches [17], making them less suitable for the modeling challenges of scientific workflows.

This paper addresses three important problems in scientific-workflow design and engineering. First, in existing systems it is often unclear what constitutes a scientific workflow, and there are few if any abstract models available to describe scientific workflows. (By abstract model, we mean a model for scientific workflows analogous to data models in database management.) Second, existing systems do not support the end-to-end development of scientific workflows, in particular, design methods and frameworks for the early stages of conceptual design do not exist. And third, in scientific workflow systems such as KEPLER [11] that aim at providing a unified environment where workflows and their components can be shared and reused, mechanisms do not exist that support the discovery, reuse, and adaptation of existing workflows and components.

To address these issues, we first develop a *formal model for scientific workflows* (Section 3) based on an actor-oriented modeling approach, originally developed for studying complex concurrent systems [9]. A benefit of actor-oriented modeling is that it separates two distinct modeling concerns: component *communication* (dataflow) and overall workflow *coordination* (a.k.a. orchestration). We then extend this framework by introducing a novel *hybrid type system*, separating further the concerns of conventional data modeling (*structural data type*) and conceptual modeling (*semantic type*). The separation of types facilitates the *independent* validation of structural and semantic type constraints and offers a number of benefits for scientific workflow design and component reuse. Structural and semantic types can also be explicitely *linked* in our approach, using special (hybridization) constraints. These constraints can be exploited in various ways, e.g., to further propagate and refine known (structural or semantic) types in scientific workflows, or to infer (partial) structural mappings between structurally incompatible (but semantically compatible) workflow components.

Based on our formal model, we also introduce a number of basic *modeling primitives* that a workflow designer can apply to evolve a formal scientific workflow design in a stepwise, controlled manner (Section 4). The different modeling primitives give rise to distinct *design strategies*, including task-driven vs. data-driven, structure-driven vs. semantics-driven, and top-down vs. bottom-up. Two important design primitives are *actor replacement* and *adapter insertion*. Both primitives, when combined with the hybrid type system, yield powerful new component discovery and adaptation mechanisms.

2 Preliminaries: Business vs. Scientific Workflows and KEPLER

The characteristics and requirements of scientific workflows partially overlap those of business workflows. Historically, business workflows have roots going back to office automation systems, and more recently gained momentum in the form of business process modeling and business process engineering [2,16,18]. Today we see influences of business workflow standards in web-service choreography standards. Examples include the Business Process Execution Language for Web Services (BPEL4WS)[1], a merger

[1] `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`

of IBM's WSFL and Microsoft's XLANG, as well as ontology-based web-service approaches such as OWL-S[2]. When analyzing the underlying design principles and execution models of business workflow approaches, a focus on control-flow patterns and events becomes apparent, whereas dataflow is often a secondary issue.

Scientific workflow systems, on the other hand, tend to have execution models that are much more dataflow-oriented. Examples include academic systems such as KEPLER [11], Taverna [15], and Triana [12], and commercial systems such as Inforsense's DiscoveryNet, Scitegic's Pipeline-Pilot, and National Instrument's LabView. With respect to their modeling paradigm and workflow execution models, these systems are closer to visual dataflow programming languages for scientific data and services than to the more control-flow and task-oriented business workflow systems, or to their early scientific workflow predecessors [13,1].

The difference between dataflow and control-flow orientation can also be observed in the underlying formalisms. For example, visualizations of business workflows often resemble flowcharts, state transition diagrams, or UML activity diagrams, all of which emphasize events and control-flow over dataflow. Formal analysis of workflows usually involves studying their control-flow patterns [8,5]. Conversely, the underlying execution model of current scientific workflow systems usually resembles dataflow process networks [10], having traditional application areas in digital signal processing and electrical engineering. Dataflow-oriented approaches are applicable at very different levels of granularity, from low-level CPU operations found in processor architectures, over embedded systems, to high-level programming paradigms such as flow-based programming [14]. Scientific workflow systems and visualization pipeline systems can also be seen as dataflow-oriented problem-solving environments [7] that scientists use to analyze and visualize their data.

Actor-Oriented Workflow Modeling in KEPLER. The KEPLER scientific workflow system is an open-source application, with contributing members from various application-oriented research projects. KEPLER aims at developing generic solutions to the process and application-integration challenges of scientific workflows. Figure 1 shows a snapshot of KEPLER running a bioinformatics scientific workflow.

KEPLER extends the PTOLEMY II system, developed for modeling heterogeneous and concurrent systems and engineering applications, to support scientific workflows. In KEPLER, users develop workflows by selecting appropriate components called "actors" (e.g., from actor libraries or by wrapping web services as actors) and placing them on the design canvas, after which they can be "wired" together to form the desired workflow graph. As shown in Figure 1, workflows can also be hierarchically structured. Actors have *input ports* and *output ports* that provide the communication interface to other actors. Control-flow elements such as branching and loops are also supported. A unique feature of PTOLEMY II (and thus of KEPLER) is that the overall execution and component interaction semantics of a workflow is not buried inside the components themselves, but rather factored out into a separate component called a *director*. PTOLEMY II supports a large number of different directors, each one corresponding to a unique model of computation. Taken together, workflows, actors, ports, connections, and directors represent the basic building blocks of actor-oriented modeling.

[2] `http://www.daml.org/services/owl-s/`

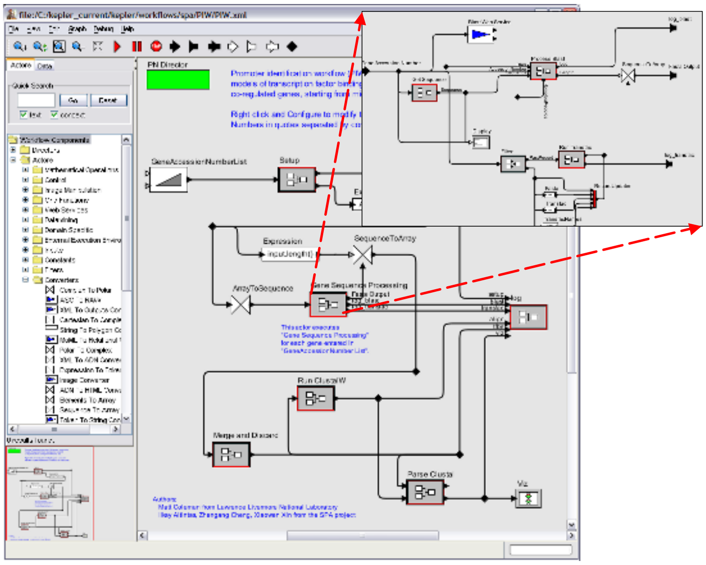


Fig. 1. A bioinformatics workflow in KEPLER: the *composite actor* (center) contains a nested *subworkflow* (upper right); workflow steps include remote service invocation and data transformation; and the execution model is enforced by a *director* (green box)

3 A Formal Model of Actor-Oriented Scientific Workflows

This section further defines actor-oriented modeling and its application to scientific workflows. We describe a formal model for scientific workflows and a rich typing system for workflows and workflow components that considers both structural and semantic types. We also briefly describe the use of directors for specifying workflow computation models, which simplifies the task of defining workflows within KEPLER and, along with the typing system, can facilitate the reuse of workflow components.

3.1 Actor-Oriented Hierarchical Workflow Graphs

Workflow Graphs. An actor-oriented *workflow graph* $W = \langle \mathbf{A}, \mathbf{D} \rangle$ consists of a set $\mathbf{A}$ of *actors* representing components or tasks and a set of *dataflow connections* $\mathbf{D}$ connecting actors via data ports. Actors have well defined interfaces and generally speaking, unlike a software agent, are passive entities that given some input data, produce output data (according to their interface). Actors communicate by passing data tokens between their ports.

Ports. Each actor $A \in \mathbf{A}$ has an associated set $\mathsf{ports}(A)$ of *data ports*, where each $p \in \mathsf{ports}(A)$ is either an *input* or *output*, i.e., $\mathsf{ports}(A) = \mathsf{in}(A) \uplus \mathsf{out}(A)$ is a disjoint union of *input ports* and *output ports*, respectively. We can think of $\mathsf{ports}(A)$ as the input/output *signature* Σ_A of A, denoted $A :: \mathsf{in}(A) \longrightarrow \mathsf{out}(A)$.[3]

[3] We may also distinguish $\mathsf{par}(A) \subseteq \mathsf{in}(A)$, the *parameter ports* of A, distinct from "regular" data input ports, and used to model different actor "configurations".

Dataflow Connections. Let $\mathsf{in}(W) = \bigcup_{A \in \mathbf{A}} \mathsf{in}(A)$ be the set of all of input ports of W; the sets $\mathsf{out}(W)$ and $\mathsf{ports}(W)$ are defined similarly. A *dataflow connection* $d \in \mathbf{D}$ is a directed hyperedge $d = \langle \mathbf{o}, \mathbf{i} \rangle$, simultaneously connecting n output ports $\mathbf{o} = \{o_1, \ldots, o_n\} \subseteq \mathsf{out}(W)$ with m input ports $\mathbf{i} = \{i_1, \ldots, i_m\} \subseteq \mathsf{in}(W)$. Intuitively, we can think of $d = \langle \mathbf{o}, \mathbf{i} \rangle$ as consisting of a *merge step* $\mathsf{merge}(d) = \mathbf{o}$ that combines data tokens from the output ports $\mathbf{o}$, and a *distribute step* $\mathsf{distrib}(d) = \mathbf{i}$ that distributes the merged tokens to the input ports $\mathbf{i}$.[4]

A dataflow connection $d = \langle \{o_1\}, \{i_1\} \rangle$ between a single output port and a single input port corresponds to a directed edge $o_1 \xrightarrow{d} i_1$. In general, however, we represent d as an auxiliary connection node having n incoming edges from all output ports $o \in \mathbf{o}$ and m outgoing edges to all input ports $i \in \mathbf{i}$. Dataflow connection $d \in \mathbf{D}$ is called *well-oriented*, if it connects at least one output and one input port. In this way, a directed dataflow dependency between ports is induced.

Workflow Abstraction and Refinement. Abstraction and refinement are crucial modeling primitives. When abstracting a workflow W, we would like to "collapse" it into a single, *composite actor* A_W (hiding W "inside"). Conversely, we might want to refine an actor A by further specifying it via a *subworkflow* W_A, thereby turning A into a composite actor with W_A "inside" (cf. Figures 1 and 3). In both cases, we need to make sure that the i/o-signature Σ_A of the composite actor matches the i/o-signature Σ_W of the contained subworkflow.

Let $W = \langle \mathbf{A}, \mathbf{D} \rangle$ be a workflow. The *free ports* of W are all ports that do not participate in any data connection, i.e., $\mathsf{freeports}(W) := \{p \mid \text{for all } d \in \mathbf{D} : p \notin d\}$. A workflow designer might not want to expose all free ports externally when abstracting W into a composite actor A_W. Instead the i/o-signature is often limited to a subset Σ_W of distinguished ports.

Composite Actors. A *composite actor* A_W is a pair $\langle W, \Sigma_W \rangle$ comprising a *subworkflow* W and a set of distinguished ports $\Sigma_W \subseteq \mathsf{freeports}(W)$, the *i/o-signature* of W. We require that the i/o-signatures of the subworkflow W and of the composite actor A_W containing W match, i.e., $\Sigma_W = \mathsf{ports}(A_W)$.

Hierarchical Workflow Graphs. A *hierarchical workflow* $W = \langle \mathbf{A}, \mathbf{D}, \Sigma \rangle$ is defined like a workflow graph, with the difference that actors might be composite. Inductively, subworkflows can be hierarchical, so that any level of nesting can be modeled. For uniformity, we also include the distinguished i/o-signature Σ of the top-level workflow.

3.2 Models of Computation

Following the paradigm of *separation of concerns*, the actor-oriented workflow graphs introduced above only specify communication links (dataflow) between components or tasks (represented by actors), and—in the case of hierarchical workflows—their nesting structure via composite actors. However, the workflow execution semantics or *model of*

[4] The semantics of merging and distributing tokens through dataflow connections is a *separate concern* that is deliberately left unspecified. Instead, this execution semantics is defined separately via *directors*.

computation is deliberately left unspecified. In PTOLEMY II a new modeling primitive called a *director* is used to represent the particular choice of model of computation [9].

Thus, we can extend our definition of workflow (graph) W to include a model of computation by means of a director M, i.e., $W = \langle \mathbf{A}, \mathbf{D}, \Sigma, M \rangle$. In the case of the unspecified merge/distribute semantics of a data connection node $d = \langle \mathbf{o}, \mathbf{i} \rangle$ above, a director M may prescribe, e.g., the merge semantics to be one of the following: non-deterministic (the token arrival order is unspecified by M); time-dependent and deterministic (tokens are merged according to their timestamps); or time-independent and deterministic (e.g., "round robin" merging of tokens, or "zipping" together tokens from all input ports, creating a single record token). Similarly, different distribution semantics may be prescribed by M: deterministic copy (replicate each incoming token on all outputs); deterministic round robin (forward a token to alternating outputs); or nondeterministic round robin (randomly choose an output port).

More generally, a model of computation specifies all inter-actor communication behavior, separating the concern of *orchestration* (director) from the concern of *actor execution*. The PTOLEMY II system comes with a number of directors including:

- *Synchronous Dataflow* (SDF): Actors communicate through data connections corresponding to queues and send or receive a fixed number of tokens each time they are fired. Actors are fired according to a predetermined static schedule. Synchronous dataflow models are highly analyzable and have been used to describe hardware and software systems.
- *Process Network* (PN): A generalisation of SDF in which each actor executes as a separate thread or process, and where data connections represent queues of unbounded size. Thus actors can always write to output ports, but may get suspended (blocked) on input ports witout a sufficient number of data tokens. The PN model of computation is closely related to the Kahn/MacQueen semantics of process networks.
- *Continuous Time* (CT): Actors communicate through data connections, which represent the value of a continuous time signal at a particular point in time. At each time point, actors compute their output based on their previous input and the tentative input at the current time, until the system stabilizes. When combined with actors that perform numerical integration with good convergence behavior, such models are conceptually similar to ordinary differential equations and are often used to model physical processes.
- *Discrete Event* (DE): Actors communicate through a queue of events in time. Events are processed in global time order, and in response to an event an actor is permitted to emit events at the present or in the future, but not in the past. Discrete event models are widely used to model asynchronous circuits and instantaneous reactions in physical systems.

3.3 Structural and Semantic Typing of Scientific Workflows

The formal model described above separates the concerns of component communication (*dataflow connections*) from the overall model of computation (a.k.a. *orchestration*), imposed by the director. This separation achieves a form of *behavioral polymorphism* [9], resulting in more reusable actor components and subworkflows. In a sense, the actor-oriented modeling approach "factors out" the concern of component coordination and centralizes it at the director.

As mentioned in Section 2, scientific workflows are typically data-oriented. The modeling primitives so far, however, have been agnostic about data types. We introduce a novel *hybrid type system* for modeling scientific data that separates *structural data types* and *semantic data types*, but allows them to be explicitly linked using *hybridization constraints*.

Structural Types. Let $\mathcal{S}$ be a language for describing structural data types. For example, $\mathcal{S}$ may be one of XML Schema, XML DTD, PTOLEMY II's token type system, or any other suitable data model or type system for describing structural aspects of data such as the relational model, an object-oriented data model, or a programming language type system (e.g., a polymorphic Hindley-Milner system).

Any port $p \in \mathsf{ports}(W)$ may have a *structural data type* $\mathtt{s} = \mathsf{dt}(p)$, where $\mathtt{s} \in \mathcal{S}$ is a type expression constraining the allowed set of values that the port p can accept (for an input port $p \in \mathsf{in}(W)$) or produce (for an output port $p \in \mathsf{out}(W)$). When using XML Schema as $\mathcal{S}$, e.g., the structural data type of a port is a concrete XML Schema type such as `xsd:date` or any user-defined type. If $\mathcal{S}$ is the relational model, $\mathtt{s}$ describes the tuple or table type of p.

Semantic Types. Let $\mathcal{O}$ be a language for expressing semantic types. By this we mean, in particular, suitable logics for expressing *ontologies*. For example, $\mathcal{O}$ might be a description logic ontology (expressed, e.g., in OWL-DL).

Any port $p \in \mathsf{ports}(W)$ may have a *port semantic type* $\mathrm{C} = \mathsf{st}(p)$, where C denotes a *concept expression* over $\mathcal{O}$. For example, $\mathrm{C}_1 = \mathsf{st}(p_1)$ might be defined as

$$\text{MEASUREMENT} \sqcap \forall \text{ITEMMEASURED.SPECIESOCCURRENCE} \qquad (\mathrm{C}_1)$$

indicating that the port p_1 accepts (or produces) data tokens that are measurements where the measured item is a species occurrence (as opposed to, e.g., a temperature).[5] In addition to port semantic types, any actor $A \in \mathbf{A}$ may also be associated with an *actor semantic type*, categorizing the overall function or purpose of A.[6]

Well-Typed Workflows. Structural and semantic types facilitate the design and implementation of workflows by constraining the possible values and interpretations of data in a scientific workflow W. Another advantage is that the scientific workflow system can validate data connections. For example, if the workflow designer connects two ports $p_1 \xrightarrow{d} p_2$ with structural types $\mathtt{s}_1 = \mathsf{dt}(p_1)$ and $\mathtt{s}_2 = \mathsf{dt}(p_2)$, the system can check whether this connection satisfies the implied subtype constraint $\mathtt{s}_1 \preceq \mathtt{s}_2$. Similarly, for semantic types $\mathrm{C}_1 = \mathsf{st}(p_1)$ and $\mathrm{C}_2 = \mathsf{st}(p_2)$, the system can check whether the implied concept subsumption $\mathrm{C}_1 \sqsubseteq \mathrm{C}_2$ holds.

3.4 Hybrid Types for Scientific Workflows

Structural and semantic types can be considered independently from one another. For example, a workflow designer might start by modeling semantic types and only later in

[5] We note that terms within a concept expression may be from distinct ontologies.

[6] Typically the vocabularies chosen for semantic port types and semantic actor types are disjoint, with the former denoting "objects" and the latter denoting "actions" or "tasks".

the design process be concerned with structural types (cf. Section 4). Conversely, when reverse-engineering existing executable workflows, structural types might be given first; and only later are semantic types introduced for the purpose of facilitating workflow integration.

Treating semantic and structural types independently offers a number of benefits, and is primarily motivated by the desire to easily interoperate legacy workflow components and components created by independent groups within KEPLER. Decoupling the structural and semantic aspects of workflow types facilitates the use of more standard and generic structural data types, while still allowing the specific semantic constraints of the data to be expressed. Also, one can provide or refine semantic types without altering the underlying structural type, can search for all components having a particular semantic type (regardless of the structural type used), and can provide multiple semantic types for a single component (e.g., drawn from distinct ontologies).

An additional feature of hybrid types is the ability to not only independently consider structural and semantic types, but also interrelate them by a constraint mechanism called *hybridization*. Thus, in general, a hybrid type has three (optional) components, the structural type, the semantic type, and the hybridization constraint.

Formally, let $\mathcal{H}$ be a language of (hybridization) *constraints*, i.e., linking structural and semantic type information. We express constraints from $\mathcal{H}$ in logic, thus requiring that structural and semantic types are expressed in a logic formalism as well. For structural types this means that for any $\mathtt{s} \in \mathcal{S}$ and any logic query expression $e(\bar{x})$ over the set $\mathsf{inst}(s)$ of instances of $\mathtt{s}$, we can evaluate $e(\bar{x})$ on a particular data instance $I \in \mathsf{inst}(s)$, returning a list[7] of variable bindings $[\ \bar{x} \mid I \models e(\bar{x})\ ]$, i.e., those parts of I that satisfy the query $e(\bar{x})$.[8]

For example, given the structural (relational) type $\mathtt{s}_1 = \mathtt{r}(\mathtt{site}, \mathsf{day}, \mathsf{spp}, \mathtt{occ})$ and the above semantic type C_1, the following constraint α_1 "hybridizes" $\mathtt{s}_1$ and C_1:

$$\begin{aligned}&\forall x_{\mathtt{site}}, x_{\mathsf{day}}, x_{\mathsf{spp}}, x_{\mathtt{occ}}\ \exists y :\ \mathtt{r}(x_{\mathtt{site}}, x_{\mathsf{day}}, x_{\mathsf{spp}}, x_{\mathtt{occ}}) \longrightarrow \\ &\qquad \textsc{Measurement}(y) \wedge \textsc{itemMeasured}(y, x_{\mathtt{occ}}) \wedge \\ &\qquad \textsc{SpeciesOccurrence}(x_{\mathtt{occ}})\end{aligned} \tag{α_1}$$

Here, the left-hand side of the implication corresponds to a query expression $e(\bar{x})$ that extracts the item being measured from a relational measurement record. The right-hand side of the implication asserts the existence of a MEASUREMENT y whose ITEMMEASURED $x_{\mathtt{occ}}$ is a SPECIESOCCURRENCE. Note that a hybridization constraint such as α_1 can be seen as a "semantic annotation" of the data structure $\mathtt{s}_1$ (the left-hand side of the constraint) with a concept expression (the right-hand side of the constraint).

Exploiting Hybrid Types. By interlinking the otherwise independent structural and semantic type systems, additional inferences can be made. Consider a data connection d that connects two ports $p_1 \xrightarrow{d} p_2$ having *incompatible* structural types $\mathtt{s}_1 = \mathsf{dt}(p_1)$ and $\mathtt{s}_2 = \mathsf{dt}(p_2)$, i.e., where $\mathtt{s}_1$ is *not* a subtype of $\mathtt{s}_2$, denoted $\mathtt{s}_1 \not\preceq \mathtt{s}_2$. Given (hybridization) constraints α_1 and α_2 that map parts of $\mathtt{s}_1$ and $\mathtt{s}_2$ to a common ontology, one can indirectly identify structural correspondences between parts of $\mathtt{s}_1$ and $\mathtt{s}_2$ by

[7] We consider variable binding *lists* to accomodate order-sensitive data models such as XML; for unordered models a *set* of bindings can be returned.

[8] Here, $\bar{x} = x_1, \ldots, x_n$ denotes a vector of logical variables.

"going through the ontology." Technically, this approach is achieved by a resolution-based reasoning technique called the chase.[9]

Exploiting I/O-Constraints. Moreover, for an actor $A \in \mathbf{A}$, a set Φ_{io} of *i/o-constraints* may be given, inter-relating the input and output ports of A. For example, an i/o-constraint can be used to define (or approximate) how values of output ports can be derived from values of input ports. Such a (partial) specification of an actor can be used to propagate hybridization constraints themselves through one or more actors. Assume that $p_1 \in \mathsf{in}(A)$ has the structural type $\mathtt{s}_1 = \mathtt{r}(\mathtt{site}, \mathtt{day}, \mathtt{spp}, \mathtt{occ})$ from above, and $p_2 \in \mathsf{out}(A)$ has a structural type $\mathtt{s}_2 = \mathtt{r}'(\mathtt{sp}, \mathtt{oc})$,[10] and that the following i/o-constraint φ_{io} is given:

$$\forall x_{\mathtt{site}}, x_{\mathtt{day}}, x_{\mathtt{spp}}, x_{\mathtt{occ}} : \mathtt{r}(x_{\mathtt{site}}, x_{\mathtt{day}}, x_{\mathtt{spp}}, x_{\mathtt{occ}}) \longrightarrow \mathtt{r}'(x_{\mathtt{spp}}, x_{\mathtt{occ}}) \qquad (\varphi_{io})$$

Using the i/o-constraint φ_{io}, we can now propagate the above constraint α_1 "through" the actor A by applying φ_{io}. We are currently exploring reasoning procedures for propagation that handle a variety of i/o-constraint operations including aggregration, union, and group-by constructs. In this simple example, by applying the propagation procedure, we would obtain a (hybridization) constraint α_2 for the output port p_2 of A:

$$\begin{aligned}&\forall x_{\mathtt{sp}}, x_{\mathtt{oc}}\, \exists y : \mathtt{r}'(x_{\mathtt{sp}}, x_{\mathtt{oc}}) \longrightarrow \\ &\qquad \textsc{Measurement}(y) \wedge \textsc{itemMeasured}(y, x_{\mathtt{oc}}) \wedge \qquad (\alpha_2)\\ &\qquad \textsc{SpeciesOccurrence}(x_{\mathtt{oc}})\end{aligned}$$

Summary. Given the various extensions described above, we can now define a *typed workflow* $W = \langle \mathbf{A}, \mathbf{D}, \Sigma, M, \Phi \rangle$ to also include a set of *constraints* Φ. More precisely, $\Phi = \langle \Phi_{\mathcal{S}}, \Phi_{\mathcal{O}}, \Phi_{\mathcal{H}}, \Phi_{io} \rangle$ consists of a set $\Phi_{\mathcal{S}}$ associating *structural types* from $\mathcal{S}$ to ports in W, $\Phi_{\mathcal{O}}$ associating *semantic types* from an ontology $\mathcal{O}$ to actors and ports, $\Phi_{\mathcal{H}}$ linking structural and semantic types of ports, and finally Φ_{io}, specifying i/o-constraints of actors.

4 Design and Implementation of Scientific Workflows

This section presents a collection of design primitives to support workflow engineering (workflow conceptual design to implementation). Each primitive corresponds to a basic operation over the formal model for actor-oriented scientific workflows. Primitives are described as transformations that return the result of applying an operation to a workflow. Workflow engineers can repeatedly apply these primitives, e.g., via the KEPLER graphical user interface, to create their desired scientific workflow (see Figure 2).

Based on the primitives, we identify design strategies to help guide workflow engineers as they develop scientific workflows (see Figure 2). Each strategy emphasizes certain primitives within a larger design process. For example, a particular design method may be divided into a set of phases, and each phase may be guided by a certain strategy.

In this section, we also outline an approach to help automate the implemention of workflow designs. Our approach leverages hybrid typing to refine a workflow into an implemented version by repeatedly applying specific design primitives.

[9] For an early version of our approach, see [4].

[10] The structural types $\mathtt{s}_1$ and $\mathtt{s}_2$ are disconnected (unless an i/o-constraint is given), so one cannot assume the values (or types) of the input match the values (or types) of the ouput.

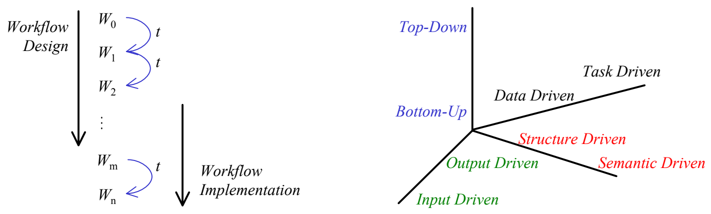


Fig. 2. Workflow engineers evolve workflows by applying design primitives (left), shown as transformations t; and primitives are grouped to form design strategies (right)

4.1 Scientific Workflow Design Primitives

Basic Actor-Oriented Design Primitives. Figure 3 summarizes the basic actor-oriented modeling primitives. In particular, we include primitives to: introduce new actors and dataflow connections into workflows (transformation t_1); add input and output ports to actors (transformation t_2); refine port structural types (transformation t_3); group (abstract) a portion of a workflow into a composite actor (transformation t_4); define an actor as a composite (transformation t_5); create dataflow connections (transformation t_6); and assign a director to a workflow (transformation t_7). For structural datatype refinement (transformation t_3), we require the "refined" datatype to be a subtype of the existing structural type. Although not shown in Figure 3, we also assume a transformation that "generalizes" structural types (structural type *abstraction*) requiring introduction of appropriate structural supertypes.

Semantic Typing Primitives. Figure 4 summarizes the semantic (hybrid) typing primitives. The first two transformations t_8 and t_9 refine actor semantic types and input and

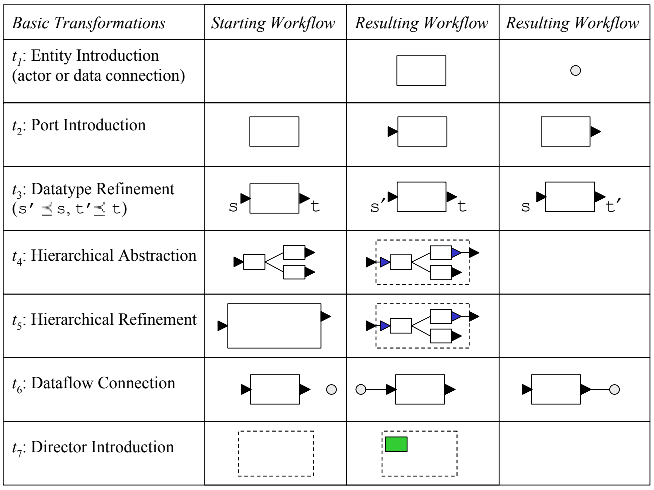

Basic Transformations
Starting Workflow
Resulting Workflow
Resulting Workflow
t_1: Entity Introduction (actor or data connection)
t_2: Port Introduction
t_3: Datatype Refinement ($s' \preceq s, t' \preceq t$)
s
t
s'
t
s
t'
t_4: Hierarchical Abstraction
t_5: Hierarchical Refinement
t_6: Dataflow Connection
t_7: Director Introduction

Fig. 3. Actor-oriented design primitives summarized as transformations where actors are represented as solid boxes; ports as triangles; dataflow connections as circles; composite actors as dashed boxes; and directors as solid (green) boxes

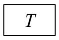
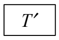
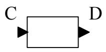
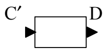
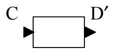
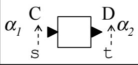
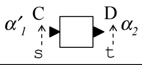
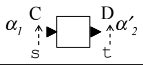
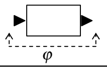
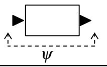
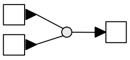
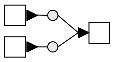
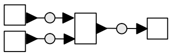
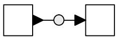
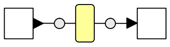
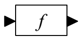
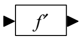
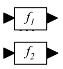
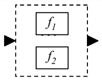
Extended Transformations	*Starting Workflow*	*Resulting Workflow*	*Resulting Workflow*
t_8: Actor Semantic Type Refinement ($T' \sqsubseteq T$)			
t_9: Port Semantic Type Refinement ($C' \sqsubseteq C$, $D' \sqsubseteq D$)			
t_{10}: Annotation Constraint Refinement ($\alpha' \rightarrow \alpha$)			
t_{11}: I/O Constraint Strengthening ($\psi \rightarrow \varphi$)			
t_{12}: Dataflow Connection Refinement			
t_{13}: Adapter Insertion			
t_{14}: Actor Replacement			
t_{15}: Workflow Combination (Map)			

Fig. 4. Additional primitives to support scientific-workflow design and implementation, where adapters are shown as solid, rounded boxes

output port semantic types, respectively. Semantic-type refinement requires the introduction of subconcepts, i.e., to refine an actor semantic type T to T′, the constraint T′ $\sqsubseteq$ T must hold. Refining the semantic types of an actor results in specializing the actor's operation. For instance, by refining an input-port semantic type, we further limit the kinds of objects an actor can process. And similarly, by refining an output-port semantic type, we further limit the kinds of objects that can be produced by an actor.

Often, actor and port semantic type refinements are performed together. For example, consider the following series of refinements (each consisting of individual actor and port semantic type refinements):

1. DATAMATRIX → [ANALYSIS] → RESULTSET
2. PHYLOGENETICMATRIX → [PHYLOGENETICANALYSIS] → PHYLOGENETICTREE
3. NEXUSMATRIX → [CLADISTICANALYSIS] → CONSENSUSTREE

The first refinement states that the semantic type of an actor is ANALYSIS, consisting of an input port of semantic type DATAMATRIX and output port of semantic type RESULTSET. Here, ANALYSIS, DATAMATRIX, and RESULTSET represent general concepts. The second refinement provides more details concerning the actor semantic type, which also influences the input and output port semantic types. The third refinement provides semantic types specific to a particular implementation of an analysis, again influencing the input and output port semantic types.

Primitives t_{10} and t_{11} are used to refine hybridization constraints and i/o-constraints, respectively. Like with semantic types, both hybridization constraint refinement and i/o-constraint strengthening specialize existing hybridization constraints and i/o-constraints (shown as the implications $\alpha' \rightarrow \alpha$ and $\psi \rightarrow \varphi$ in Figure 4).

Similar to the structural type refinement operation, each semantic type refinement operation is assumed to have a corresponding version for abstraction (i.e., generalization of types).

Extended Primitives for Dataflow Connections. It is often convenient to "loosely" connect actors through dataflow connections and then give the details of the connection later as the workflow becomes more complete. The dataflow-connection refinement (transformation t_{12}) provides two approaches for specifying the details of such a connection. The first (shown as the first resulting workflow for the refinment in Figure 4) splits a dataflow-connection node d into two separate dataflow-connection nodes d_1 and d_2 such that:

$$\mathsf{merge}(d_1) \cup \mathsf{merge}(d_2) \equiv \mathsf{merge}(d) \text{ and } \mathsf{distrib}(d_1) \cup \mathsf{distrib}(d_2) \equiv \mathsf{distrib}(d)$$

The second refinement transforms a dataflow-connection node d into an actor node A, which is constructed from d as follows: (1) each port p in $\mathsf{merge}(d)$ generates a new port p' that is added to $\mathsf{in}(A)$; (2) a new dataflow-connection node is created to connect the ports p and p'; (3) a new port p'' is created and added to $\mathsf{out}(A)$; and (4) $\mathsf{merge}(d)$ is assigned the singleton set $\{p''\}$.

Although not shown in Figure 4, we assume both versions of dataflow-connection refinement have corresponding generalization primitives.

Primitives for Adapter Insertion. The adapter insertion primitive (transformation t_{13}) is used to insert special actors called *adapters* between incompatible dataflow connections. We focus on adapters for situations in which a connection contains a semantic or structural incompatibility.

A *semantic adapter* is used to align input and output port connections that do not satisfy the subconcept typing constraint. We consider two cases for semantic adapter insertion. In the first case, an output port with semantic type C is connected to an input port with semantic type D. We assume that C and D are incompatible such that the constraint C $\sqsubseteq$ D does not hold. For example, let C and D be defined as follows.

$$\mathrm{C} \equiv \textsc{Measurement} \sqcap \forall \textsc{itemMeasured}.\textsc{SpeciesOccurrence}$$
$$\mathrm{D} \equiv \textsc{Measurement} \sqcap \forall \textsc{itemMeasured}.\textsc{SpeciesRichness}$$

The first actor produces data containing species' occurrence measurements and the second actor consumes data containing species' richness measurements. The semantic types are not compatible because SPECIESOCCURRENCE is not a subconcept of SPECIESRICHNESS. In general, however, richness data can be obtained from occurrence data through a simple conversion, namely, by summing occurrrence.

In this case, one may choose to insert a semantic adapter between the two actors. Conceptually, the adapter provides a data conversion that can reconcile the semantic differences between the two actors. Typically the input and output semantic types of a semantic adapter will be assigned the corresponding actor output and input, respectively. A semantic adapter can also have a more general input semantic type (e.g., a semantic type C$'$ $\sqsupseteq$ C) and a more restrictive output semantic type (e.g., D$'$ $\sqsubseteq$ D).

A *structural adapter* is similar to a semantic adapter, but is used to reconcile incompatible structural types found in data connections (as opposed to incompatible semantic types). Within KEPLER, users can determine whether connections are created that are

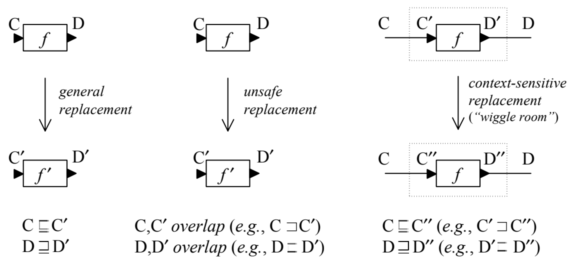


Fig. 5. Semantic type constraints for general, unsafe, and context-sensitive replacement

semantically or structurally incompatible. Incompatible types can be fixed by: (1) inserting an appropriate adapter; (2) modifying the data connection; or (3) abstracting and/or refining the problem types.

Primitives for Actor Replacement. The actor replacement primitive (transition t_{14}) is used to "swap" one actor in a workflow with another actor. We use standard object-oriented inheritance rules [6] to determine when a particular actor replacement is appropriate. Figure 5 shows three simple cases: the general case of safe replacement (shown on the left), unsafe replacement (shown in the middle), and context-sensitive replacement (shown on the right). For general replacement, an actor A_1 can be replaced by another actor A_2 if the following conditions hold: [11]

1. A_2 has an input (output) port for each of A_1's input (output) ports[12];
2. A_2's actor semantic type is a subconcept of A_1's actor semantic type;
3. A_2's input port types are equivalent or more general than A_1's; and
4. A_2's output port types are equivalent or more specific than A_1's.

As shown in Figure 5, *unsafe replacement* occurs when the semantic (or structural) port types do not satisfy the above conditions. However, unsafe replacement may still be considered appropriate when the replacement is taken in context. That is, the general form of unsafe replacement (the middle case of Figure 5) may become safe when the surrounding data connections are considered. We call this case *context-sensitive replacement*, as shown in Figure 5, the input and output semantic (and structural) replacement rules are determined by the semantic (and structural) types of corresponding data connections.

Primitives for Combining Workflows. The workflow combination primitive (transformation t_{15}) is used to assemble two or more workflows into a single "conglomerate." To be combined, the input and output structural and semantic types of the separate workflows must be combatible. The most specific input types of the separate workflows are used as the combined-workflow input types; and the most general output types of the separate workflows are used as the combined-workflow output types. Combining

[11] Note that in general we also require the i/o-constraint f' of the replacement to imply the i/o-constraint f of the original actor (i.e., $f' \rightarrow f$).

[12] Here, A_2 may contain more output ports than A_1, and possibly more input ports so long as the "extra" ports are not required. As future work, we are also more generally considering matching aggregrations of ports.

similar workflows is useful for cases where multiple algorithms exist to perform a similar function, e.g., to perform multiple multivariate statistics over the same input data.

The workflow combination primitive is similar to the higher order function `map :: [a] -> (a -> b) -> [b]`, which returns the result of applying a function to each element of a list. In particular, the workflow combination primitive can be viewed as a variant `Map :: a -> [(a -> b)] -> [b]` that takes a value v and a list of functions $f_1, f_2, ..., f_n$, and returns a list containing the values $f_1(v), f_2(v), ..., f_n(v)$.

4.2 Strategies for Workflow Design

As shown in Figure 2 (and similar in spirit to [3]), we define high-level design strategies that emphasize specific transformation primitives. The strategies can be used to describe design methods where at each stage, a particular strategy (a point in the design space of Figure 2) is applied. The design strategies are defined as follows.

- *Task-Driven Design*: Workflow engineers focus on identifying the conceptual actors of a workflow. This strategy can involve defining actor ports, semantic types, structural types, associations, and i/o-constraints along with hierarchcial refinements and replacements to convert abstract actors to implemented versions.
- *Data-Driven Design*: Wofkflow engineers focus on identifying the input data and dataflow connections of workflows. Dataflow connections may be elaborated using refinement.
- *Semantic-Driven Design*: Workflow engineers focus on specifying the semantic types of the workflow. The engineer may start with a "blank" workflow topology containing basic actors and dataflow connections, and identify the appropriate semantic types, adding concepts and roles to ontologies as needed.
- *Structure-Driven Design*: Like semantic-driven design, but for structural types.
- *Input-Driven Design*: Workflow engineers focus on identifying the input of a workflow, and design from "left to right," i.e., from the input side to the output side of the workflow.
- *Output-Driven Design*: Like input-driven design, but focus on data products first.
- *Top-Down Design*: Workflow engineers focus on refining actors and dataflow connections. The engineer may begin with a single empty workflow and iteratively apply hierarchical and dataflow connection refinement.
- *Bottom-Up Design*: Workflow engineers focus on abstraction of actors and dataflow connections. The engineer may first define specific parts of a workflow and iteratively abstract the workflow using hierarchical abstraction to connect the various parts.

Different workflow design methods apply in different situations. We have found that the process of re-engineering existing applications into workflows often starts with top-down, structure driven strategies. But, when scientists develop new workflows (e.g., new analyses as opposed to "re-engineered" ones), a mix of semantic, input, and output strategies are used.

4.3 From Design to Implementation of Scientific Workflows

Here we outline an approach that leverages hybrid typing, replacement rules, and adapter insertion to help automate the task of finding appropriate actor implementations for workflow specifications. We assume there is a repository $\mathbf{R}$ of semantically typed actors and workflows. We use the term *abstract actor* to refer to actors that cannot be executed (i.e., without implementations) and *concrete actor* to refer to executable actors. $\mathbf{R}$ may consist of abstract or concrete actors, composite actors, and entire workflows. The following steps sketch the approach for finding implementations of a workflow W:

1. if W is a concrete workflow, output W
2. select an abstract actor $A_T \in \mathbf{A}$ that has an actor replacement $A_C \in \mathbf{R}$
3. let W' be the workflow that results from replacing A_T by A_C
4. if W' has an incompatible dataflow connection, insert an abstract adapter
5. repeat with $W := W'$

The basic idea of the approach is to define a search space such that each node represents a workflow and transitions between nodes are defined using steps 2-4 above. The procedure for finding implementations of W is to navigate the search space (e.g., using a breadth-first or depth-first search algorithm) looking for nodes that represent concrete workflows. In the transitions (steps 2-4) defined above, we replace individual abstract actors in a workflow with valid replacements from the respository. When a concrete actor is inserted that violates a semantic or structural typing rule, we also insert an abstract adapter actor, which can also be replaced (in subsequent steps). In general, for a given worfklow W there may be many associated concrete workflows, depending on whenever an abstract actor can be replaced by more than one repository element. The user may wish to combine some or all of the resulting workflows using the workflow combination primitive.

5 Summary

This paper extends our previous work by describing a formal model of scientific workflows based on actor-oriented modeling and design. The approach facilitates conceptual modeling of scientific workflows through a novel hybrid type system, and by providing a set of primitive modeling operations for end-to-end scientific workflow development. Our approach can also support the conceptual and structural validation of scientific workflows, as well as the discovery of type-conforming workflow implementations via replacement rules and by inserting appropriate semantic and structural adapters for workflow integration. Much of this work is currently implemented within the KEPLER system, and we are currently extending KEPLER with semantic propagation and additional reasoning techniques to further exploit hybrid types.

References

1. A. Ailamaki, Y. E. Ioannidis, and M. Livny. Scientific Workflow Management by Database Management. *In Proc. of SSDBM*, pages 190199, 1998.
2. G. Alonso and C. Mohan. Workflow Management Systems: *The Next Generation of Distributed Processing Tools. In Advanced Transaction Models and Architectures*. 1997.
3. C. Batini, S. Ceri, and S. Navathe. *Conceptual Database Design: An Entity-Relationship Approach*. Benjamin/Cummings, 1992.

4. S. Bowers and B. Ludascher. An Ontology-Driven Framework for Data Transformation in ScientificWorkflows. *In Proc. of the Intl. Workshop on Data Integration in the Life Sciences (DILS), volume 2994 of LNCS*, pages 116. Springer, 2004.
5. F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Conceptual Modelling ofWorkFlows. In *Object-Oriented and Entity-Relationship Modelling Conference (OOER)*, volume 1021 of *LNCS*, pages 341354. Springer, 1995.
6. G. Castagna. Covariance and contravariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 17(3), 1995.
7. K. W. B. H. Wright and M. J. Brown. The Dataflow Visualization Pipeline as a Problem Solving Environment. In *Virtual Environments and Scientific Visualization*. Springer, 1996.
8. B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. Ph.D. Thesis, Queensland University of Technology, 2002.
9. E. A. Lee and S. Neuendorffer. Actor-oriented Models for Codesign: Balancing Re-Use and Performance. In *Formal Methods and Models for Systems*. Kluwer, 2004.
10. E. A. Lee and T. M. Parks. Dataflow process networks. *Proc. of the IEEE*, 83(5):773801, 1995.
11. B. Ludascher, I. Altintas, D. H. Chad Berkley, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows*, 2005. to appear.
12. S. Majithia, M. S. Shields, I. J. Taylor, and I. Wang. Triana: A Graphical Web Service Composition and Execution Toolkit. In *Proc. of the IEEE Intl. Conf. onWeb Services (ICWS)*. IEEE Computer Society, 2004.
13. J. Meidanis, G. Vossen, and M. Weske. Using Workflow Management in DNA Sequencing. In *Proc. of CoopIS*, pages 114123, 1996.
14. J. P. Morrison. *Flow-Based Programming: A New Approach to Application Development*. Van Nostrand Reinhold, 1994.
15. T. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):30453054, 2004.
16. W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems (Cooperative Information Systems)*. MIT Press, 2002.
17. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):551, 2003.
18. M. zur Muehlen. *Workflow-based Process Controlling: Foundation, Design, and Application of workflow-driven Process Information Systems*. Logos Verlag, Berlin, 2004.

Blueprints and Measures for ETL Workflows

Panos Vassiliadis[1], Alkis Simitsis[2], Manolis Terrovitis[2], and Spiros Skiadopoulos[2]

[1] University of Ioannina, Dept. of Computer Science, Ioannina, Hellas
pvassil@cs.uoi.gr
[2] National Technical University of Athens, Dept. of Electrical and Computer Eng., Athens, Hellas
{asimi, mter, spiros}@dbnet.ece.ntua.gr

Abstract. Extract-Transform-Load (ETL) workflows are data centric workflows responsible for transferring, cleaning, and loading data from their respective sources to the warehouse. Previous research has identified graph-based techniques that construct the *blueprints* for the structure of such workflows. In this paper, we extend existing results by explicitly incorporating the internal semantics of each activity in the workflow graph. Apart from the value that blueprints have per se, we exploit our modeling to introduce rigorous techniques for the measurement of ETL workflows. To this end, we build upon an existing formal framework for software quality metrics and formally prove how our quality measures fit within this framework.

1 Introduction

All engineering disciplines employ blueprints during the design of their engineering artifacts. Modeling in this fashion is not a task with a value by itself; as [1] mentions "we build models to communicate the desired structure and behavior of our system … to visualize and control the system's architecture … to better understand the system we are building … to manage risk".

In this paper, we discuss the constructing entities and the usage of blueprints for a particular category of database-centric software, namely, the Extract-Transform-Load (ETL) workflows. ETL workflows are an integral part of the back-stage of data warehouse architectures, where data are (a) collected from the operational sources, (b) cleansed (to remove any noise or inconsistencies), (c) transformed (so that they syntactically comply with the schema of the warehouse tables) and finally, (d) loaded to the target warehouse tables. Out of the aforementioned benefits of modeling, control of the system's architecture and risk management are of particular importance. For example, we would like to answer questions like:

- Which attributes/tables are involved in the population of a certain attribute?
- What part of the scenario is affected if we delete an attribute?
- How good is the design of my ETL scenario? Is variant A or variant B better?

Previous research has provided some results towards the aforementioned tasks. The work of [9, 11] provides conceptual modeling techniques for ETL. In [10] we have presented a first attempt towards a graph-based model for the definition of the ETL scenarios. The model of [10] treats ETL scenarios as graphs. Activities and data stores are modeled as the nodes of the graph; the attributes that constitute them are modeled as nodes too. Activities have input and output schemata and *provider relationships* relate inputs and outputs between data providers and data consumers. Nevertheless, what is missing from previous efforts is a full model of the semantics of ETL workflows and a rigorous framework for the measurement of our design artifacts.

In this paper, we significantly extend previous works to capture the internals of the workflow activities in sufficient detail. We make use of a logical abstraction of ETL activity semantics in the form of LDL++ programs [14]. The approach is not unrealistic: in fact, in [12] the authors discuss the possibility of providing extensible libraries of ETL tasks, logically described in LDL. On the basis of this result, it is reasonable to assume the reusability of these libraries. In this paper, we extend the graph of [12] by incorporating the internals of the activity semantics to the graph. To this end, we provide a principled way of transforming LDL programs to graphs, in a way that gracefully complements the model of [10, 12]. The resulting graph, which is called *Architecture Graph* can provide sufficient answers to what-if and dependency analysis in the process of understanding or managing the risk of the environment.

Moreover, another question can also be answered: "How good is my design?". The community of software engineering has provided numerous metrics towards evaluating the quality of software designs [4]. Are these metrics sufficient? In this paper, we build upon the fundamental contribution of [2] that develops a rigorous and systematic framework that classifies usually encountered metrics into five families, each with its own characteristics. These five families are *size*, *length*, *complexity*, *cohesion* and *coupling* of software artifacts. In this paper, we develop specific measures for the Architecture Graph and formally prove their fitness for the rigorous framework of [2].

In a nutshell, our contributions can be listed as follows:

- an extension of [10] to incorporate internal semantics of activities in the architecture graph;
- a principled way of transforming LDL programs to the graph, so that the latter can be explored both at the granular (i.e., attribute) level of detail and at different levels of abstraction;
- a systematic definition of software measures for the Architecture Graph, based on the rigorous framework of [2].

This paper is organized as follows. In Section 2, we present the graph model for ETL activities. Section 3 discusses measures for the introduced model. In Section 4, we present related work. Finally, in Section 5 we conclude our results and provide insights for future work. The reader is encouraged to refer to the long version of this paper [13] for all the proofs and several technical issues, omitted from this paper for lack of space.

2 A Generic Model of ETL Activities

The purpose of this section is to present a formal logical model for the activities of an ETL environment. We start with the background constructs of the model, already introduced in [10, 12]. Then, we move on to extend this modeling with formal semantics of the internals of the activities.

In order to formally define the semantics of ETL workflow, we can use any 3GL/4GL programming language (C++, PL/SQL etc.). We do not consider the actual implementation of the workflow in some programming language, but rather, we employ LDL++ [14] in order to describe its semantics in a declarative nature and understandable way. LDL++ is a logic-programming, declarative language that supports recursion, complex objects and negation. Moreover, LDL++ supports external functions, choice, (user-defined) aggregation and updates. LDL was carefully chosen as the language for expressing ETL semantics. First, it is elegant and has a simple model for expressing activity semantics. Second, the *head-body* combination is particularly suitable for relating both (a) input and output in the simple case, and, (b) consecutive layers of intermediate schemata in complex cases. Finally, LDL is both generic and powerful, so that (large parts of) other languages can be reduced to the Architecture Graph constructs that result from it.

2.1 Preliminaries

In this subsection, we introduce the modeling constructs of [10, 12] upon which we will subsequently build our contribution. In brief, the basic components of this modeling framework are:

- *Data types*. Each data type `T` is characterized by a name and a domain, i.e., a countable set of values. The values of the domains are also referred to as *constants*.
- *Attributes*. Attributes are characterized by their name and data type. For single-valued attributes, the domain of an attribute is a subset of the domain of its data type, whereas for set-valued, their domain is a subset of the powerset of the domain of their data type $2^{\mathrm{dom(T)}}$.
- A *Schema* is a finite list of attributes. Each entity that is characterized by one or more schemata will be called *Structured Entity*.
- *Records & RecordSets*. We define a *record* as the instantiation of a schema to a list of values belonging to the domains of the respective schema attributes. Formally, a recordset is characterized by its name, its (logical) schema and its (physical) extension (i.e., a finite set of records under the recordset schema). In the rest of this paper, we will mainly deal with the two most popular types of recordsets, namely *relational tables* and *record files*.
- *Functions*. A *Function Type* comprises a name, a finite list of *parameter data types*, and a single *return data type*.
- *Elementary Activities*. In the framework of [12], activities are logical abstractions representing parts, or full modules of code. An *Elementary Activity* (simply referred to as *Activity* from now on) is formally described by the following elements:
    - *Name*: a unique identifier for the activity.
    - *Input Schemata*: a finite list of one or more input schemata that receive data from the data providers of the activity.
    - *Output Schemata*: a finite list of one or more output schemata that describe the placeholders for the rows that pass the checks and transformations performed by the elementary activity.

  - *Operational Semantics*: a program, in LDL++, describing the content passing from the input schemata towards the output schemata. For example, the operational semantics can describe the content that the activity reads from a data provider through an input schema, the operation performed on these rows before they arrive to an output schema and an implicit mapping between the attributes of the input schema(ta) and the respective attributes of the output schema(ta).
  - *Execution priority*. In the context of a scenario, an activity instance must have a priority of execution, determining when the activity will be initiated.
- *Provider* relationships. These are `1:N` relationships that involve attributes with a provider-consumer relationship. The flow of data from the data sources towards the data warehouse is performed through the composition of activities in a larger scenario. In this context, the input for an activity can be either a persistent data store, or another activity. Provider relationships capture the mapping between the attributes of the schemata of the involved entities. Note that a consumer attribute can also be populated by a constant, in certain cases.
- *Part_of* relationships. These relationships involve attributes and parameters and relate them to their respective activity, recordset or function to which they belong.

Based upon the previous constructs, already available from [12], we proceed with their extension towards fully incorporating the semantics of ETL workflow in our framework. To this end we introduce *programs* as another modeling construct.

- *Programs*. We assume that the semantics of each activity is given by a declarative program expressed in LDL++. Each program is a finite list of LDL++ rules. Each rule is identified by an (internal) rule identifier. We assume a normal form for the LDL++ rules that we employ. In our setting, there are three types of programs, and normal forms, respectively:
  (i) *intra-activity* programs that characterize the operational semantics, i.e., the internals of activities (e.g., a program that declares that the activity reads data from the input schema, checks for NULL values and populates the output schema only with records having non-NULL values),
  (ii) *inter-activity* programs that link the input/output of an activity to a data provider/consumer,
  (iii) *side-effect* programs that characterize whether the provision of data is an insert, update, or delete action. Due to lack of space, we discuss side-effect rules in detail in the long version of this paper [13].
- *Regulator Relationships.* A regulator relationship in a safe rule is an equality/inequality relationship between two terms, i.e., of the form $term_1$ θ $term_2$, such that neither of them appears in the head of a rule. In terms of the architecture graph, the regulator relationship is represented (a) by a node for each of the terms, (b) by a node representing the condition θ, and (c) by two edges among the node of the condition and the nodes of the term. The direction of the edges follows the way the expression is written in LDL (i.e., from the left to right). Regulator relationships are used in order to capture the selection conditions and joins that take place in an LDL program.

We assume that each activity is defined in isolation. In other words, the inter-activity program for each activity is a stand-alone program, assuming the input schemata of the activity as its EDB predicates. Then, activities are plugged in the overall scenario that consists of inter-activity and side-effect rules and an overall *scenario program* can be obtained from this combination.

Intra-activity programs. The intra-activity programs abide by the following rules:

1. All input schemata are EDB predicates.
2. All output schemata appear only as IDB predicates. Furthermore, output schemata are the only IDB predicates that appear in such a program.
3. Intermediate rules are possibly employed to help with intermediate results.
4. We assume non-recursive admissible programs. The safety of the program is guarantied by the requirement for *admissibility*, which is a generalization of stratifiability [3]. An admissible program does not contain any self-referential *set definitions* or any predicates defined in terms of their own *negations*.

Inter-activity programs. The inter-activity programs are very simple. There is exactly one rule per provider relationship, with the consumer in the head and the provider in the body. The consumer attributes are mapped to their corresponding providers either through the synonym mechanism or through explicit equalities. No other atoms or predicates are allowed in the body of an inter-activity program; all the consumer attributes should be populated from the provider.

$$\texttt{Consumer\_input}(a_1,\dots,a_n) \texttt{ <- } \texttt{provider\_output}(a_1,\dots,a_m),\ m \geq n$$

2.2 Incorporating Activity Semantics in the Architecture Graph

The focus of [10, 12] is on the input-output role of the activities instead of their internal operation. In this section, we extend the model of those works by translating the formal semantics of the internals of the activities to graph constructs, as part of the overall Architecture Graph. We organize this discussion as follows: first, we consider how individual rules are represented by graphs for intra-activity and inter-activity programs. The interested author can find a discussion about side-effect programs in the long version of this paper [13]. Then, we discuss how the programs of activities are constructed from the composition of different rules and finally, we discuss how a scenario program can be obtained from the composition of the graph representations of individual programs.

Intuitively, instead of simply stating which schema populates another schema, we trace how this is done through the internals of an activity. The programs that facilitate the input to output mappings take part in the graph, too. Any filters, joins or aggregations are part of the graph as first-class citizens. There is a straightforward way to determine the architecture graph with respect to the LDL program that defines the ETL scenario.

Intra-activity rules. Given the program of the activity as a stand-alone LDL++ program, we introduce the following constructs, by default:

– A node for the activity per se.
– A node for each of the schemata of the activity and a node for the activity program. Part-of edges connect the activity with these components.
– A node for each rule, connected through a part-of relationship to the program node of the activity.

If we treat each rule as a stand-alone program, we can construct its graph as follows:

– We introduce a node for each predicate of the rule. These nodes are connected to the rule node through a part-of relationship. The edge of the head predicate is tagged as 'head' and the edges of the negated literals of the body are tagged as '¬'. Functions are treated as predicates. A different predicate node is introduced for each instance of the same predicate (e.g., in the case of a self-join). Such nodes are connected to each other through *alias* edges. In the long version [13], we detail the parts of the last cases that require extra attention.
– We introduce a node for each variable of a predicate. Part-of relationships connect these nodes with their corresponding predicates.
– For each condition of the form *Input attribute* = *Output attribute* (or its equivalent presence of *synonyms* in the output and input schemata), we add a provider edge. Here, we assume as input (output) attributes, attributes belonging to predicates of the rule body (head). A provider relationship is thus, an edge from the body towards the head of the rule.
– For relationships among input attributes (practically, involving functions and built-ins), a regulator edge is introduced.

```
R06: sk.a_in1(pkey,suppkey,date,qty,cost)<-
        dsa_ps(pkey,suppkey,date,qty,cost).

R07: sk.a_in2(pkey,source,skey)<-
       lookUp(l_pkey,source,l_skey),pkey=l_pkey,skey=l_skey,
       source=1.
R08: sk.a_out(pkey,suppkey,date,qty,cost,skey)<-
       add_sk1.a_in1(pkey,date,qty,cost),
       add_sk1.a_in2(pkey,source,l_skey).

R09: dollar2euro.a_in(skey,suppkey,date,qty,cost)<-
       sk.a_out(pkey,suppkey,date,qty,cost,skey).
```

Fig. 1. LDL++ for a small part of a scenario

Inter-activity rules. For each recordset of a scenario, we assume a node representing its schema. For simplicity, we do not discriminate a recordset from its schema using different nodes. For each intra-activity rule (between input-output schemata of different activities and/or recordsets) there is a simple way to construct its corresponding graph: we introduce a provider edge from the input towards the output attributes.

Observe Fig. 1. Activity `SK` (Surrogate Key assignment) takes as input the data from a recordset `DSA_PS(`<u>`PKey`</u>`,`<u>`SuppKey`</u>`,`<u>`Date`</u>`,Qty,Cost)`, and obtains a globally unique surrogate key `SKey` for the production key `PKey`, through a lookup table `LookUp(`<u>`PKey`</u>, <u>`Source`</u>, `SKey)`; in this example, we consider that data originate from source `1`. Then, the transformed data are propagated to another activity `dollar2euro` that converts the dollar values of attribute `cost`, to Euros (only the input schema of this activity is depicted). The rules `R06`, `R07` and `R09` are inter-activity rules: they describe how the input schemata of the activities are populated from their providers.

Activities and recordsets can both play the role of provider, as one can see. Rule R08 is an intra-activity rule. Fig. 2 depicts the architecture graph for this example. The grey area concerns the intra-activity program; the rest concern the inter-activity program rules. Solid arrows depict provider relationships, dotted arrows depict regulator relationships and part-of relationships are depicted with simple lines.

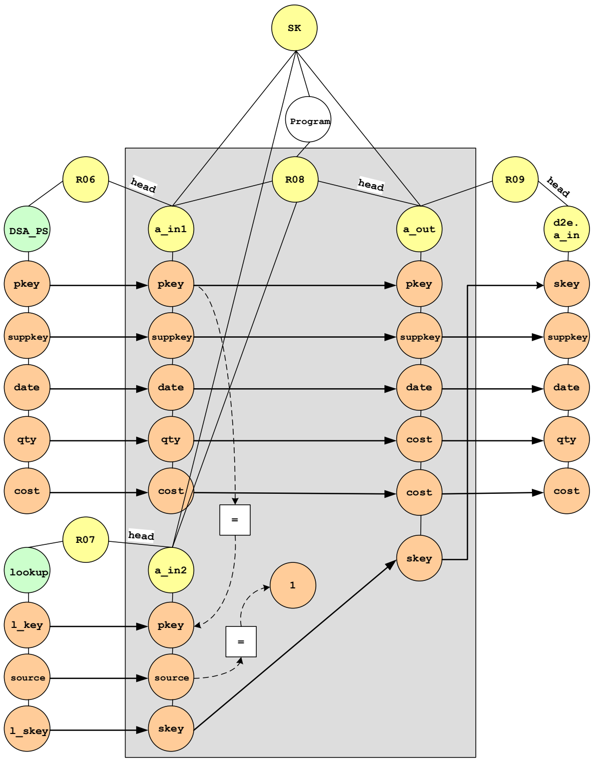


Fig. 2. Architecture graph for the example of Fig. 1

Deriving the graph of an activity program from the graphs of its rules. Combining the graphs of the rules of an activity is straightforward. Recall that so far, we have created the graph for each rule, considering each rule in isolation. Then, the graph for the overall activity is as follows:

- The nodes of the new graph comprise all the nodes of the rule graphs. If the same predicate appears to more than one rule, we merge all its corresponding nodes (i.e., the predicate node and all its variables). In the case where more than one instance of the same predicate exists in one rule, we randomly select one of these occurrences to be merged with the nodes of other rules.
- The edges of the new graph are all the edges, of the individual rules, after the merging takes place.

- All edges are tagged with the rule identifier of the rule they belong to. Through part of relationships and edge tagging, we can reconstruct the graphs of the individual rules, if necessary.

Deriving the graph of a scenario program from the graphs of its components. The construction of the graph for the scenario program is simple.

- First, we introduce all inter-activity and side-effect rules. We merge all multiple instances of the same recordset and its attributes. The same applies with the input and output schemata of an activity. We annotate all edges with the rule identifier of their corresponding rule.
- Then, intra-activity graphs are introduced too. Activity input and output schemata are merged with the nodes that already represent them in the combined intra-activity/side-effect graph. The same applies to activity nodes, too. No other action needs to be taken, since intra-activity programs are connected to the rest of the workflow only through their input and output schemata.

Once again, the reader is encouraged to refer to the long version of this paper [13], where we handle several issues omitted here due to lack of space. Specifically, these issues involve updates, aggregation, negation, functions and self-join queries. Moreover, the possibility of zooming in/out the Architecture Graph is also provided in [13]. The latter is a most useful interactive facility, necessary for avoiding the information overload due to the potentially high volume of detailed information at the attribute level, as described in this section.

3 Measuring the Architecture Graph: A Principled Approach

One of the main roles of blueprints is their usage as testbeds for the evaluation of the design of an engineer. In other words, blueprints serve as the modeling tool that provides answers to the questions "*How good is my design?*" or "*Between these two designs, which one is better?*". In other words, one can define metrics or, more generally, measurement tests, to evaluate the quality of a design. In this section, we will address this issue, for our ETL workflows, in a principled manner.

There is a huge amount of literature devoted in the evaluation of software artifacts. Fenton proves that it is impossible to derive a unique measure of software quality [6]. Rather, measurement theory should be employed in order to define meaningful measures of particular software attributes. A couple of years later, Briand et al., employ measurement theory to provide a set of five generic categories of measures for software artifacts [2]:

- *Size*, referring to the number of entities that constitute the software artifact.
- *Length*, referring to the longest path of relationships among these entities.
- *Complexity*, referring to the amount of inter-relationships of a component.
- *Cohesion*, measuring the extent to which each module performs exactly one job, by evaluating how closely related are its components.
- *Coupling*, capturing the amount of interrelationships between the different modules of a system.

Systems and their modules are considered to be graphs with the nodes representing their constituent entities and the edges representing different kinds of interrelationships among them (Fig. 3). The definition of these categories is generic, in the sense, that depending on the underlying context, one can define his own measures that fit within one of the aforementioned categories. In order to be able to claim fitness within one of the aforementioned categories, there is a specific list of properties that the proposed measure must fulfill. For example, the size of a system modeled as a graph `S(E,R)` is a function `Size(S)` that is characterized by the properties: (a) *nonnegativity*, i.e., $\mathtt{Size(S)}\geq 0$; (b) *null value*; $\mathtt{E}=\emptyset \Rightarrow \mathtt{Size(S)=0}$; and (c) *module additivity*, i.e., if a system `S` has two modules m_1 and m_2, then $\mathtt{Size(S)} = \mathtt{Size}(m_1) + \mathtt{Size}(m_2)$. The last property shows that adding elements to a system cannot decrease its size (*size monotonicity*). For instance, the size of the system in Fig. 3 is the sum of the sizes of its two modules m_1 and m_2. The intuition here is that if the size of a certain module is greater than the size of another, then we can safely argue that the former is comprised of more entities than the latter.

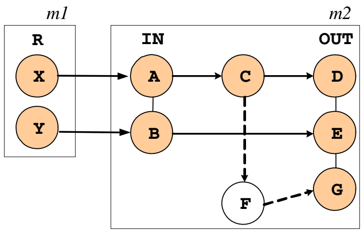


Fig. 3. A modular system graph containing two modules

Another important observation, found in both [6] and [2], is that measurement theory imperatively demands that a measure describes an intuitively clear concept, i.e., there is a clear interpretation of what we measure. This should be coupled with clear procedures for determining the parameters of the model and interpreting the results.

In this paper, we propose a set of measures that evaluate our ETL blueprints and stay within the context of the measures proposed in [2]. *Our fundamental concern, for defining our measures is the effort required (a) to define and (b) to maintain the Architecture Graph, in the presence of changes.* Therefore, the statements that one can make, concerning our measures characterize the effort/impact of these two phases of the software lifecycle.

First, we identify the correspondence of the constructs of the Architecture Graph to the concepts of [2]. In [2], a system `S` is a graph `S=(E,R)`, where `E` is the set of elements of the system and `R` is the set of relationships between the elements. A module `m` is a subset of the elements (i.e., the nodes) of the system (observe that a module is defined only in terms of nodes and not edges). In general, modules can overlap. However, when the modules partition the nodes in a system, then this system is called a modular system, `MS`. The authors distinguish two categories of edges: (a) the *intermodule* edges that have end points in different modules and (b) the *intramodule* edges that have end points in the same module. In terms of our modeling:

- The architecture graph `G(V,E)` is a modular *system*.
- Recordsets and activities are the *modules* of the graph. The nodes of the graph involve attributes, functions, constants, etc. All kinds of relationships are the *edges* of the graph.
- The system is indeed modular, i.e., there are no elements (nodes) that do not belong to exactly one module (activity or recordset).
- Inter-activity and side-effect rules result in intermodule edges. All the rest of the relationships result in intramodule edges.
- The union of two interacting activities can be defined: it requires merging the input/output nodes (attribute/schemata) connected by provider relationships.

3.1 Measures

Next, we define our measures. Due to lack of space, the proof of fitness within the set of properties of [2] for each measure can be found in [13]. We strongly encourage the reader to read the correctness proofs as they offer a deeper comprehension of the nature of the measures that we propose.

Size. Size is a measure of the amount of constituting elements of a system. Therefore, it can be considered as a reasonable indicator of the amount to define the system. In our framework, we adopt the *number of nodes* as the measuring rule for the size of the Architecture Graph; thus, the size of the architecture graph `G(V,E)` is given by the formula `Size(G) = card(V)`.

Length. Length is a measure that refers to the maximum length of "retransmission" of a certain attribute value. Length measures the longest path that we possibly need to maintain if we make an alteration in the structure of the Architecture Graph. For example, this could involve the deletion of an attribute at the source side. Then, the length characterizes how many nodes in the graph we need to modify as a result of this change (practically involving the nodes corresponding to this particular attribute, within the workflow).

In [10] the authors define the (transitive) dependency of a node as the cardinality of the (transitive closure of) provider relationships arriving at this node. To define the length of path from a module `m` backwards to the fountains of the graph, we use the maximum of its transitive dependency measure for the attributes of its output schemata. We avoid cycles in the graph by special treatment of side-effects [13]. Thereby, the *length of a module* `m` is given by the formula:

$$\mathrm{Length}(m)=\max\{\mathrm{transitive\_dependency}(i)\},\ i\in \mathrm{output\_schemata}(m).$$

The *length of the graph* is defined as the maximum length over all its modules `m`:

$$\mathrm{Length}(G)=\max(\mathrm{Length}(m_j))$$

Observe the reference example of Fig. 1. Although it does not depict a complete graph, the length of the depicted subgraph is 3, since the maximum length of its modules is 3 (input schema of activity `$2E`).

Complexity. Complexity is an inherent property of systems; in our case, complexity stands to the amount of interconnection of constituent entities of the Architecture Graph. This is an indicator of maintenance effort in the presence of changes. The more complex a system is the more amount of maintenance effort is expected to be required in the case of changes. Briand et al. [2] indicate that the properties of complexity focus on edges, thus, our function for complexity concerns the edges of the graph G(V,E) at the most detailed level. Again, we distinguish module from system complexity.

We define the *overall degree* of a module to be the overall number of edges of any kind (i.e., provider, part-of, etc) among its components, independently of direction. We count inter-module provider edges as half for each module. Then,

$$\text{Complexity(m)} = |E_{\text{intramodule}}| + 0.5 * |E_{\text{intermodule}}|$$

The complexity of the architecture graph G(V,E) is defined as the summary of the complexities of all the modules of the graph (i.e., recordsets and activities).

$$\text{Complexity(G)} = \text{overall\_degree(G)} = |E|$$

Cohesion. A commonly agreed upon property of modular software is that each module ideally performs exactly one job. Cohesion is the measure employed to assess the extent to which the modules of a system abide by this rule. In our case, we can exploit the peculiarities of our setting to assess the cohesion of our ETL workflows.

ETL operations can largely be classified in two categories. Each activity in our model performs one of two tasks: (a) filtering, meaning that a certain criterion is applied over the employed data in order to block those that do not pass the test and (b) transformation, meaning that a certain function is applied in order to generate some new value in the workflow. Both these tasks involve regulator relationships among the involved attributes and the functions/built-in selectors (=, ≤, etc.) of the activities. Therefore, the amount of regulator relationships should be a good indicator of the cohesion of a system. Moreover, we impose two extra requirements that we consider reasonable: (a) the more functions/built-ins employed, the less cohesive the module is (i.e., it is assumed/expected to perform more than one job) and (b) if more attributes are involved in regulator relationships, cohesion increases. In the sequel, we will refer to functions and built-ins as *functionality* nodes.

Before giving the formal definition, we will present the intuition of our proposed measure. Due to requirement (a), we need the inverse of the number of employed functionality nodes. Also due to the requirement (b) we need a measure analog to the number of attributes involved in a regulator relationship. Since Briand et al. [2] require cohesion to be normalized within a range [0…max], we need to normalize the number of attributes involved with the total number of attributes. To simplify things we measure only input and output attributes. Still, we count an input/output attribute as *functionality-related* even if it is not directly involved in a regulator relationship, but transitively dependent (or responsible) with an internal attribute that is involved.

In Fig. 3, we depict providers with solid lines and regulators with dotted lines. The input attribute A of module m_2 is involved in a regulator relationship transitively (through attribute C), whereas the attribute G is directly involved in a regulator relationship. Now, we are ready to define cohesion for our modules and system.

$$\text{Cohesion(m)} = \frac{\text{F\_IN+F\_OUT}}{\text{F*(IN+OUT)}},$$

where F is the number of functions of the module, IN (OUT) is the number of input (output) attributes of the module, and F_IN (F_OUT) is the number of functionally-related input (output) attributes of module m.

$$\text{Cohesion(G)} = \text{avg(cohesion}(m_i)), \text{ for all the modules } m_i \text{ of G}$$

Cohesion for the module m_2 of Fig. 3 takes the value of (1+1)/1*5=0.4.

Coupling. In our framework, coupling captures the amount of relationship between the attributes belonging to different recordsets or activities (i.e., modules) of the graph. Briand et al. [2] indicate that the properties of complexity focus on intermodule edges, thus our function for complexity concerns the provider edges of the graph G(V,E) that start from an output node of a module and terminate to an input node of another module. Thus, we define the coupling of a graph G(V,E) as the sum of incoming and outgoing provider edges of each activity or recordset. This summary of edges for a certain module is called *local degree* according to the terminology we introduced in [10]. Thus, coupling is given by:

$$\text{Coupling(G)} = \textstyle\sum_i \text{local\_degree}(m_i), \text{ for all the modules } m_i \text{ of G}$$

In the reference example of Section 2, the coupling of the activity SK is 13, i.e., the total number of its incoming and outgoing provider relationships.

3.2 Example

In order to demonstrate the usage of our proposed metrics, we present an exemplary scenario, implemented in three different ways. For each of these implementations we measure the different properties that we have proposed and discuss the observed phenomena.

The scenario involves the propagation of data from the product suppliers table DSA_PS(PKEY,SUPPKEY,DATE,QTY,COST) towards the table DW_V1(PKEY, SUM_COSTS) with the obvious semantics. Three operations need to take place between the two data stores: (a) a selection involving dates after 1/1/2004, (b) a second selection test involving only quantities greater than zero and (c) a summation of costs per product key. In the first scenario, we employ a different activity for each of the operations, with the activities connected serially. In the second scenario, we have merged the two filters in a single activity. In the third scenario, the selections are performed in parallel, the results are then joined and subsequently aggregated. The graph representation of the scenarios is partially depicted in Fig. 4, where the abstract representation of each scenario is shown in the upper part of each column and the part of the detailed representation is depicted in the lower part. We omit part-of relationships and details higher than the schema level for reasons of space and presentation. In the figure, we refer to attribute SUPPKEY as SUPP for lack of space. The metrics for each scenario are depicted in the tables 1- 3 and refer to the depicted graphs (with very small discrepancies from the overall graphs).

The observation of these measures reveals that the second scenario outperforms all the others in all categories. This is due to the fact that by merging the selections in a single activity, all provider relationships among modules are shortened. The same applies, of course, for the size of the graph. In terms of individual measures, we can observe the following:

- **Size** has obvious results, simply due to the number of attributes in the input and output schemata of the activities.
- The **length** is a clear indication of the maximum reproduction path of a datum and, obviously, no major differences are observed.

  The **complexity** and **cohesion** of the second scenario are quite impressing. Ideally, for reasons of maintainability, we would appreciate a scenario with low complexity and high cohesion. The complexity of the second scenario is significantly lower than any other alternative, since, obviously, fewer activities and fewer operations are performed. Although the combined selection activity has the same cohesion with the two individual ones (by a simple application of the formula), the overall cohesion drops due to the smaller number of involved activities. Thus, the cohesion of the second scenario is noticeably higher than the other two. Also, the fact that the cohesion of the combined activity remains the same is not surprising: although two selections are performed, the number of attributes involved increases, so the fraction remains stable.

Table 1. Measures for Scenario 1, involving a linear composition of three activities

	Size	Length	Complexity	Cohesion	Coupling
DSA_PS	6	0	7.50	-	5
σ1	14	2	24.00	0.10	10
σ2	14	4	24.00	0.10	10
sum	14	7	20.75	0.29	7
DW_V1	3	8	3.00	-	2
Overall	**51**	**8**	**79.25**	**0.16**	**34**

Table 2. Measures for Scenario 2, where selections are merged

	Size	Length	Complexity	Cohesion	Coupling
DSA_PS	6	0	7.50	-	5
Σ	16	2	28.00	0.10	10
Sum	14	5	20.75	0.29	7
DW_V1	3	6	3.00	-	2
Overall	**39**	**6**	**59.25**	**0.19**	**24**

Table 3. Measures for Scenario 3, where selections are performed in parallel

	Size	Length	Complexity	Cohesion	Coupling
DSA_PS	6	0	7.50	-	5
σ1	14	2	24.00	0.10	10
σ2	14	2	24.00	0.10	10
join	20	4	36.50	0.07	15
sum	14	7	20.75	0.29	7
DW_V1	3	8	3.00	-	2
Overall	**71**	**8**	**115.75**	**0.14**	**49**

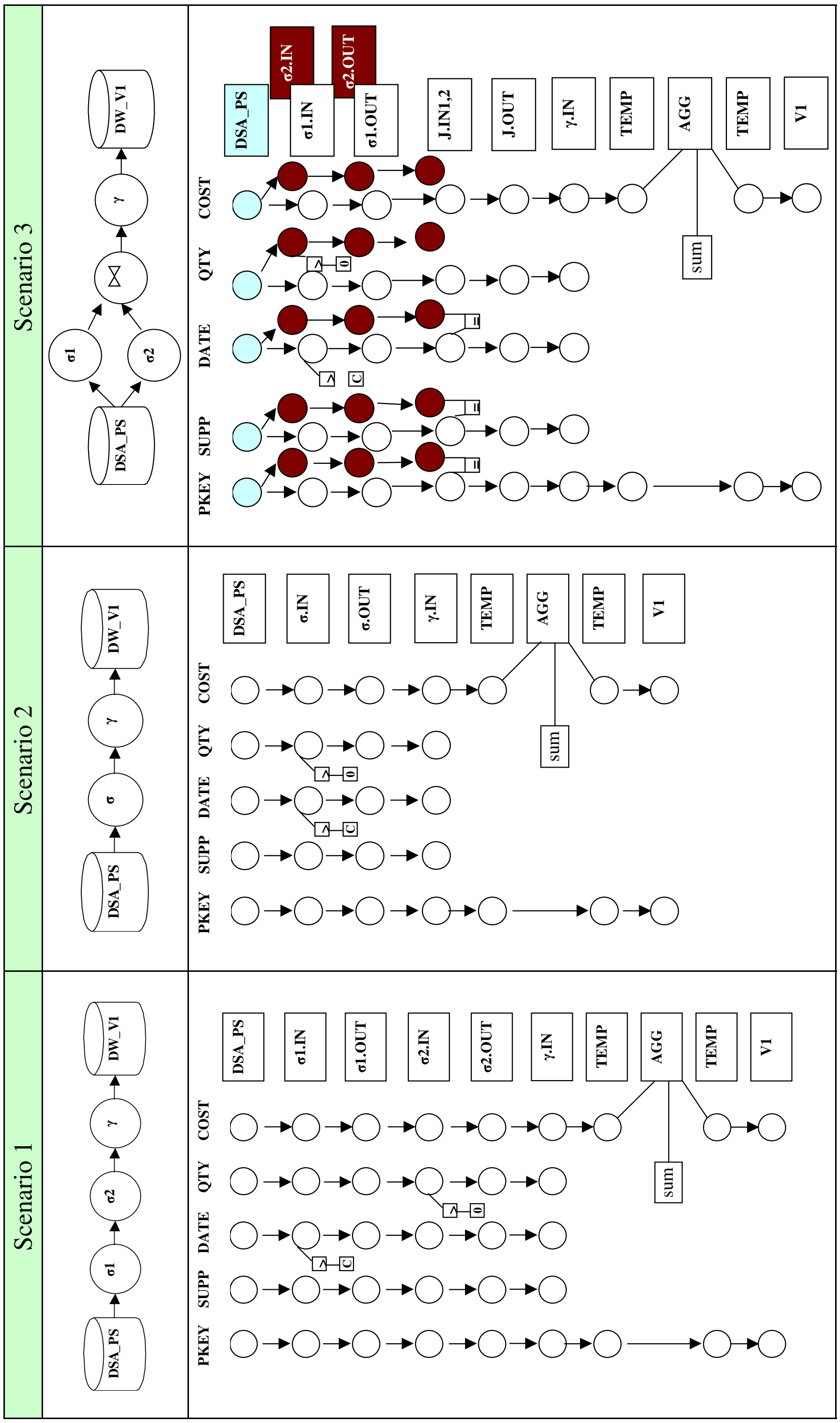


Fig. 4. Equivalent scenarios for the propagation of data from a data source to the warehouse

- The **coupling** is clearly a subset of the complexity measure: by isolating only provider relationships, we clearly see that module interconnections are lowest when the second scenario is employed.

Finally, we can easily observe (both by visualization and measurement) that there exist attributes that should not participate in the workflow, either in the first place (e.g., attribute SUPPKEY) or after their corresponding selections have taken place (e.g., attributes DATE and COST).

4 Related Work

There are several efforts that present systems tailored for ETL tasks [7, 8]. The main focus of these works is on achieving functionality, rather than on modeling the internals or dealing with the software design or maintenance of these tasks. Concerning the conceptual modeling of ETL, [9] and [11] are the first attempts that we know of. The former approach employs UML as a modeling language whereas the latter introduces a generic graphical notation. Still, the focus is only on the conceptual modeling part. As far as the logical modeling of ETL is concerned, in [10, 12] the authors present a graph-based model and an extensible template-based mechanism to define ETL workflows. As already mentioned, in this paper, we extend this model by incorporating the internals of the activity semantics to the graph (more extensions, e.g., updates, can be found in [13]).

Concerning related work on software measurement, we have already mentioned the fundamental works that have guided our approach. Fenton [6] gives the fundamentals of measurement theory and the way they should applied in the case of measuring software artifacts. Briand et al. [2] present the overall framework for defining our particular measures. The particular contribution of this paper is that it gives the principles for defining large categories of software measures. In our case, we prove that the proposed measures fit within the context given by [2]. There is an extensive discussion of software metrics in [4] and an interesting discussion of this area in [5].

5 Conclusions

In this paper, we construct the blueprints for the structure ETL workflows by mapping both their inter-connection and their internal semantics to a graph, which we call the Architecture Graph. The Architecture Graph constitutes the blueprint over which we can perform further analysis for the structure of such a workflow. The first of our contributions involves extending existing results by capturing the internal semantics of each activity in the workflow. We employ the LDL language in order to capture the semantics of ETL activities and we have provided a principled way of transforming LDL programs to the graph. Apart from the value that blueprints have as modeling constructs, we can also exploit them in order to introduce rigorous techniques for the measurement of ETL workflows. To this end, we have built upon the formal framework of [2] and provide software measures to quantify the size, length, complexity, cohesion and coupling of ETL workflows. Several issues omitted in this paper for lack of space, can be found in [13].

Research can be continued in more than one direction. We need an extra step, in order to link our results to the control flow of the graph. Precise algorithms for the evaluation of the impact of changes in the Architecture Graph can also be devised. New metrics can also be discovered, if they appear to reveal properties not covered here. Finally, the usage of the Architecture Graph in all phases of the software lifecycle (e.g., testing) can also be evaluated.

Acknowledgments

This work is partially supported by the European Commission and the Greek Ministry of Education through the EPEAEK Program.

References

1. G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language User Guide. Addison-Wesley, 1999.
2. L.C. Briand, S. Morasca, V.R. Basili. Property-Based Software Engineering Measurement. In IEEE Trans. on Software Engineering, 22(1), Jan 1996.
3. S. Ceri, G. Gottlob, L. Tanca. Logic Programming and Databases. Springer-Verlag, 1990.
4. R.R. Dumke. Software Metrics: a subdivided bibliography. Available at http://irb.cs.uni-magdeburg.de/sw-eng/us/bibliography/bib_main.shtml
5. N.E. Fenton, M. Neil. Software metrics: roadmap. ICSE - Future of SE Track 2000. pp. 357-370, 2000.
6. N. Fenton. Software Measurement: A Necessary Scientific Basis. In IEEE Trans. on Software Engineering, 20(3), March 1994.
7. H. Galhardas, D. Florescu, D. Shasha and E. Simon. Ajax: An Extensible Data Cleaning Tool. In Proc. ACM SIGMOD Intl. Conf. on the Management of Data, pp. 590, Dallas, Texas, 2000.
8. V. Raman, J. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. Proceedings of 27th International Conference on Very Large Data Bases (VLDB'01), pp. 381-390, Roma, Italy, 2001.
9. J. Trujillo, S. Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In Proc. 22nd Intl. Conference on Conceptual Modeling (ER 2003), pp. 307-320, Chicago, IL, USA, October 13-16, 2003.
10. P. Vassiliadis, A. Simitsis, S. Skiadopoulos. Modeling ETL Activities as Graphs. In Proc. 4th Intl. Workshop on Design and Management of Data Warehouses (DMDW'02), pp. 52–61, Toronto, Canada, 2002.
11. P. Vassiliadis, A. Simitsis, S. Skiadopoulos. Conceptual Modeling for ETL Processes. In Proc. 5th ACM Intl. Workshop on Data Warehousing and OLAP (DOLAP), pp. 14–21, McLean, Virginia, USA, 2002.
12. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis. A Framework for the Design of ETL Scenarios. In Proc. 15th Conf. on Advanced Information Systems Engineering (CAiSE '03), pp. 520-535, Klagenfurt/Velden, Austria, June, 2003.
13. P. Vassiliadis, A. Simitsis, M. Terrovitis, S. Skiadopoulos. Blueprints for ETL workflows (long version). Available through http://www.cs.uoi.gr/~pvassil/publications/2005_ER_AG/ETL_blueprints_long.pdf
14. C. Zaniolo. LDL++ Tutorial. UCLA. http://pike.cs.ucla.edu/ldl/, December 1998.

Vague Sets or Intuitionistic Fuzzy Sets for Handling Vague Data: Which One Is Better?

An Lu and Wilfred Ng

Department of Computer Science,
The Hong Kong University of Science and Technology,
Hong Kong, China
{anlu, wilfred}@cs.ust.hk

Abstract. In the real world there are vaguely specified data values in many applications, such as sensor information. Fuzzy set theory has been proposed to handle such vagueness by generalizing the notion of membership in a set. Essentially, in a Fuzzy Set (FS) each element is associated with a point-value selected from the unit interval [0,1], which is termed the grade of membership in the set. A Vague Set (VS), as well as an Intuitionistic Fuzzy Set (IFS), is a further generalization of an FS. Instead of using point-based membership as in FSs, interval-based membership is used in a VS. The interval-based membership in VSs is more expressive in capturing vagueness of data. In the literature, the notions of IFSs and VSs are regarded as equivalent, in the sense that an IFS is isomorphic to a VS. Furthermore, due to such equivalence and IFSs being earlier known as a tradition, the interesting features for handling vague data that are unique to VSs are largely ignored. In this paper, we attempt to make a comparison between VSs and IFSs from various perspectives of algebraic properties, graphical representations and practical applications. We find that there are many interesting differences from a data modelling point of view. Incorporating the notion of VSs in relations, we describe Vague SQL (VSQL), which is an extension of SQL for the vague relational model, and show that VSQL combines the capabilities of a standard SQL with the power of manipulating vague relations. Although VSQL is a minimal extension to illustrate its usages, VSQL allows users to formulate a wide range of queries that occur in different modes of interaction between vague data and queries.

1 Introduction

Fuzzy set theory has long been introduced to handle inexact and imprecise data by Zadeh's seminal paper in [1], since in the real world there is vague information about different applications, such as in sensor databases, we can formalize the measurements from different sensors to a vague set. In fuzzy set theory, each object $u \in U$ is assigned a single real value, called the *grade of membership*, between zero and one. (Here U is a classical set of objects, called the *universe of discourse*.) In [2], Gau et al. point out that the drawback of using the single membership value in fuzzy set theory is that the evidence for $u \in U$ and the evidence against $u \in U$ are in fact mixed together. In order to tackle this problem, Gau et al. propose the notion of *Vague Sets* (VSs), which allow using interval-based membership instead of using point-based membership as in FSs. The interval-based membership generalization in VSs is more expressive in capturing vagueness of data. However, VSs are shown to be equivalent to that of *Intuitionistic Fuzzy Sets* (IFSs) [3,4,5,6] in [7]. For this reason, the interesting features for handling vague data that are unique to VSs are largely ignored.

In this paper, we attempt to make a more detailed comparison between VSs and IFSs from various perspectives of algebraic properties, graphical representations and practical applications. We find that there are many interesting features of VSs from a data modelling point of view. Essentially, due to the fact that a VS corresponds to a more intuitive graphical view of data sets, it is much easier to define and visualize the relationship of vague data objects. The classical nulls representing incompleteness can be viewed as a special case of a vague set and then generalized to vague data. In addition, we show that the notions of crisp and imprecision in vague data can be captured by interval relationships.

We further incorporate the notion of VSs in relations and describe Vague SQL (VSQL), which is an extension of SQL for the vague relational model. We show that VSQL combines the capabilities of standard SQL with the power of manipulating vague relations. We refine the notion of *degree of similar equality* (S_{EQ}) defined in [8] for comparing data values in order to process the vague selection predicates used in VSQL. Although VSQL is a minimal extension to illustrate its usages, VSQL allows the users to formulate a wide range of queries that occur in different modes of interaction between vague data and queries.

The main contributions of this paper are fourfold. First, we examine in more diversified ways, the notions of VSs and IFSs, which has so far not been done in the literature and leads to the undermining of the development of VSs. Second, we study the relationships between vague memberships and nulls in VSs and present the crisp and the imprecision orders. Third, we discuss the similarity measure between vague values and sets. Finally, we propose VSQL in order to gain more expressive power to formulate queries involving vague information. We classify the interactions between queries and data in four modes and demonstrate how the queries arising in different modes can be formulated using VSQL.

The rest of the paper is organized as follows. Section 2 presents some basic concepts related to VSs and IFSs. We also briefly discuss some ways of measuring vagueness in practice. In Section 3, we discuss the representations and the graphical view of nulls in VSs. In Section 4, we present the median and the imprecision membership of VSs and the crisp and the imprecision order in VSs. In Section 5, we discuss the similarity of vague values and sets. In Section 6, we propose VSQL, which is powerful enough to retrieve the data of a specified degree of vagueness. In Section 7 we offer our concluding remarks.

2 Vague Sets and Intuitionistic Fuzzy Sets

In this section, we introduce some basic concepts related to Vague Sets (VSs) and Intuitionistic Fuzzy Sets (IFSs). We illustrate that the graphical representation of VSs is more intuitive in perceiving vague values.

2.1 Basics

Let U be a classical set of objects, called the universe of discourse, where an element of U is denoted by u.

Definition 1. (Fuzzy Set) *A fuzzy set $A = \{< u, \mu_A(u) > |u \in U\}$ in a universe of discourse U is characterized by a membership function, μ_A, as follows: $\mu_A : U \to [0,1]$.*

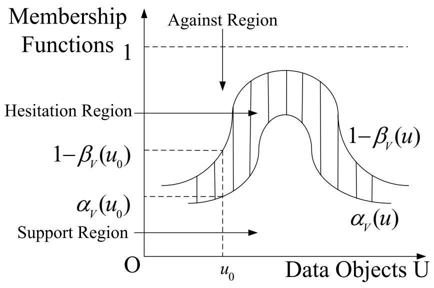
Definition 2. (Vague Set) *A vague set V in a universe of discourse U is characterized by a true membership function, α_V, and a false membership function, β_V, as follows: $\alpha_V : U \to [0,1], \beta_V : U \to [0,1]$, and $\alpha_V(u) + \beta_V(u) \leq 1$, where $\alpha_V(u)$ is a lower bound on the grade of membership of u derived from the evidence for u, and $\beta_V(u)$ is a lower bound on the grade of membership of the negation of u derived from the evidence against u.*

Suppose $U = \{u_1, u_2, \ldots, u_n\}$. A vague set V of the universe of discourse U can be represented by $V = \sum_{i=1}^{n}[\alpha(u_i), 1 - \beta(u_i)]/u_i$, where $0 \leq \alpha(u_i) \leq 1 - \beta(u_i) \leq 1$ and $1 \leq i \leq n$. In other words, the grade of membership of u_i is bounded to a subinterval $[\alpha_V(u_i), 1 - \beta_V(u_i)]$ of $[0,1]$. Thus, VSs are a generalization of FSs, since the grade of membership $\mu_V(u)$ of u in Definition 1 may be inexact in a VS. The idea of membership generalization via an interval has actually proposed earlier as Intuitionistic Fuzzy Sets (IFSs) [3,4] as follows:

Definition 3. (Intuitionistic Fuzzy Sets) *An intuitionistic fuzzy set $A = \{< u, \mu_A(u), \nu_A(u) > |u \in U\}$ in a universe of discourse U is characterized by a membership function, μ_A, and a non-membership function, ν_A, as follows: $\mu_A : U \to [0,1], \nu_A : U \to [0,1]$, and $0 \leq \mu_A(u) + \nu_A(u) \leq 1$.*

As we can see that the difference between VSs and IFSs is due to the definition of membership intervals. We have $[\alpha_V(u), 1 - \beta_V(u)]$ for u in V but $< \mu_A(u), \nu_A(u) >$ for u in A. Here the semantics of μ_A is the same as with α_V and ν_A is the same as with β_V. However, the boundary $(1 - \beta_V)$ is able to indicate the possible existence of a data value, as already mentioned in [7]. This subtle difference gives rise to a simpler but meaningful graphical view of data sets. We now depict a VS in Fig. 1 and an IFS in Fig. 2 respectively. It can be seen that, the shaded part formed by the boundary in a given VS in Fig. 1 naturally represents the *possible existence of data*. Thus, this "hesitation region" corresponds to the intuition of representing vague data.

We will see more benefits of using vague membership intervals in capturing data semantics in subsequent sections. The choice of the membership boundary also has interesting implications on modelling relationship between vague data.



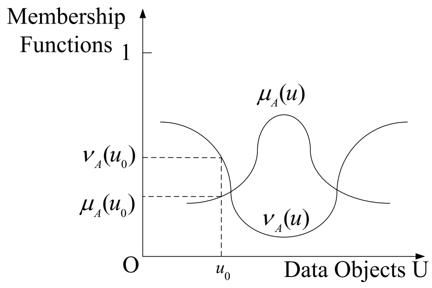


Fig. 1. Membership Functions of a VS **Fig. 2.** Membership Functions of an IFS

2.2 Algebraic Operations

In this subsection, we present the basic operations of VSs and IFSs, which include complement, containment, equal, union, intersection, and so on. The details of most operations related to VSs can be consulted from [2].

Definition 4. (Complement) *The complement of a vague set V is denoted by V' and is defined by*

$$\alpha_{V'}(u) = \beta_V(u),$$
$$1 - \beta_{V'}(u) = 1 - \alpha_V(u).$$

Definition 5. (Containment) *A vague set V_A is contained in another vague set V_B, $V_A \subseteq V_B$, if and only if,*

$$\alpha_{V_A}(u) \leq \alpha_{V_B}(u),$$
$$1 - \beta_{V_A}(u) \leq 1 - \beta_{V_B}(u).$$

Definition 6. (Equal) *Two vague sets V_A and V_B are equal, written as $V_A = V_B$, if and only if, $V_A \subseteq V_B$ and $V_B \subseteq V_A$; that is*

$$\alpha_{V_A}(u) = \alpha_{V_B}(u),$$
$$1 - \beta_{V_A}(u) = 1 - \beta_{V_B}(u).$$

Definition 7. (Union) *The union of two vague sets V_A and V_B is a vague set V_C, written as $V_C = V_A \cup V_B$, whose true membership and false membership functions are related to those of V_A and V_B by*

$$\alpha_{V_C}(u) = max(\alpha_{V_A}(u), \alpha_{V_B}(u)),$$
$$1 - \beta_{V_C}(u) = max(1 - \beta_{V_A}(u), 1 - \beta_{V_B}(u)) = 1 - min(\beta_{V_A}(u), \beta_{V_B}(u)).$$

Definition 8. (Intersection) *The intersection of two vague sets V_A and V_B is a vague set V_C, written as $V_C = V_A \cap V_B$, whose true membership and false membership functions are related to those of V_A and V_B by*

$$\alpha_{V_C}(u) = min(\alpha_{V_A}(u), \alpha_{V_B}(u)),$$
$$1 - \beta_{V_C}(u) = min(1 - \beta_{V_A}(u), 1 - \beta_{V_B}(u)) = 1 - max(\beta_{V_A}(u), \beta_{V_B}(u)).$$

As a comparison, we present the counterpart operations for IFSs [3].

Definition 9. *If A and B are two IFSs of the set U, then*
$\bar{A} = \{< u, \nu_A(u), \mu_A(u) > | u \in U\}$,
$A \subseteq B$ *iff* $\forall u \in U, \mu_A(u) \leq \mu_B(u)$ *and* $\nu_A(u) \geq \nu_B(u)$,
$A = B$ *iff* $\forall u \in U, \mu_A(u) = \mu_B(u)$ *and* $\nu_A(u) = \nu_B(u)$,
$A \cup B = \{< u, max(\mu_A(u), \mu_B(u)), min(\nu_A(u), \nu_B(u)) > | u \in U\}$, *and*
$A \cap B = \{< u, min(\mu_A(u), \mu_B(u)), max(\nu_A(u), \nu_B(u)) > | u \in U\}$.

2.3 Measurements of Vagueness in Practice

We now discuss some ideas of how to measure memberships of VSs and IFSs. There is actually no consensus in the interpretation of what a membership grade means in the literature [9].

In [10], Bilgiç and Türkşen present a review of various interpretations of the fuzzy membership function and the ways of obtaining a membership function. VSs also share similar interpretation of membership grades. For example, the vague predicate "John is tall" is given by an interval in the unit interval, [0.6,0.8]. There are several possible views on how to measure the membership:

Likelihood view: 60-80% of a given population declares that John is tall.

Random set view: 60-80% of a given population describes "tall" as an interval containing John's height.

Similarity view: John's height is away from the *benchmark* object which is truly "tall" to the degree 0.2-0.4. Here if we assume a benchmark example of "tall" is 250cm with the full degree [1,1], then John's height is away from 250cm to the degree 0.2-0.4 means his height is between $(1 - 0.4) \times 250$ and $(1 - 0.2) \times 250$cm, that is, 150-200cm.

For IFSs, we may have the following interpretations:

Likelihood view: 60% of a given population declares that John is tall while 20% does not. (Another 20% is neutral.)

Random set view: 60% of a given population describes "tall" as an interval containing John's height while 20% does not. (Another 20% is neutral.)

Similarity view: The same as in the VS case.

The following is a more detailed example which helps to understand the collection of vague data, as well as IFS data.

Example 1. In a sensor database application, suppose in a testing region we have a set of ten sensors $\{S_1, S_2, \ldots, S_{10}\}$. We then obtain ten corresponding measurements, {20, 22, 20, 21, 20, -, 20, 20, -, 20} at a certain time t. Here "-" means that the sensor data is not reachable/accessible at time t. (i.e. we have six 20, one 21, one 22 and two missing values.) Now, we formalize the results to a vague set V_t as follows. There are six occurrences of 20, but two values (21 and 22) are against it. There are also two missing values (neutral), thus the true membership α is 0.6 and the false membership β is 0.2 (i.e. 1-β = 0.8). Thus, we obtain the vague membership value [0.6,0.8] for 20. Similarly, we obtain the vague membership value [0.1,0.3] for 21 and [0.1,0.3] for 22. Combining these results, we have the VS, V_t = [0.6,0.8]/20+[0.1,0.3]/21+[0.1,0.3]/22. Equivalently, we have the IFS, $A_t = \{< 20, 0.6, 0.2 >, < 21, 0.1, 0.7 >, < 22, 0.1, 0.7 >\}$.

The above example also indicates that, using a VS is more natural than an IFS for merging fuzzy objects. For example, suppose we merge three fuzzy values $0.4/u$, $0.5/u$ and $0.6/u$. We can then directly obtain the vague value $[0.4, 0.6]/u$, which means that the lower bound of the membership of u is the minimum of the fuzzy membership, 0.4, and that the upper bound is the maximum of the fuzzy membership, 0.6. However, by using the intuitionistic fuzzy value we have $< u, 0.4, 0.4 >$, which is much less intuitive.

3 Relationships Between VS Memberships and Nulls

In this section, we discuss how VSs capture different notions of incompleteness. We need to define an empty vague set first.

Definition 10. (Empty Vague Set) *A vague set V is an empty vague set, if and only if, its true membership function $\alpha = 0$ and false membership function $\beta = 1$ for all u. We use $\emptyset_V$ to denote it.*

It is worth mentioning that $\emptyset_V$ can be regarded as the generalization of the empty set in the ordinary set theory, which is not the same empty concept as defined in [2]. In [2], a vague set V is empty, if and only if, its true membership function α and false membership function β are both 0, which means that we have no information about whether the corresponding object belongs to the vague set or not. However, our definition of empty vague set means that we know exactly that no object belongs to the empty vague set. Furthermore, we define *empty vague value*, or simply *empty value*, as $[0, 0]$ (i.e. $\alpha = 0$, $1 - \beta = 0$).

We first review the three kinds of classical null values and the crisp value, which are represented in Fig. 3 and then generalize them to the vague domain as shown in Fig. 4.

1. Unknown (UNK) represents that the value (a classical data object) exists but is unknown at the represent time. In crisp sets, all memberships of objects are assumed as [1,1], which can be regarded as a special case of

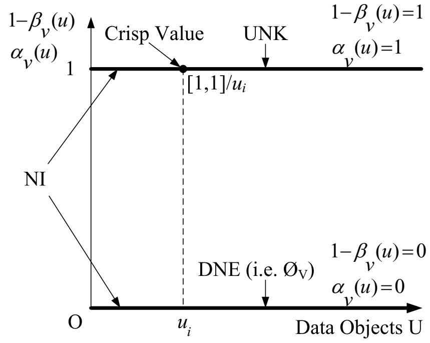


Fig. 3. Classical Nulls in a VS

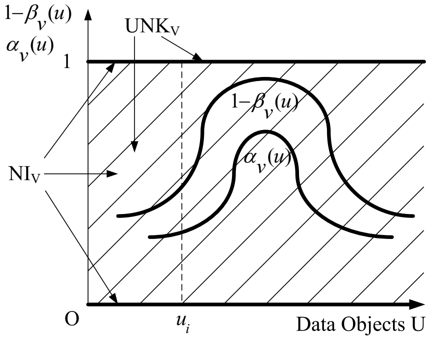


Fig. 4. Generalized Nulls in a VS

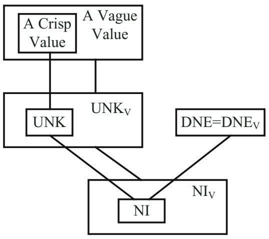


Fig. 5. Relationships between Various Cases of Nulls

VSs. For example, it may not be known at the present time the AGE of an EMPLOYEE in an employee relation. We can view UNK in the form of a VS by $V = \sum_{i=1}^{n}[1,1]/u_i$, which means that each object $u_i \in U$ $(1 \leq i \leq n)$ may "totally" belong to the vague set V. The UNK is represented as the horizontal lines such that $\alpha_V(u_i) = 1 - \beta_V(u_i) = 1$ in Fig. 3.

2. Does Not Exist (DNE) presents that "the value is inapplicable". For example, if Tom has not married, the SpouseName of Tom in an employee relation is denoted as DNE. We can view DNE in the form of a VS, V, as given by $V = \sum_{i=1}^{n}[0,0]/u_i$ $(1 \leq i \leq n)$, which means that we are sure that u_i does not belong to the vague set and the evidence is totally against it, that is, $\alpha_V(u_i) = 0$ but $\beta_V(u_i) = 1$. Thus, we obtain the empty vague set $\emptyset_V$. The DNE is represented as the horizontal line as shown in Fig. 3.
3. No Information (NI) represents that "no information is available for the values", i.e. it is either UNK or DNE. For example, it is not known if an EMPLOYEE has got married or not when there is no data in STATUS. The NI is represented as horizontal lines of UNK and DNE in Fig. 3.

The above null values are limited to the crisp sets which contain classical data objects. We now generalize the notions of UNK and NI to VSs as follows. Note that such a generalization has no effect on DNE which is still the U-axis.

1. Generalized UNK (UNK_V) represents that memberships can be vague values but are unknown at the represent time. We can view UNK_V as any possible VS such that, $\forall u_i \in U$, $\alpha_V(u_i) \in (0,1]$ and $\beta_V(u_i) \in (0,1]$, as shown in Fig. 4, i.e. the shaded area excluding the DNE line (excluding the U-axis) but including the UNK line.
2. Generalized DNE (DNE_V) is the same as DNE.
3. Generalized NI (NI_V) is the region of NI_V which includes the DNE line and the UNK_V region as shown in Fig. 4.

Based on the above discussion, we use Fig. 5 to illustrate the relationships between various cases of nulls discussed so far.

4 Relationships Between VS Memberships

In this section, we discuss the following relationships of vague membership values (vague values for short) in VSs: crisp and imprecision. Remarkably, there are no

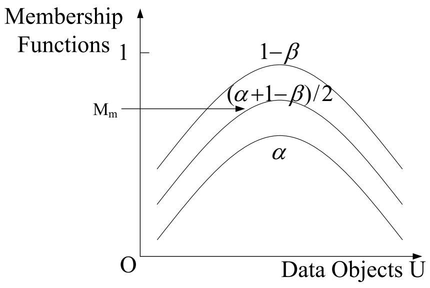


Fig. 6. Median Membership of a VS

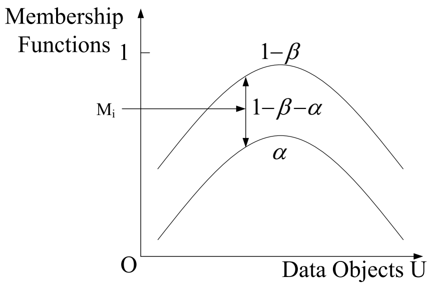


Fig. 7. Imprecision Membership of a VS

such meaningful relationships based on IFS membership values. We also show two lattices arising from the crisp and the imprecision orders.

In order to compare vague values, we need to introduce two derived memberships for discussion.

The first is called the *median membership*, $M_m = (\alpha + 1 - \beta)/2$, which represents the overall evidence contained in a vague value and is shown in Fig. 6. It can be checked that $0 \leq (\alpha + 1 - \beta)/2 \leq 1$. In addition, the vague value [1,1] has the highest M_m, which means the corresponding object totally belongs to the VS (i.e. a crisp value). While the vague value [0,0] has the lowest M_m which means that the corresponding object totally does not belong to the VS (i.e. the empty vague value).

The second is called the *imprecision membership*, $M_i = (1 - \beta - \alpha)$, which represents the overall imprecision of a vague value and is shown in Fig. 7. It can be checked that $0 \leq (1-\beta-\alpha) \leq 1$. In addition, the vague value $[a, a](a \in [0, 1])$ has the lowest M_i which means that we know exactly the membership of the corresponding object (i.e. a fuzzy value). While the vague value [0,1] has the highest M_i which means that we know nothing about the membership of the corresponding object.

Suppose x and y are two vague values defined for a certain object u such that $x = [\alpha_x, 1 - \beta_x]$, $y = [\alpha_y, 1 - \beta_y]$. We then have the following crisp order and the imprecision order relating the x and y membership intervals, which are analogous to the "degree of truth" and the "amount of knowledge" in [11].

Definition 11. (Crisp Order and Imprecision Order) *Let x and y are two vague values defined for a certain object u. We say x is less crisp than y, denoted as $x \leq_c y$, if $\alpha_x \leq \alpha_y$ and $1 - \beta_x \leq 1 - \beta_y$. We say x is less imprecise than y, denoted as $x \leq_i y$, if $\alpha_x \geq \alpha_y$ and $1 - \beta_x \leq 1 - \beta_y$.*

It is straightforward to check that $\leq_i$ and $\leq_c$ are two orthogonal concepts. For example, [0.7, 0.9] $\leq_c$ [0.8, 0.9] but [0.7, 0.9] $\not\leq_i$ [0.8, 0.9]. On the other hand, [0.7, 0.9] $\leq_i$ [0.6, 0.9] but [0.7, 0.9] $\not\leq_c$ [0.6, 0.9]. The other interesting relationships between the crisp order and the imprecision order, and the three interval relationships of precedence, overlap and contain are depicted in Fig. 8.

The relationship between the crisp and imprecision order, and the median and the imprecision memberships is as follows.

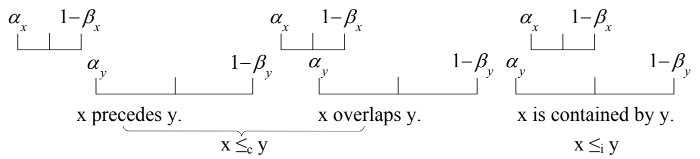


Fig. 8. Interaction between Crisp and Imprecision Order, and Precedence, Overlap and Contain Relationships

1. If $x \leq_c y$, then $M_m(x) \leq M_m(y)$, but not vice versa.
2. If $x \leq_i y$, then $M_i(x) \leq M_i(y)$, but not vice versa.

We now use $\wedge$ and $\vee$ for c-meet and c-join under $\leq_c$, and $\otimes$ and $\oplus$ for i-meet and i-join under $\leq_i$. We define $[\alpha_x, 1-\beta_x] \wedge [\alpha_y, 1-\beta_y] = [min\{\alpha_x, \alpha_y\}, min\{1-\beta_x, 1-\beta_y\}]$ and $[\alpha_x, 1-\beta_x] \vee [\alpha_y, 1-\beta_y] = [max\{\alpha_x, \alpha_y\}, max\{1-\beta_x, 1-\beta_y\}]$. We define $[\alpha_x, 1-\beta_x] \otimes [\alpha_y, 1-\beta_y] = [min\{\alpha_x, \alpha_y\}, max\{1-\beta_x, 1-\beta_y\}]$ and $[\alpha_x, 1-\beta_x] \oplus [\alpha_y, 1-\beta_y] = [max\{\alpha_x, \alpha_y\}, min\{1-\beta_x, 1-\beta_y\}]$. It is easy to check that the crisp order $\leq_c$ induces a complete lattice by using $\wedge$ and $\vee$. Under the crisp order, [0,0] is the bottom ($\perp_c$), and [1,1] is the top ($\top_c$). On the other hand, the imprecision order $\leq_i$ induces a complete semi-lattice by using $\otimes$ and $\oplus$. It should be noted that i-join is *not defined* when $max\{\alpha_x, \alpha_y\} > min\{1-\beta_x, 1-\beta_y\}$, since the join result is not a valid interval. From now on, we restrict our discussion to the i-join that gives rise to valid intervals as a result. Under the imprecision order, [0,1] is the top ($\top_i$), but there is no bottom for the semi-lattice.

Theorem 1. *The following statements are true.*

1. *If $x \leq_i y$, then*
$$\begin{cases} x \leq_i (x \wedge y) \leq_i y; \\ x \leq_i (x \vee y) \leq_i y. \end{cases} \quad \begin{cases} (x \wedge y) \leq_c x \leq_c (x \vee y); \\ (x \wedge y) \leq_c y \leq_c (x \vee y). \end{cases} \quad \begin{cases} x \otimes y = y; \\ x \oplus y = x. \end{cases}$$
2. *If $x \leq_c y$, then*
$$\begin{cases} x \leq_c (x \otimes y) \leq_c y; \\ x \leq_c (x \oplus y) \leq_c y. \end{cases} \quad \begin{cases} (x \oplus y) \leq_i x \leq_i (x \otimes y); \\ (x \oplus y) \leq_i y \leq_i (x \otimes y). \end{cases} \quad \begin{cases} x \wedge y = x; \\ x \vee y = y. \end{cases}$$

As an example, we discretize [0,1] to the granularity of 0.1 unit for α and $1-\beta$ and show in Fig. 9 a complete lattice induced by the crisp order, which is along the crisp dimension, and a complete semi-lattice induced by the imprecision order, which is along the imprecision dimension. The distance between two adjoining elements is 0.1. For example, it can be checked that [0.3,0.4] $\wedge$ [0.1,0.7] = [0.1,0.4], and [0.3,0.4] $\vee$ [0.1,0.7] = [0.3,0.7] in the crisp dimension. It can also be checked from Fig. 9 in the imprecision dimension that, [0,0.6] $\otimes$ [0.4,0.8] = [0,0.8], and [0,0.6] $\oplus$ [0.4,0.8] = [0.4,0.6]. However, [0,0.1] $\oplus$ [0.2,0.3] is undefined, and there is no Greatest Lower Bound (GLB) for these two vague values in the imprecision lattice. The lattice size is exponential to the square of the discretization on the unit membership interval.

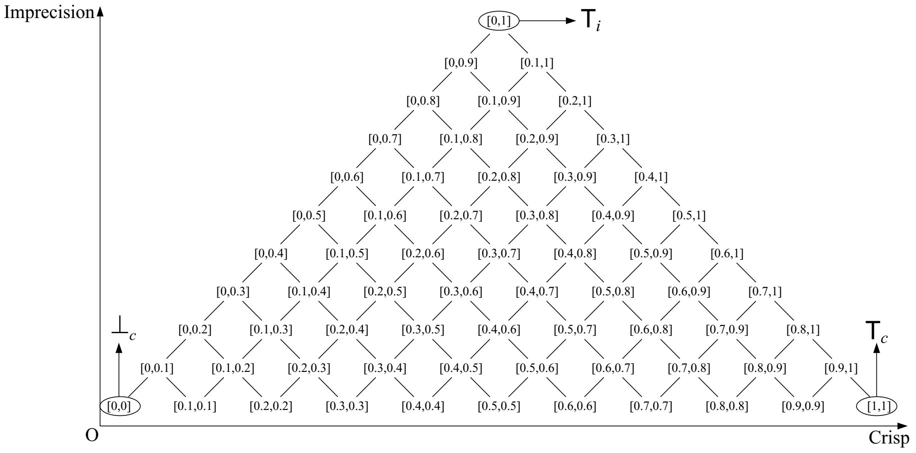


Fig. 9. Complete Imprecision Semi-Lattice and Complete Crisp Lattice

5 Similarity Measures of VSs

In this section, we discuss a similarity measure between two VSs, which is based on the median membership and the imprecision membership.

We now let x and y be two vague values to some $u \in U$ such that $x = [\alpha_x, 1-\beta_x]$, $y = [\alpha_y, 1-\beta_y]$. We define ΔM_m as the difference between median memberships, which is given by $\Delta M_m = |(\alpha_x + 1 - \beta_x) - (\alpha_y + 1 - \beta_y)|/2 = |(\alpha_x - \alpha_y) - (\beta_x - \beta_y)|/2$, such that $0 \le \Delta M_m \le 1$. We define ΔM_i as the difference between imprecision memberships, which is given by $\Delta M_i = |(1 - \beta_x - \alpha_x) - (1 - \beta_y - \alpha_y)| = |(\alpha_x - \alpha_y) + (\beta_x - \beta_y)|$, such that $0 \le \Delta M_i \le 1$. We define the similarity measure between two vague values x and y as follows:

Definition 12. (Similarity Measure between Two Vague Values)

$$M(x,y) = \sqrt{(1-\Delta M_m)(1-\Delta M_i)}$$
$$= \sqrt{(1 - \frac{|(\alpha_x - \alpha_y) - (\beta_x - \beta_y)|}{2})(1 - |(\alpha_x - \alpha_y) + (\beta_x - \beta_y)|)}.$$

We extend the similarity measure from two vague values to two vague sets.

Definition 13. (Similarity Measure between Two Vague Sets) *Let X and Y be two vague sets, where*

$$X = \sum_{k=1}^{n} [\alpha_X(u_k), 1 - \beta_X(u_k)]/u_k; \; Y = \sum_{k=1}^{n} [\alpha_Y(u_k), 1 - \beta_Y(u_k)]/u_k.$$

The similarity measure between the vague sets X and Y can be evaluated as follows:

$$M(X,Y) = \frac{1}{n} \sum_{k=1}^{n} M([\alpha_X(u_k), 1 - \beta_X(u_k)], [\alpha_Y(u_k), 1 - \beta_Y(u_k)])$$

$$= \frac{1}{n}\sum_{k=1}^{n}\sqrt{(1-\Delta M_{m,k})(1-\Delta M_{i,k})}$$

$$= \frac{1}{n}\sum_{k=1}^{n}\sqrt{(1-\frac{|(\alpha_X(u_k)-\alpha_Y(u_k))-(\beta_X(u_k)-\beta_Y(u_k))|}{2})}\cdot$$
$$\sqrt{(1-|(\alpha_X(u_k)-\alpha_Y(u_k))+(\beta_X(u_k)-\beta_Y(u_k))|)}.$$

The following theorem illustrates some good features for similarity measure between vague sets, which follows from Definition 13.

Theorem 2. *Let X and Y be two vague sets. The following statements are true.*

1. *The similarity measure is bounded, i.e., $0 \leq M(X,Y) \leq 1$.*
2. *$M(X,Y) = 1$ if and only if $X = Y$.*
3. *$M(X,Y) = 0$, if and only if, all the vague values in X and Y are (i) $[0,0]$ and $[1,1]$, or (ii) $[0,1]$ and $[a,a]$, where $0 \leq a \leq 1$.*
4. *The similarity measure is commutative, i.e., $M(X,Y) = M(Y,X)$.*

6 Vague Relations and VSQL

In this section, we propose VSQL in order to gain more expressive power to formulate queries involving vague information. We classify the interactions between queries and data in four modes and demonstrate how the queries arising in different modes can be formulated using VSQL.

6.1 Vague Relations

Definition 14. (Vague Relation) *Let $\mathcal{U} = \{U_1,\ldots,U_m\}$ be a collection of universes of discourse. Let $Dom(A_i)$ be the domain corresponding to the attribute A_i. We define $Dom(A_i) = \{V \mid V \text{ is a VS of } U_i\}$. A vague tuple $t = (a_1,a_2,\ldots,a_m)$ over a relation scheme, $R = \{A_1,A_2,\ldots,A_m\}$, is an element in $Dom(A_1) \times Dom(A_2) \times \cdots \times Dom(A_m)$. A vague relation r over R is a subset of $Dom(A_1) \times Dom(A_2) \times \cdots \times Dom(A_m)$.*

Unlike classical and fuzzy relations, in vague relations, $Dom(A_i)$ is a set of vague sets. Vague relations can be considered as an extension of classical relations (all vague values are [1, 1]) and fuzzy relations (all vague values are $[a,a]$, $0 \leq a \leq 1$), which can capture more information about vagueness.

Consider the vague relation r over Product(ID, Weight, Price) given in Table 1. In r, Weight and Price are vague attributes. The attribute ID is crisp, where values are presented as the usual atomic values. The first tuple in r means the product with ID=1 has the weight of $[1,1]/10$ and the price of $[0.4,0.6]/50+[1,1]/80$.

6.2 Similar Equality

In this subsection, we extend the notion of *similar equality* (S_{EQ}) defined in [8] for comparing data values to deal with selection predicates in VSQL.

The *degree of similar equality* (S_{EQ}) of vague relations defined below can be used as a vague similarity measure to compare elements of a given domain. Suppose t_p and t_q are two tuples in a relation r.

Definition 15. (Degree of Similar Equality) *The degree of similar equality of two vague tuples t_p and t_q on the attribute A_i in a vague relation is given by:*

$$S_{EQ}(t_p[A_i], t_q[A_i]) = \frac{1}{n}\sum_{k=1}^{n}\sqrt{(1-\varDelta M_{m,k})(1-\varDelta M_{i,k})}$$

$$= \frac{1}{n}\sum_{k=1}^{n}\sqrt{\left(1-\frac{|(\alpha_{t_p[A_i]}(u_k)-\alpha_{t_q[A_i]}(u_k))-(\beta_{t_p[A_i]}(u_k)-\beta_{t_q[A_i]}(u_k))|}{2}\right)}\cdot$$
$$\sqrt{(1-|(\alpha_{t_p[A_i]}(u_k)-\alpha_{t_q[A_i]}(u_k))+(\beta_{t_p[A_i]}(u_k)-\beta_{t_q[A_i]}(u_k))|)}.$$

The degree of similar equality of two vague tuples t_p and t_q on attributes $X = \{A_1, \ldots, A_s\}$ ($X \subseteq R$) in a vague relation is $S_{EQ}(t_p[X], t_q[X]) = min\{S_{EQ}(t_p[A_1], t_q[A_1]), \ldots, S_{EQ}(t_p[A_s], t_q[A_s])\}$.

6.3 VSQL

There have been some studies which discuss the topic concerning fuzzy SQL queries in fuzzy databases [12,13] which only cater for true membership (or we can say they combine true membership and false membership together). We now describe the extensions of VSQL to standard SQL. The VSQL is powerful enough to retrieve any set of items of any degree of vagueness.

Data Definition Language. The syntax of VSQL allows users to define semantic domains using the CREATE DOMAIN command as follows:

CREATE DOMAIN <domain name> <data types>.

The command is similar to the SQL standard statement that declares a domain. In Vague SQL DDL, attributes constraints for vague data and vague data definitions are added to the standard DDL. By using the keyword VAGUE, the attributes which store vague data are specified. For example, we define a vague domain SCALE as follows:

CREATE DOMAIN SCALE VAGUE INTEGER.

Table 1. A Vague Relation r

ID	Weight	Price
1	[1,1]/10	[0.4,0.6]/50+[1,1]/80
2	[1,1]/20	[0.8,0.9]/100+[0.6,0.8]/150
3	[1,1]/20	[1,1]/100+[0.7,0.9]/150
4	[1,1]/10+[0.6,0.8]/15	[1,1]/80+[0.6,0.8]/100
5	[0.6,0.8]/10+[1,1]/15+[0.6,0.8]/20	[0.6,0.8]/80+[1,1]/100

Vague Data Definition. Vague data to be used in VSQL DML are defined in VSQL DDL. Vague data are defined by specifying the table and attribute in which they are used. As an example, we define the vague data for the attribute "Weight" in Table 1.

CREATE VAGUE DATA ON SCALE(4) AS light = [1,1]/0 + [0.8,1]/10 + [0.7,0.9]/20 + [0.6,0.8]/30 + [0.5,0.7]/40 + [0.4,0.6]/50 + [0.3,0.5]/60 + [0.2,0.4]/70 + [0.1,0.3]/80 + [0,0.2]/90 + [0,0]/100.

CREATE VAGUE DATA ON SCALE(4) AS middle = [0,0]/0 + [0.1,0.3]/10 + [0.3,0.5]/20 + [0.5,0.7]/30 + [0.7,0.9]/40 + [1,1]/50 + [0.7,0.9]/60 + [0.5,0.7]/70 + [0.3,0.5]/80 + [0.1,0.3]/90 + [0,0]/100.

CREATE VAGUE DATA ON SCALE(4) AS heavy = [0,0]/0 + [0,0.2]/10 + [0.1,0.3]/20 + [0.2,0.4]/30 + [0.3,0.5]/40 + [0.4,0.6]/50 + [0.5,0.7]/60 + [0.6,0.8]/70 + [0.7,0.9]/80 + [0.8,1]/90 + [1,1]/100.

Table Definition. An example of Vague SQL DDL is shown as follows. Here SCALE(4) means the value of "Weight" is a 4-digit vague integer.

CREATE TABLE Product (ID INTEGER(8) NOT NULL, Name CHAR(16) NOT NULL, Weight SCALE(4), Size SCALE(8), Price SCALE(8)).

Data Manipulation Language. The expression of a basic VSQL query is given as follows:

SELECT <lists of attributes> [ANY|ALL] [ASC|DESC] FROM <lists of vague relations> WHERE <extended predicates>.

An attribute list is a list of attributes similar to the usual one, except that it provides us with an option that an attribute can be associated with a vague domain.

In Vague SQL DML, vague data can be used for many kinds of operator such as predicates, and the like. An example of Vague SQL DML is shown as follows.

Query: "Find the products which are heavy."

VSQL: SELECT ID, Weight, Price FROM Product WHERE Weight=heavy.

Here "heavy" is a vague set as mentioned above. Following the FROM keyword is a comma separated list of all relations used in a query. A typical form of a semantic comparison is: <attribute> <comparator> <attribute>.

6.4 VSQL Modes

We consider that data vagueness can occur in both relations and query expressions. Thus, we develop VSQL and allow users to formulate a wide range of vague queries that occur in different modes of interaction between the data and the queries. We classify our VSQL Modes as shown in Fig. 10. We now show the power of VSQL in formulating queries in these modes.

Mode I (Crisp Data, Conventional SQL). The first mode concerns only conventional region of SQL usages, where data values and VSQL are both crisp (i.e. no vague data involved). This mode is no different from the classical relational databases and therefore VSQL is downward compatible to SQL. For

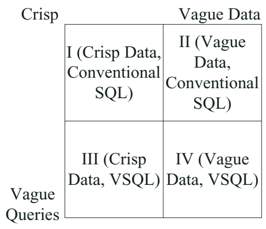


Fig. 10. The four modes of VSQL

example, Table 2 represents a classical relational table in which all data are crisp. The query (Q_1) falls into this region.

(Q_1) "Find the products which are equal to 20 kg."

SELECT * FROM Product WHERE Weight = 20.

Thus, we obtain the answers given in Tables 4.

Mode II (Vague Data, Conventional SQL). The second mode concerns the scenario that data values are vague but queries are conventional. We allow classical SQL statements referencing vague tables to be formulated. For example, Table 3 represents a vague relational table where Weight and Price data are vague. (The vague data "light", "middle" and "heavy" are defined in Section 6.3.) We now formulate the query (Q_2) as below.

(Q_2) "Find the products which are equal to 20 kg."

SELECT * FROM Product WHERE Weight = 20.

We first transform 20 into the VS [1,1]/20, and then determine the S_{EQ} between the "Weight" values in r_2 (also in the form of VSs) and [1,1]/20. For example, for the tuple with ID = 4 in r_2 given in Table 3, according to Definition

Table 2. A Crisp Product Relation r_1

ID	Weight	Price
1	10	50
2	20	100
3	20	150
4	50	200
5	80	350

Table 3. A Vague Product Relation r_2

ID	Weight	Price
1	light	[1,1]/50
2	light	[0.6,0.8]/100
3	[1,1]/20 + [0.5,0.6]/50	[0.5,0.9]/150
4	middle	[0.8,0.9]/200
5	heavy	[0.7,1]/350

Table 4. Answer for (Q_1)

ID	Weight	Price
2	20	100
3	20	150

Table 5. Answer for (Q_2)

ID	Weight	Price	*Rank*
3	[1,1]/20 + [0.5,0.6]/50	[0.5,0.9]/150	0.967
1	light	[1,1]/50	0.624
2	light	[0.6,0.8]/100	0.624
4	middle	[0.8,0.9]/200	0.617
5	heavy	[0.7,1]/350	0.551

Table 6. Answer for (Q_3)

ID	Weight	Price	*Rank*
5	80	350	0.624
4	50	200	0.587
2	20	100	0.551
3	20	150	0.551
1	10	50	0.536

Table 7. Answer for (Q_4)

ID	Weight	Price	*Rank*
5	heavy	[0.7,1]/350	1
4	middle	[0.8,0.9]/200	0.750
1	light	[1,1]/50	0.591
2	light	[0.6,0.8]/100	0.591
3	[1,1]/20 + [0.5,0.6]/50	[0.5,0.9]/150	0.578

15, we obtain $S_{EQ}(t_4[Weight], [1,1]/20) = S_{EQ}(middle, [1,1]/20) = 0.617$. Then we rank the tuples by this S_{EQ} value and obtain the answer given in Table 5.

Mode III (Crisp Data, Vague SQL). The third mode concerns the scenario that data values are crisp but SQL is vague. For example, we have the query (Q_3) in Table 2 as follows.

(Q_3) "Find the products which are heavy in weight."

SELECT * FROM Product WHERE Weight = heavy.

We first transform the "Weight" values in r_1 into VSs, and then determine the S_{EQ} between the "Weight" values and the VS "heavy". For example, for the tuple with ID = 2 in r_1 given in Table 2, we obtain $S_{EQ}(t_2[Weight], heavy) = S_{EQ}([1,1]/20, heavy) = 0.551$. Then we rank the tuples by this S_{EQ} value and obtain the answer given in Table 6.

Mode IV (Vague Data, Vague SQL). In the fourth mode, both data values and SQL are vague. We have the query (Q_4) as follows.

(Q_4) "Find the products which are heavy in weight."

SELECT * FROM Product WHERE Weight = heavy.

We first determine the S_{EQ} between the "Weight" values in r_2 and the VS "heavy". For example, for the tuple with ID = 4 in r_2, we obtain $S_{EQ}(t_4[Weight], heavy) = S_{EQ}(middle, heavy) = 0.750$. Then we rank the tuples by this S_{EQ} value and obtain the answer given in Table 7.

7 Conclusions

We have examined the two known generalizations, VSs and IFSs, of FSs. VSs are based on an interval-based membership and thus more expressive in capturing vagueness of data and the notions of IFSs and VSs are regarded as equivalent, in the sense that an IFS is isomorphic to a VS. We compare VSs and IFSs by their notions, algebraic properties, practical applications, and most importantly, the graphical representations of vague data objects.

Although VSs and IFSs are equivalent by basic definition, we show throughout the paper VSs allow for a more intuitive graphical representation of vague data, which facilitates significantly better analysis in data relationships, incompleteness, and similarity measures. When measuring vagueness in practice, we have showed by example that using a VS is more natural than using an IFS, especially for merging fuzzy objects. Using the interval memberships defined in VSs, we study the interactions of vague membership values and present the interesting relationships of crisp order and imprecision order. We also discuss the similarity measures between vague data.

In order to retrieve vague data by the well-established database query language, we incorporate the notion of vagueness into the relational data model and demonstrate how VSQL can be employed to formulate a wide range of queries arising from four modes of query/data interaction. We are still investigating optimization and processing techniques for vague queries. This also relies on the development of efficient indexing schemes for vague data.

References

1. Zadeh, L.A.: Fuzzy sets. Information and Control **8** (1965) 338–353
2. Gao, W.L., Danied, J.B.: Vague sets. IEEE Transactions on Systems, Man, and Cybernetics **23** (1993) 610–614
3. Atanassov, K.T.: Intuitionistic fuzzy sets. Fuzzy Sets and Systems **20** (1986) 87–96
4. Atanassov, K.T.: More on intuitionistic fuzzy sets. Fuzzy Sets and Systems **33** (1989) 37–45
5. Atanassov, K.T., Gargov, G.: Interval-valued intuitionistic fuzzy sets. Fuzzy Sets and Systems **31** (1989) 343–349
6. Atanassov, K.T.: Intuitionistic Fuzzy Sets: Theory and Applications (Studies in Fuzziness and Soft Computing). Springer-Verlag Telos (1999)
7. Bustince, H., Burillo, P.: Vague sets are intuitionistic fuzzy sets. Fuzzy Sets and Systems **79** (1996) 403–405
8. Lu, A., Ng, W.: Managing merged data by vague functional dependencies. In Atzeni, P., Chu, W.W., Lu, H., Zhou, S., Ling, T.W., eds.: ER. Volume 3288 of Lecture Notes in Computer Science., Springer (2004) 259–272
9. Dubois, D., Ostasiewicz, W., Prade, H.: Fuzzy sets: History and basic notions. In Dubois, D., Prade, H., eds.: Fundamentals of Fuzzy Sets, the Handbooks of Fuzzy Sets Series. Kluwer Academic Publishers, Boston, MA (2000) 195–230
10. Bilgiç, T., Türkşen, I.B.: Measurement of membership functions: Theoretical and experimental work. In Dubois, D., Prade, H., eds.: Fundamentals of Fuzzy Sets, the Handbooks of Fuzzy Sets Series. Kluwer Academic Publishers, Boston, MA (2000) 195–230
11. Fitting, M.: Kleene's logic, generalized. J. Log. Comput. **1** (1991) 797–810
12. Bosc, P., Pivert, O.: Sqlf: A relational database language for fuzzy querying. IEEE Transactions on Fuzzy Systems **3** (1995) 1–17
13. Nakajima, H., Sogoh, T., Arao, M.: Fuzzy database language and library - fuzzy extension to sql. In: Proc. Second IEEE Int. Conf. on Fuzzy Systems. (1993) 477–482

A Semantic Approach to Query Rewriting for Integrated XML Data

Xia Yang[1], Mong Li Lee[1], Tok Wang Ling[1], and Gillian Dobbie[2]

[1] School of Computing, National University of Singapore
{yangxia, leeml, lingtw}@comp.nus.edu.sg
[2] Department of Computer Science, The University of Auckland, New Zealand
gill@cs.auckland.ac.nz

Abstract. Query rewriting is a fundamental task in query optimization and data integration. With the advent of the web, there has been renewed interest in data integration, where data is dispersed among many sources and an integrated view over these sources is provided. Queries on the integrated view are rewritten to query the underlying source repositories. In this paper, we develop a novel algorithm for rewriting queries that considers the XML hierarchy structure and the semantic relationship between the source schemas and the integrated schema. Our approach is based on the semantically rich Object-Relationship-Attribute model for SemiStructured data (ORA-SS), and guarantees that the rewritten queries give the expected results, even where the integrated view is complex.

1 Introduction

Many query rewriting algorithms have been developed for answering queries using views in relational databases and in mediators. When answering queries using views, the objective is to find efficient methods to answer a query using a set of materialized views over the database, instead of accessing the database itself [5, 14, 16, 17].

In data integration, many systems construct a global or mediated schema from numerous heterogeneous data sources [6, 13, 18]. Users issue queries on the global schema, and the system will rewrite the query to the local sources. Each local source may not necessarily contain all the information needed to answer the query. Partial results from various local sources are combined to produce the result for the query.

When integration is carried out over XML repositories, query rewriting algorithms need to take into consideration the hierarchical structure of XML schemas. This gives rise to structural conflicts which need to be resolved during the rewriting process [22]. XML schemas such as DTD and XML Schema lack the semantic information necessary for schema integration and query rewriting. Although proposals have been put forth to augment DTD and XML Schema with information such as keys [2], and functional dependencies [10], their semantics remain limited.

In this paper, we describe a rewriting algorithm for integrated views over XML repositories. The proposed algorithm utilizes the ORA-SS model [11] which provides the necessary semantic information to produce expected answers even when the integrated view is complex. In contrast to the work in [12] which describes how relational databases can be integrated into an XML global schema, we assume that the local sources are XML repositories. XML schemas are first transformed to ORA-SS schemas with enriched semantics [3]. An ORA-SS integrated schema can be obtained using the algorithm in [21]. Compared to existing global-as-view approaches which incorporate the integrated view definition in the unfolding process, our approach uses a mapping table that is created during the integration process to rewrite queries. We also use a query allocation table to find groups of local schemas that together can answer a user query. When a query is decomposed to subqueries on the local schemas, the subqueries for each group of local schemas are composed, and answers from the composed queries are combined to give the expected results.

The rest of the paper is organized as follows. Section 2 reviews the ORA-SS model. Section 3 describes the mapping table and the query allocation table. Section 4 gives the details of the proposed query rewriting algorithm. Section 5 compares our approach with related work and we conclude in Section 6.

2 ORA-SS Model

Our rewriting algorithm employs the ORA-SS model which is a semantically rich data model designed for semistructured data [11]. This model distinguishes between objects, relationships and attributes. An object class in the ORA-SS model is similar to the concept of an entity type in an ER model. An object class may be related to other object classes through a relationship type. Attributes are properties, and may belong to an object class or a relationship type. An attribute of an object class or relationship type in an ORA-SS schema may be represented as an attribute or sub-element in XML document. The main difference between the ORA-SS model and the ER model is that the ORA-SS model has a tree-like structure, which is more suitable for XML data. The nesting of the objects is reflected directly in the ORA-SS model. Other concepts that can be modeled in ORA-SS diagrams and not in ER diagrams include the ordering of elements and attributes, and fixed and default attribute values. An algorithm to translate XML schemas to the ORA-SS Schema Diagram is given in [3]. Note that user input may be needed to identify some semantics such as attributes of relationship types.

```
   <project jno="j01">
      <part pno="p01">
         <supplier sno="s01">
            <quantity>500</quantity>
         </supplier>
      </part>
</project>
```

Fig. 1(a). An XML document

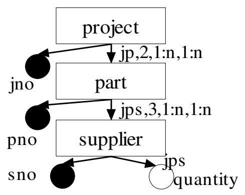


Fig. 1(b). ORA-SS schema of document in Fig. 1(a)

Fig. 1 shows an XML document and the corresponding ORA-SS schema diagram. Object classes "project" and "part" are denoted by labeled rectangles. The label "jps,3,1:n, 1:n" denotes a ternary relationship type "jps" involving object classes "project", "part" and "supplier", with parent cardinality 1:n and child cardinality 1:n.

That is, parts in a project may have one or more suppliers and a supplier can supply one or more parts to one or more project. Labeled circles denote attributes, and filled circles indicate identifiers. Attributes with labeled edges are relationship type attributes. For example, “jno” is an attribute of object class “project”, while “quantity” is an attribute of relationship type “jps”. Details on ORA-SS can be found in [11].

3 Mapping Table and Query Allocation Table

The proposed query rewriting algorithm utilizes two constructs: a mapping table and a query allocation table. A *mapping table* is created when an integrated schema is derived from the local schemas. This table contains the mappings from the integrated schema to the local schemas. We use the path-to-path mapping defined in [4]. The path from the root to the object class or attribute is captured in the mapping, so that one can tell the context of the object class or attribute. This will differentiate object classes and attributes with the same labels but different paths.

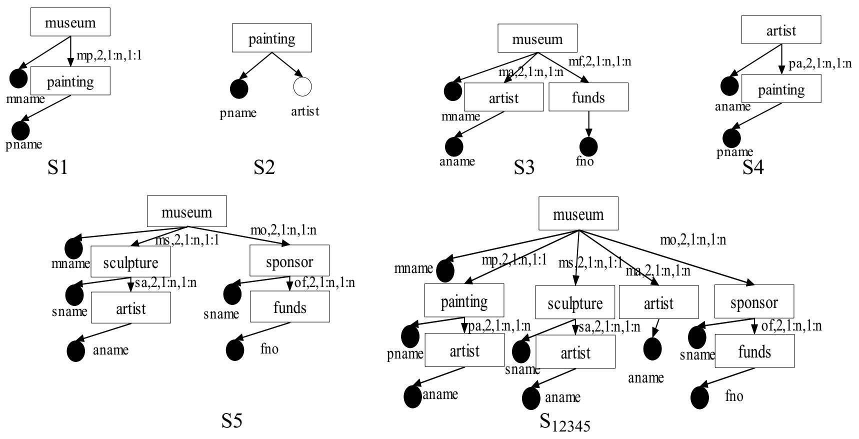


Fig. 2. S_{12345} is the integrated schema of local schemas S1, S2, S3, S4 and S5

Table 1. Mapping table for integrated schema S_{12345} in Fig. 2

Integrated schema	**Local schema**
S12345/museum	S1/museum, S3/museum, S5/museum
S12345/museum/mname	S1/museum/mname, S3/museum/mname, S5/museum/mname
S12345/museum/painting	S1/museum/painting, S2/painting, S4/artist/painting
S12345/museum/painting/pname	S1/museum/painting/pname, S2/painting/pname, S4/artist/painting/pname
S12345/museum/painting/artist	S2/painting/artist, S4/artist
S12345/museum/painting/artist/aname	S2/painting/artist, S4/artist/aname
S12345/museum/sponsor/funds	S3/museum/funds, S5/museum/sponsor/funds
…	…

Consider Fig. 2 where schema S_{12345} is an integration of the local schemas S1, S2, S3, S4, and S5. Table 1 shows a subset of the mapping table generated during the integration process. The first column of the mapping table gives the path from the root to each object class or attribute in the integrated schema; the second column shows the local schema id and the path to the equivalent object classes or attributes in the local schemas. When the mapping is not one-to-one, XQuery functions or user-defined functions are given in the second column

A query in XQuery format has two main parts: the first part contains the selection conditions, and the second part describes how the result is restructured. A *query allocation table* stores the selection condition paths and the return result paths of a query, as well as the local schemas where the data for these paths can be found. Details on the construction of the query allocation table are given in Section 4.1.

4 Query Rewriting

In this section, we will present our approach to rewrite a query on the integrated schema to query the local data sources. Partial information from various local data sources may need to be combined to produce the results of the user query. There are four steps in the proposed algorithm:

Step 1. Build the query allocation table.
Step 2. Group local schemas to form *join groups* that answer the user's query.
Step 3. Decompose user query to subqueries on the local sources.
Step 4. Compose subqueries for local schemas in a join group.

4.1 Build Query Allocation Table

A query allocation table (QAT) consists of a selection condition table and a return result table. The path of each selection condition and the return result is inserted into the selection condition table and the return result table respectively. The associated schemas identified from the mapping table are inserted into the corresponding rows. Two special cases need to be considered which can be treated as two rules.

Case 1: If a path corresponds to a branch in an ORA-SS schema with *n* (*n>1*) relationship types, it must be split into *n* subrows, one for each relationship type. Any attributes of an object class or a relationship type will appear in the row with their object class or relationship type.

Case 2: If a path contains "//" or "/*/" and does not contain any recursive relationship type, then the row that stores the original is retained and rows are created to store the expansion of each path. An expanded path that contains more than one relationship type is handled using Case 1. If "//" or "/*/" involves some recursive relationship type, then "//" or "/*/" will not need to be expanded.

Note that recursive relationship types are represented in the ORA-SS schema diagram by using reference arrow to point to some ancestor in the same path. These cases identify the relationship types involved in the query so that they can be handled properly and the results returned are expected and correct. This also highlights the advantage of using ORA-SS schema diagrams to distinguish between binary and n-ary relationship types and treat them properly in the algorithm. For example, n-ary relationship types should not be split into n-1 binary relationship types in the query allocation table.

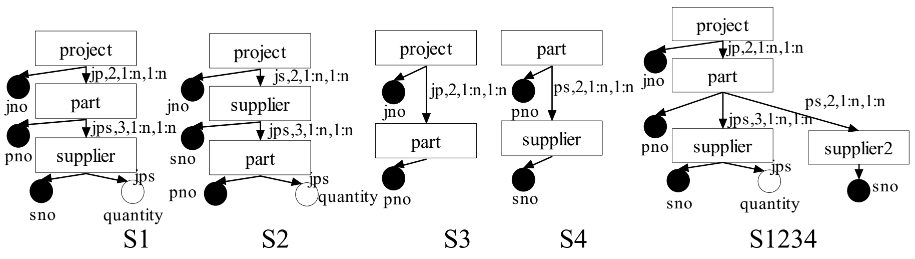


Fig. 3. S_{1234} is the integrated schema of S1, S2, S3 and S4

Example 1: Consider the schemas in Fig 3, where schema S_{1234} is an integrated schema of schemas S1, S2, S3, and S4. We issue query Q1 on the integrated schema to retrieve information about projects and their parts, and which supplier supplies the part to the project. Table 2 shows the query allocation table for query Q1. We note that the relationship type among project, part and supplier is a ternary relationship type. Hence, in the return result table, the path "/project/part/supplier" is not split into two paths. Since the local schema S4 does not model this ternary relationship type, it is not associated with this path. This prevents the retrieval of wrong results by joining the sources in S3 and S4.

```
Query Q1: for $j in /project
          return <project> {$j/jno}
              {for $p in $j/part
              return <part>{$p/pno}
                      {for $s in $p/supplier    return {$s}} </part>}
              </project>
```

Table 2. Query Allocation Table for Query Q1

Selection Condition Table: Empty
Return Result Table:

/project/jno	S1, S2, S3
/project/part/pno	S1, S2, S3
/project/part/supplier	S1, S2,

Example 2: Let us now **c**onsider Fig. 2, and the query Q2 on the integrated schema S_{12345}, which retrieves the names of artists that have works in a museum with name "field". The query allocation table is shown in Table. 3. The aim of QAT is to find the join groups. Since the rewritten queries will need to refer to the user query on the integrated schema, the QAT does not need to contain the details of selection

conditions such as “field” in Q2. Note /museum//aname is expanded into two XPath expressions /museum/painting/artist/aname and /museum/sculpture/artist/aname each of which are further split into two paths because of the binary relationship types. The path “/museum//aname” is retained and rows for each expansion of this path are inserted in the QAT.

Query Q2: for $m in /museum[mname=”field”],$a in distinct-values($m//aname) return <artist> {$a} </artist>

Table 3. Query Allocation Table for Query Q2

Selection Condition Table :

/museum/mname	S1, S3, S5

Return Result Table:

/museum//aname	S3
/museum/painting	S1
painting/artist/aname	S2, S4
/museum/sculpture	S5
sculpture/artist/aname	S5

4.2 Identify Local Sources to Answer User Query

Next, we need to determine which local schemas must be combined to get the expected results. These groups of local schemas are called *join groups*. The local schemas in each join group must contain all the paths required for the selection condition and must have at least one path for the result.

Algorithm GenerateJoinGroups scans the query allocation table (QAT) to find the join groups. Lines 1-5 create an ordering on the local schemas based on the rows in which they first occur in the QAT and store the ordered list in *lt*. A local schema is low in the ordering if it first occurs in the top row and high in the ordering if it first occurs in the bottom row of the QAT. Lines 6-31 use a stack to find the join groups. The local schemas are considered based on the ordering in the list *lt* from lowest to highest. Initially the lowest local schema is pushed onto the stack, and the next schema to be pushed onto the stack is the next lowest that occurs in a different row. When the schemas on the stack cover all the selection condition paths in the QAT, we output them as a join group. The top schema is popped off the stack, and the algorithm goes on to find the next schema which could contribute to the user query. The algorithm scans the schemas in the order of *lt*, so there is no duplication or missing join groups.

Example 3: Consider the schemas in Fig. 4. The attribute “location” in S_{12345} is a combination of the attributes “address” and “postal code” in S5. The query Q3 retrieves the year and title of the books that were written by “Tom” in the year “2000”. The corresponding query allocation table is shown in Table 4.

Algorithm GenerateJoinGroups first looks at the first row “/book/author” in the Selection Condition Table, and adds S1, S2, S3 in the list *lt*. Then it checks the second row “/book/year”, and adds S4 in the list *lt*. Thus, the *lt* has local schema order as S1, S2, S3, and S4. After the order is computed, S1 is first pushed on the stack, and S2 is then considered. Since it does not add any extra paths, it is not pushed on the stack. S3 is considered and because it does cover extra paths, it is pushed on the stack. Together S1 and S3 cover all the path information in the QAT, so {S1, S3} is output as a join group. S3 is then popped off the stack, S4 is considered. Together S1 and S4 cover all the path information, and {S1, S4} is output as a join group. {S2, S4} and {S3} are output after that.

Note that {S2, S3} is not a join group, because although they cover all the path information in the selection condition table of the QAT, S2 does not cover any more path information that S3 does not cover and consequently would not add new answers to the result of the query. Note that {S3} is a join group, even though {S1, S3} is also a join group. The result from the rewritten query in {S1, S3} can return the result as Q2, while {S3} can return the partial result which has missing information of the title of book.

The final result is found by taking the union of all the answers from the different join groups. Given that the relationship type information is captured in the ORA-SS model, the union can be based on the relationship type information. For each relationship type, we take the deep union [23], that is, we take the union of the objects if and only if all of their ancestors are the same.

```
Algorithm GenerateJoinGroups
  Input: Query allocation table qat;
  Output: join groups
1. create an empty list lt;
2. for i=1 to num_of_row of qat
3.     for j=1 to num_of_schema_id of row i
4.          if schemaij is present in the rowi and not
in list lt
5.            add schemaij to list lt;
6. n=the number of local sources in qat;
7. create an empty stack st;
8. for i=1 to n from lt
9. {
10.    if schemai is not in the top row in qat
11. break;
12.    push schemai on the stack st;
13.    if schemai is present in all rows of qat
14.    {
15. Output {schemai};
16. st=null;
17. continue;
18.    }
19.    for j=i+1 to n if schemaj occurs in the rows,
which the other schemas in st do not occur in, and
schemaj does not occur in all the rows that the top
element of st occurs in
20.    {
21. push schemaj on the stack st;
22. if (the local schemas in st has included all the
path information in qat)
23. {
```

```
24.          output all the schemas in the stack st
split by”,” in a “{}”;
25.     pop the top schema off the stack st;
26. }
27.    }
28.    if (j= =n and st has included all the path in-
formation of the selection condition table and at least
one result in return result table)
29. output all the schemas in the stack st split by”,”
in a “{}”;
30.    st=null;
31.}
```

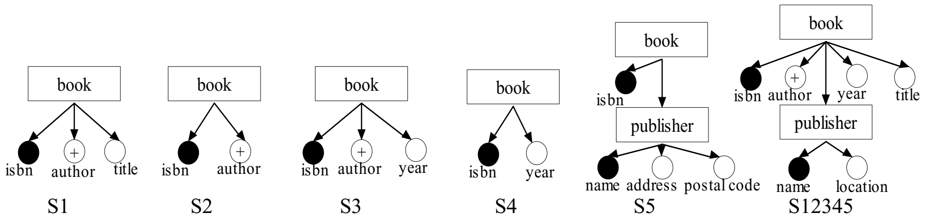


Fig. 4. S_{12345} is the integrated schema of local schemas S1, S2, S3, S4, S5

Query Q3: for $b in /book where $b/author=”Tom” and $b/year=”2000” return <result> {$b/year/text()} {$b/title/text()} </result>

Table 4. Query Allocation Table for Query Q3

Selection Condition Table:

/book/author	S1, S2, S3
/book/year	S3, S4

Return Result Table:

/book/year	S3, S4
/book/title	S1

4.3 Decompose User Query to Subqueries on Local Sources

This step decomposes the user query into queries on the local schema based on the join groups. Subqueries are composed to compute the answers in the same join group in Step 4. Hence, in addition to retrieving the data required by the user query, we also need the data necessary to join the parts of the answers from different local schemas together. We call the classes that are necessary for joining the parts of answers as *join* *object classes*. The key of the join object class is used for testing the equivalence when joining the subqueries.

A join object class depends on the semantics of the schema. We have 3 cases:

Case 1: For a join group, if there are *n* paths in the QAT from different local schemas with a common ancestor in the user query, then the least common ancestor in the user query is a join object class. An object class O is the least common ancestor of paths P1 and P2, if O is an object class that occurs in both P1 and P2, and O does not have any descendant object class that also occurs in P1 and P2.

Case 2: For a join group, if the paths in the QAT are from different local schemas, and there is an object class that is the end of one path and the start of the other path, then this intermediate object class is a join object class.

Case 3: For a join group, if two attributes of the same relationship type in a user query are from different local schemas, then all the object classes involved in this relationship type are join object classes.

Example 4: Recall Example 3 and the join group {S1, S3}. S1 provides "/book/title", "/book/author" and S3 provides "/book/year", "/book/author". To answer the query Q3, the subqueries from S1 and S3 need to be composed using the key of their least common ancestor i.e. the key "isbn" of the join object class "book".

We first consider the case where the local schemas are projections of the integrated schema. The rewritten query for a local schema will effectively be a projection of the user query with the join object class identifier included in the *return* part of the rewritten query. The rewritten query can be derived as follows:

1. For every path in the *let* part, *for* part, *where* part and *return* part of the user query, retain the path if it exists in the local schema.
2. Add the path to any join object class identifiers that are relevant to this local schema in the join group being considered.

When the local schemas are not projections of the integrated schema, the query will need to be rewritten based on the local schema structure. We will first describe how to rewrite a user query for a local schema where the subquery on the local schema returns only one object class or attribute. Then we discuss how to rewrite a user query for a local schema where the subquery on the local schema returns many object classes or attributes.

4.3.1 Subquery Returns Only One Object Class or Attribute.

We have two cases. The first case is for queries involving one object class or attribute, while the second case is for queries involving more than one object class.

Case A1. Queries involve one object class or attribute

An object class in an integrated schema can originate from either an object class or an attribute in a local schema, or it can be derived from object classes and attributes in one local schema.

Case A1-i. *Integrated object class originates from a source object class.*

When an integrated object class is mapped to an equivalent object class from a local schema, but the path from the root to the equivalent object class is different, variable bindings in the *for* clause or *let* clause are changed according to the mapping table that specifies the path of the equivalent source object class.

Example 5: Consider the schemas in Fig. 2. Query Q4 on the integrated schema S_{12345} retrieves all the information on the object class "funds", which is in path "/museum/sponsor/funds":

```
Query Q4: for $f  in /museum/sponsor/funds
                   return  <result> {$f} </result>
```

Based on the mapping table, we have S_{12345}/museum/sponsor/funds: S3/museum/funds, S5/museum/sponsor/funds. This indicates that the query can be rewritten to query local sources S3 and S5. The rewritten query on source S5 will be the same as Q4, while the query on S3 will be as follows:

```
Query Q4_S3: for $f in /museum/funds
                      return  <result> {$f} </result>
```

Case A1-ii. *Integrated object class originates from an attribute.*

An object class can also originate from an attribute, because a concept can be expressed as an attribute in one schema, and as an object class in another schema. When rewriting such queries, variable bindings in the *for* clause or *let* clause are changed according to the mapping table that specifies the path of the equivalent attribute; the equivalent object class is created in the *return* clause with the attribute as an attribute of this object class.

Example 6: The following query is on the integrated schema S_{12345} of Fig. 2. Query Q 5 retrieves the information of artists of the painting with pname "hero".

```
Query Q5:  for $p in /museum/painting
                 where $p/pname="hero"
                 return  <result> {$p/artist} </result>
```

Query Q5 will be rewritten for S2 and S4. Since schema S2 (see Fig. 2) models "artist" as an attribute of the object class "painting", Query Q5_S2 will compute the information for artist on local schema S2:

```
Query Q5_S2:  for $p in /painting
                      where $p/pname="hero"
                         return <result> <artist> <aname> {$p/artist/text()} </aname>
                      </artist>  </result>
```

Case A1-iii. *Integrated object class or attribute originates from a set of object classes (attributes) or vice versa.*

When one object class (attribute) in the integrated schema is the combination of many object classes (attributes) of another local schema or vice versa, XQuery or user-defined functions can be used to substitute the path in the user query.

Example 7: Consider the schemas in Fig. 4. Query Q6 retrieves the publisher location of the book with isbn "7-5053-4849-3/TP.2370" on the integrated schema S_{12345}:

```
Query Q6: for $b in /book
                 where  $b/isbn="7-5053-4849-3/TP.2370"
                 return <result>{$b/publisher/location}</result>
```

Q6 will be rewritten on S5. The mapping in the mapping table shows that S12345/book/publisher/location:string-join((S5/book/publisher/address/text(), S5/ book/publisher/postalcode/text()),“ ”). We assume that the attribute “location” is expressed by the address followed by a space and the postal code. The query on S5 is shown in Query Q6_S5. It combines the address and postal code by the XQuery functions from the mapping table. The rewritten query on S5 will be:

Query Q6_S5: for $b in /book
where $b/isbn=”7-5053-4849-3/TP.2370”
return <result> <location> {string-join(($b/publiser/address/text(),
$b/publisher/postalcode/text()),” ”)}</location> </result>

Case A2. Query involves more than one object classes

When the number of object classes in the query path is more than one, we need to consider the structural relationship type between the object classes. There are two cases: (1) object classes are swapped in the integrated schema, and (2) siblings in a local schema are mapped to ancestor and descendent in the integrated schema.

Case A2-i. When object classes in the integrated schema are swapped in the hierarchy compared to the local schema, the path in the subquery needs to be rewritten based on the path of the local schemas.

Example 8: The following query on the integrated schema S_{12345} in Fig. 2 retrieves all the “museum” which have the paintings by artist “David”.

Query Q7: for $m in /museum where $m/painting/artist/aname=”David”
return<museum>{$m/mname/text()}</museum>

The join groups are {S1, S2} and {S1, S4}. In join group {S1, S4}, the join object class is painting for S4. The projection subquery on S4 is:

Query Q7_S4’: for $p in /painting where $p/artist/aname=”David”
return<painting>{$p/pname}</painting>

The path expression in the *where* clauses are changed to the corresponding object class (attributes) by using /../. The rewritten query on S4 is:

Query Q7_S4: for $p in/artist/painting where $p/../aname=”David”
return <painting>{$p/pname}</painting>

This query needs to be joined with the subquery for S1 to get the final result.

Case A2-ii. When two object classes have an ancestor-descendant relationship type in the integrated schema, but they are siblings in the local schema, then the least common ancestor of these object classes must be used as binding variables to connect them. The related path in the where and return clause must be revised based on the structure of the local schemas.

Example 9: In Fig. 5, students work for projects, and students have their labs. The lab also has coordinators. Consider the query Q8 on the integrated schema S123, which retrieves a project lab coordinator where pno is “p01”.

Query Q8: for $p in /project where $p/@pno=”p01”
return <result>{$p/student/lab/coordinator}</result>

The join groups are {S1, S3} and {S2, S3}. The return clause in Q8 shows that the query path is from $p to lab. In order to rewrite the query for schema S1, the algorithm looks for the nearest ancestor node that is common to both project and lab. Student is then bound to the variable in the *for* clause as follows:

Query Q8_S1: for $s in /student where $s/project/@pno="p01"
return <result>{$s/lab/@lno}</result>

This query needs to join with the subquery for S3 to get the final result.

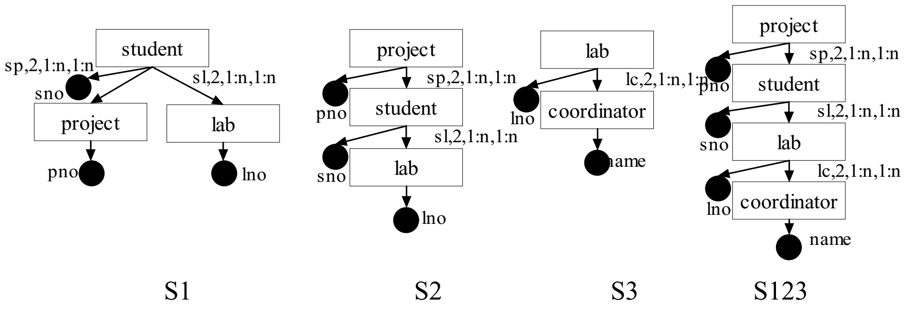


Fig. 5. S_{123} is the integrated schema of local schemas S1, S2, S3

4.3.2 Subquery Returns Many Object Classes or Attributes

Chen et al. in [3] introduce an algorithm for the automatic generation of XQuery view definitions for ORA-SS views, focusing on the view definitions for hierarchical structures of XML. Due to space limitations we do not cover this case in this paper except to note that their algorithm can be used to rewrite such queries.

4.4 Compose Subqueries for Join Group

When joining subqueries on local schemas in the same join group, the identifier of the join object classes must be tested for equivalence.

We start by considering the basic case where the same object attributes are from different local schemas. To compose subqueries from these local schemas in join groups, the clauses *for*, *where*, and *return* are combined together with the join condition equivalence test inserted in the *where* clause.

We allow the return results to have missing information. The parent object will not be removed from the return result if it has a missing child. For each return object or attribute, the join equivalence condition test related to this return object or attribute is nested in the appropriate part of the query.

Example 10: Consider the schemas in Fig. 4. and query Q9 that retrieves year and title of the books that were written by "Tom" in year "2000" and retrieves the publisher name if the book's publisher location is Singapore.

Query Q9: for $b in /book where $b/author="Tom" and $b/year="2000"
return<result>{$b/year/text()} {$b/title/text()}{
for $p in $b/publisher
where contains ($b/publisher/location/text(),"Singapore")

```
                    return<publisher> {$b/publisher/name} </publisher> }
                   </result>
```

The join groups are {S1, S3, S5}, {S1, S4, S5}, {S2, S4, S5} and {S3, S5}. We show the query example for the join group {S1, S3, S5}. The user query is decomposed into subqueries on the local schemas S1, S3, and S5. The join object class is "book" for these local schemas. The subqueries on S1, S3 and S5 are shown below:

```
Query Q9_S1:  for $b in /book
                where $b/author="Tom"
               return <result> {$b/isbn/text()} {$b/title/text()} </result>
Query Q9_S3: for $b in /book
               where $b/author="Tom" and $b/year="2000"
                return <result> {$b/isbn/text()} {$b/year/text()} </result>
Query Q9_S5: for $b in /book
               where contains ($b/publisher/address/text(),"Singapore")
                  return<result>{$b/isbn/text()}
             <publisher>{$b/publisher/name} </publisher></result>
```

The composition of the subqueries for local schemas S1, S3 and S5 are as follows:

```
for $b1 in doc("S1.xml")/book, $b3 in doc("S3.xml")/book
where $b1/author="Tom" and $b3/author="Tom" and $b3/year="2000"
and $b1/isbn=$b3/isbn
return <result>{$b3/year/text()} {$b1/title/text()}
                  {for $b5 in doc("S5.xml")/book
                  where contains ($b5/publisher/address/text(),"Singapore") and
                  $b5/isbn=$b1/isbn
                       return<publisher>
                   {$b5/publisher/name}<publisher>}</result>
```

Note that although the join object class for S1, S3 and S5 is book, the equivalence tests are on separate lines in the rewritten query. This is because we allow parent information to be returned even when a child object class is missing.

5 Comparison with Related Work

Amman et al. in [1] propose a mediator architecture for querying and integrating XML data sources. Their global schema is described as an ontology, which is expressed in a light weight conceptual model. Similar to our algorithm, their method also finds join groups, where the local sources of the join groups can together compute the results for the user query. However, the limitation in [1] is that a query cannot return nested structures.

Lakshmanan and Sadri in [8] propose an infrastructure for interoperability among XML data sources. Mapping rules are created to map the items in local schemas to a common vocabulary. They also address the query processing and optimization in the system. For query processing, they differentiate between inter-source query and intrasource query, which query across local schemas and within one local schema respectively. Consistency conditions are used to optimize inter-source queries. One limitation of this work is that when results from local schemas are joined, the join variable is limited to the lowest common ancestor of nodes.

Yu and Popa in [22] introduce an algorithm for answering queries via a target schema. The algorithm uses target constraints that are used to express data merging rules. The mappings from the integrated schema and local schemas are tree to tree. Generating such mappings is expensive, especially when the XML sources are complicated.

The models that are utilized in the works [1, 8, 22] cannot specify whether a relationship type is binary or n-ary and do not distinguish between attributes of object classes and attributes of relationship types from the local XML sources. The lack of such semantic information may lead to the retrieval of wrong results as the following example illustrates.

Example 11: Recall Example 1 where only S1 and S2 will be considered for the query Q1. Since the works in [1, 8, 22] cannot distinguish between binary or n-ary relationship types, they will join the sources from S3 and S4 to get the result, which is not correct for the user query. The example below highlights the problem for the attributes and n-ary realtionship. For simplicity, schemas S3 and S4 are omitted here. Let the data source for S1 be X1, and the data source for S2 be X2 as shown in Table 5. Table 6 shows the results for query Q1 that are retrieved by our algorithm and the methods in [1, 8, 22].

We observe that the results returned by the query rewriting method in [1, 8, 22] contain the project with jno "j01" has part "p01", which is supplied by suppliers with sno "s01" and "s02". This violates the local data sources X1 and X2, where the project with jno "j01" has part "p01" is only supplied by suppliers with sno "s01". This is because the methods in [1, 8, 22] treat the relationship type between part and supplier as a binary relationship type, instead of the intended ternary relationship type involving project, part, and supplier. They treat the quantity as the attribute of part in S2, so when they find the part with pno "p01" has quantity "100" in X1, and has quantity "200" in X2, they will combine them to make the final result. This leads to the wrong answer returned. In contrast, our algorithm takes the XML hierarchy structure into consideration and retrieves the correct answers.

Table 5. Data sources for S1 and S2 in Fig. 3

X1:	X2:
<project jno="j01">	<project jno="j02">
<part pno="p01">	<supplier sno="s02">
<supplier sno="s01">	<part pno="p01">
<quantity> 100 </quantity>	<quantity> 200 </quantity>
</supplier>	</part>
</part>	</supplier>
</project>	</project>

To summarize, our algorithm differs from existing works in the following ways:

1. We treat binary and n-ary relationship types differently. Treating an n-ary relationship type as n-1 binary relationship types gives wrong results.
2. We treat attributes of object classes and attributes of relationship types differently in the QAT and when we compose the sub queries of the local sources.
3. Our algorithm takes the XML hierarchy structure into consideration when doing the rewriting.

Table 6. Results retrieved by our algorithm and [1, 8, 22]

Results obtained by our proposed algorithm	**Result obtained by [1, 8, 22]**
`<result>`	`<result>`
`<project jno=”j01”>`	`<project jno=”j01”>`
`<part pno=”p01”>`	`<part pno=”p01”>`
`<supplier sno=”s01”>`	`<supplier sno=”s01”>`
`<quantity> 100 </quantity>`	`<quantity> 100 </quantity>`
`</supplier>`	`</supplier>`
`</part>`	`<supplier sno=”s02”>`
`</project>`	`<quantity> 200 </quantity>`
`<project jno=”j02”>`	`</supplier>`
`<part pno=”p01”>`	`</part>`
`<supplier sno=”s02”>`	`</project>`
`<quantity> 200 </quantity>`	`<project jno=”j02”>`
`</supplier>`	`<part pno=”p01”>`
`</part>`	`<supplier sno=”s01”>`
`</project>`	`<quantity> 100 </quantity>`
`</result>`	`</supplier>`
	`<supplier sno=”s02”>`
	`<quantity> 200 </quantity>`
	`</supplier>`
	`</part>`
	`</project>`
	`</result>`

6 Conclusion and Future Work

In this paper, we have introduced a semantic approach to rewrite queries for semistructured data integration. A user’s queries on the integrated schema are rewritten to query the local sources. When XML repositories are integrated there may be semantics that are not expressed explicitly, and without the necessary semantics it is possible to misinterpret the meaning of the data and combine the results from different local schemas to give unexpected results. Our algorithm uses the ORA-SS model to describe the schemas of the local data sources and the integrated schemas. This allows us to distinguish between binary and n-ary relationship types, attributes of object classes and attributes of relationship types, and in turn treat these cases differently in our rewriting algorithm.

References

1. B. Amann, C. Beeri, I. Fundulaki, M. Scholl. Querying XML sources using an Ontology-based Mediator. CoopIS, 2002.
2. P. Buneman, S. Davidson, W. Fan, C. Hara, W.C. Tan. Keys for XML. WWW Conference, 2001.
3. Y. Chen, T.W. Ling, M.L. Lee. Automatic Generation of XQuery View Definitions from ORA-SS Views. ER, 2003.
4. S. Cluet, P. Veltri, D. Vodislav. Views in a Large Scale XML Repository. VLDB, 2001.
5. O.M. Duschka, M.R. Genesereth. Answering Recursive Queries Using Views. ACM PODS, 1997.
6. L. Haas, D. Kossmann, E. Wimmers, J. Yang. Optimizing queries across diverse data sources. VLDB, 1997.
7. A. Halevy. Theory of Answering Queries Using Views. ACM SIGMOD Record 29(4), 2000.
8. L.V.S. Lakshmanan, F. Sadri. Interoperability on XML Data. ICSW, 2003.
9. M.L. Lee, T.W. Ling, W.L. Low. Designing Functional Dependencies for XML. EDBT, 2002.
10. A. Levy. Logic-Based Techniques in Data Integration. Logic based artificial intelligence, 1999.
11. T.W. Ling, M.L. Lee, G. Dobbie. Semistructured Database Design, ISBN: 0-387-23567-1, Springer, 2005.
12. I. Manolescu, D. Florescu, D. Kossman. Answering XML queries over heterogeneous data sources. VLDB, 2001.
13. H. Garcia-Molina, Y. Papakonstantinou, D. Quass, et al. The TSIMMIS project: Integration of heterogeneous information sources. Journal of Intelligent Information Systems, 1997.
14. K. Passi, E. Chaudhry. A Global-to-Local Rewriting Querying Mechanism using Semantic Mapping for XML Schema Integration. ODBASE 2003.
15. K. Passi, L. Lane, S.Madria, Bipin C. Sakamuri, M. Mohania, S. Bhowmick. A Model for XML Schema Integration. EC-Web, 2002.
16. R. Pottinger, A. Levy. A Scalable Algorithm for Answering Queries Using Views. VLDB, 2000.
17. M. Stonebraker. Implementation of integrity constraints and views by query modification. ACM SIGMOD, 1975.
18. Xyleme. A dynamic warehouse for XML Data of the Web. IEEE Data Engineering Bulletin, 2001.
19. H.Z. Yang, P.A. Larson. Query Transformation for PSJ-queries. VLDB, 1987.
20. X. Yang. Global Schema Generation and Query Rewriting In XML Integration. MSc thesis, National University of Singapore, 2005.
21. X. Yang, M.L. Lee, T.W. Ling. Resolving Structural Conflicts in the Integration of XML Schemas: A Semantic Approach. ER 2003.
22. C. Yu, L. Popa. Constraint-Based XML Query Rewriting for Data Integration. SIGMOD 2004.
23. P. Buneman, A Deutsch, W.C. Tan. A deterministic model for semistructured data. Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, 1998.

A Taxonomy of Inaccurate Summaries and Their Management in OLAP Systems

John Horner and Il-Yeol Song

College of Information Science and Technology, Drexel University,
Philadelphia, PA 19104
(jh38, song)@drexel.edu

Abstract. Accurate summarizability is an important property in OLAP systems because inaccurate summaries can result in poor decisions. Furthermore, it is important to understand and identify the potential sources of inaccurate summaries. In this paper, we present a taxonomy of inaccurate summary factors and practical rules for handling them. We consolidate relevant terms and concepts in statistical databases with those in OLAP systems and explore factors that are important for measuring the impact of erroneous summaries. We discuss these issues from the perspectives of schema, data, and computation. This paper contributes to a comprehensive understanding of summarizability and its impact on decision-making. Our work could help designers and users of OLAP systems reduce unnecessary constraints caused by imposing rules to eliminate all summarizability violations and give designers a means to prioritize problems.

1 Introduction

Data warehouses contain large sets of subject oriented, integrated, historical, and relatively static data used for strategic decision making. Because data warehouses are typically magnitudes larger than operational systems, they typically contain many aggregate summaries of base data. Thus, accurate summaries are necessary to ensure that the decisions based on them are sound.

Summarizability refers to the property of whether performing an aggregate operation will result in an accurate result. Martyn [1] describes three design criteria necessary for all database systems, consisting of correctness, efficiency, and usability, and argues that data correctness is of utmost importance. Lenz and Shoshani [2] also argue that summarizability in online analytic processing (OLAP) is an important property because violating this condition can result in erroneous conclusions and decisions. Shoshani [3] further argues that it is important for OLAP systems to borrow some areas of statistical database systems, such as strict classification hierarchies and the distinction between summary data from attributes.

Prior research in both statistical databases and OLAP has explored numerous conditions that could result in erroneous summarizations. However, the issues are dispersed throughout the literature in OLAP systems and statistical databases and have not been consolidated into a comprehensive taxonomy of issues that could result in inaccurate summaries. These analyses have focused on identifying the existence of erroneous summaries, rather than the impact on decision making. We note that the type of query and proximity of the summary outputs to decision points also affect the impact of inaccurate summaries. Furthermore, not all issues can be eliminated and methods for eliminating the problems can restrict number and types of queries that can be entered. It is therefore important to identify the factors that could lead to inaccurate summaries and approaches to manage them.

In [4], we analyze various causes of non- and semi- additive data in OLAP systems, and suggest rules for identifying and managing these data. This paper expands our earlier work on additivity and look beyond simply identifying the existence of aggregate summary problems.

In this paper, we present a taxonomy of inaccurate summary factors and practical rules for handling them. The primary contributions of this paper are as follows: first, our taxonomy of inaccurate summaries is comprehensive in that (1) we cover them from the perspectives of schema, data, and computation, and (2) we consolidate relevant terms and concepts in statistical databases with those in OLAP systems. Second, we suggest metadata that can be used to identify schema, data, and computational problems and suggest how to use this metadata to detect the impact that an invalid summary may have on a decision. Third, we present practical rules that can be used to quickly identify problems that have the potential to impact decisions:

Our paper is organized as follows: Section 2 describes relevant literature. Section 3 details a comprehensive taxonomy of summarizability issues in OLAP systems. Section 4 examines how these issues influence decision-making. Section 5 suggests techniques for managing summarization problems. Finally, Section 6 concludes our paper.

2 Background and Related Literature

Data warehouses are typically conceptualized as facts and dimensions, whereby facts are measures of interest, and dimensions are attributes used to browse, select, group, and aggregate measures of fact tables. Attributes that are used to aggregate measures are labeled classification attributes, and are typically conceptualized as hierarchies. An example of a classification hierarchy is the time dimension, upon which measures can be aggregated from the lowest level of granularity, dates, into progressive higher months, quarters, and years. For example, a profit measure may be aggregated from the daily profit to the monthly, quarterly, or yearly profit.

Typically, data is aggregated along multiple hierarchies, summarizing data along multiple dimensions. For example, a summary may show the total sales in the year 2004 at all branch locations in Pennsylvania. In this case, the sales measure is rolled up along the time dimension and location dimension. Because of the enormous size of the data sources, operations are performed to summarize measures in a meaningful way. The typical OLAP operations include Roll-up, Drill-down, Slice, Dice, Pivoting, and Merging.

Data are most commonly aggregated using the SUM operator in OLAP systems [5]. Measures can be classified based on whether they can be meaningfully added along hierarchies in various dimensions. Specifically, measures are classified as non-additive, semi-additive, or fully-additive, whereby a measure is:

- **fully-additive** if it is additive across all dimensions;
- **semi-additive** if it is only additive across *certain* dimensions; and,
- **non-additive** if it is not additive across any dimension.

Previous research on summarizability has focused on three primary areas. The first is identifying problems that could lead to summarizability problems [2, 3, 4]. The second focus is on defining methods for eliminating issues that could result in inaccurate summaries [6, 7]. The final focus is on making these problems visible through conceptual models [7, 8, 12].

Classifying measures based on the number of dimensions along which they can be aggregated is useful for making visible where inaccuracies may occur. However, this classification scheme does not give insight into the reason why measures are not additive, nor does it focus on problems that could result using other aggregate operators. In our previous work [4], we analyzed the reasons why certain attributes were not additive along certain dimensions, and distinguished between temporally and categorically semi-additive measures.

Lenz and Shoshani [2] take a broader look at the problem of erroneous summaries, and identify issues that are applicable to various aggregate operations. They describe three necessary conditions for summarizability, including disjointness, completeness, and type compatibility. Lehner, Albrecht, and Wedekind [6] suggest normal forms for multi-dimensional databases that can be used to guarantee summarizability. The focus of their research is to ensure that a broad range of schemas can be designed to meet both the completeness and disjointedness summarizability conditions specified by Lenz and Shoshani [2]. Hüsemann, Lechtenbörger, and Vossen [7] also suggest an approach for designing data warehouses that avoids aggregation anomalies. The approaches of both Lehner et al. and Hüsemann et al. focus on eliminating all possible aggregation anomalies through normal forms.

Tryfona et al. [8] suggest incorporating summary properties into the conceptual modeling of data warehouses. Specifically, they note that measures should be classified as *stock*, *flow*, or *value per unit* because different properties behave differently with different summary functions. Additionally, they note that object properties, including strictness and completeness should also be modeled. By incorporating this information into the model, potential problems can be made apparent at the conceptual level.

Shoshani [3] notes that OLAP systems and statistical databases are quite similar, and compares the work done in both areas. He argues that data warehouses should include a statistical object data type. Furthermore, he states that this statistical object data type should support the semantics, operations, and physical structure of the multi-dimensional space, and must also manage metadata of the category values and hierarchical associations.

In our paper, we expand upon this research by creating a comprehensive taxonomy of issues that could result in summarizability violations.

3 Taxonomy of Inaccurate Summaries

In OLAP systems, inaccurate summaries have typically been categorized based on both the number of hierarchies that measures could be aggregated, as is the case with labeling measures as additive, semi-additive, non-additive. In the statistical database (SDB) community, the terms *Flow*, *Stock*, and *Value per Unit* are used to classify summarizability problems. There are similarities among the problems identified in both the SDB and OLAP communities. We clarify the relationships between the terms used to identify problems in OLAP systems with those in statistical databases in Table 1.

Table 1. Comparison of summarizability terms used in OLAP and Statistical databases

SDB Label	OLAP Label	Description	Examples
Flow	Type of Fully Additive Snapshot	Total of transactions that occurred during the time between snapshots	*Change* in account balance during period
			Change in population during period
Stock	*Temporally* Non-Additive Snapshot	The level recorded at the end of a snapshot period	Account balance at specific point in time
			Population snapshot at specific point in time
Value per Unit	Type of Non-Additive Fraction	A fraction or rate used in expressions to convert units.	Exchange rates, which are used to convert different units of financial currency
			Cancer rate per million people, which can be used to convert population to number of cancer cases

In addition to the problems described above, the utility of a summary is also partially dependant on the extent of missing, biased, and inaccurate data, the type and purpose of queries, the visibility of any problems, and the content of the data. Therefore, we present a comprehensive taxonomy that differentiates structural issues, data issues, and computation issues. This taxonomy is intended to illuminate properties of the schema, data, and aggregate operator that could lead to erroneous summaries.

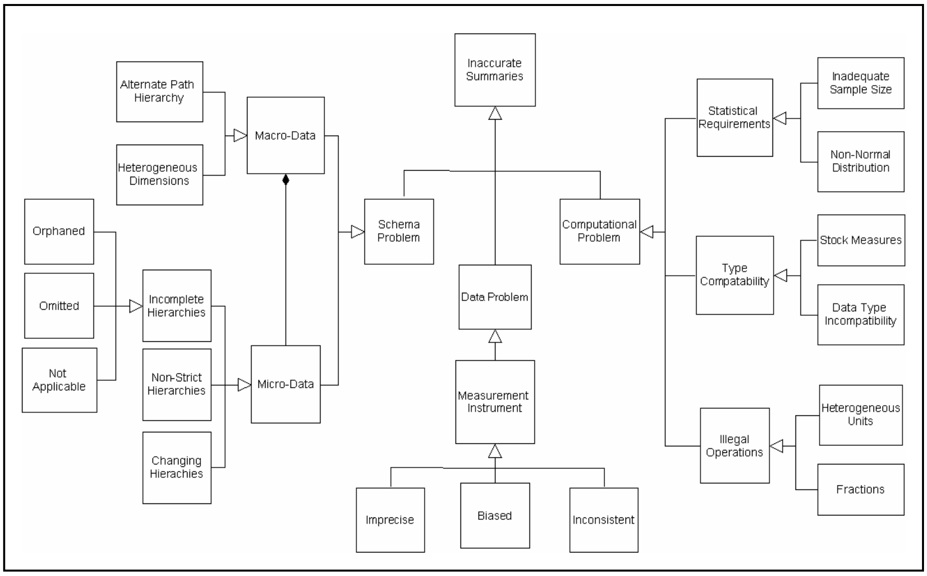


Fig. 1. The Taxonomy of inaccurate summaries in a UML class diagram

At the highest level, we differentiate three primary causes of aggregation problems: schema, data, and computation. *Inaccuracies due to schema* refer to those problems that are associated with the dimensional hierarchy, including non-strict and incomplete hierarchies, multiple path hierarchies, and heterogeneous dimensions. *Data-based inaccuracies* refer to problems related with the specific data instances, including changing data, inaccurate data, and imprecise or changing measurement

Table 2. Taxonomy of Potential Summarization Problems

Level 1	**Level 2**	**Level 3**	**Description**
Schema	Micro-Level	Non-strict	Hierarchy member has more than one parent
		Incomplete	*Orphaned*: Lower level hierarchy members do not have a parent in a higher level of a hierarchy
			Omitted: Null because real world data are not captured by the system
			Not Applicable: Null because there is no applicable value
		Changed	Hierarchy member splits into two members, merges into one, or moves from one parent to another.
	Macro-Level	Multiple Path Hierarchies	Lower objects in hierarchy have more than one aggregation path at higher levels
		Heterogeneous Dimensions	In schemas with multiple fact tables, the dimensions are not shared or exactly the same, but share a semantic relationship
Data	Measurement Instrument	Imprecise	Measurement instruments have inherent error associated with them
		Biased	Measurement instruments may capture data that are persistently higher or lower than the actual values
		Inconsistent	Aggregating data captured using different measurement instruments can result in erroneous summaries
Computation	Illegal Operations	Units (also referred to as categorically non-additive)	Cannot meaningfully aggregate data that have different units
		Fractions	Cannot meaningfully average measures derived from fractions, rather the numerator and denominator must be aggregated separately.
	Type Compatibility	Stock (Also referred to as temporally non-additive)	Stock levels (snapshot levels) are stored instead of flow (change over time) data; includes measures that represent averages, maximums, and minimums
		Data Type	Cannot use certain aggregate operators with some data types
	Statistical Requirements	Sample Size	Some operations require minimal sample sizes to be significant
		Distribution	Certain operators are more appropriate for normally distributed data, while other operators are more appropriate for non-normally distributed

instruments. And, *computational inaccuracies* refer to problems related to inappropriately computing aggregates summaries, such as those that could result from summing measures of intensity, summing data snapshots, or using the mean to find the central tendency of log-normally distributed data. Figure 1 summarizes the taxonomy in a UML class diagram and Table 2 depicts our taxonomy of potential summarization problems.

3.1 Schema Level

Inaccurate summaries can result when the structure of the classification hierarchies does not meet certain necessary conditions. Specifically, problems can result from non-strict and incomplete hierarchies, and can occur at the level of either micro-data or macro-data. Micro-data refers to base data, while macro-data refers to schema objects.

A strict hierarchy refers to a classification hierarchy whereby each object at a lower level belongs to only one value at a higher level. *Non-strict hierarchies* can be thought of as many-to-many relationships between a higher level of a hierarchy and a lower level. Lenz and Shoshani [2] refer to strict hierarchies as disjointed, and note that disjointedness of category-attributes is a necessary condition for summarizability. In order to test for disjointedness, or hierarchy strictness, it is necessary to examine the semantic knowledge of the micro-data or test the actual data. They describe students being assigned to a single department as an example of disjointedness; whereas, if students could be assigned to multiple departments, the disjointedness property would be violated. The non-strict hierarchy in Figure 2 depicts a situation where at the schema level, the student object rolls up only to one higher level object, Department, but at the micro-data level, an individual student could be assigned to more than one department. If each student has values for *tuition_paid* associated with them, then the payments associated with these students may be counted more than once.

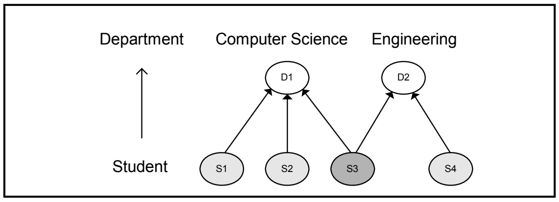


Fig. 2. Non-strict Hierarchy

In order to ensure that dimensional hierarchies are complete, it is necessary that they satisfy two conditions. All lower level members must belong to one higher level object, and that object must consist of those members only. We will refer to these different types of incomplete hierarchies as *Orphaned-incomplete*, which will include hierarchies where lower level records are stored, but are not associated with parents. *Omitted-incomplete* include hierarchies where records are not stored in the database.

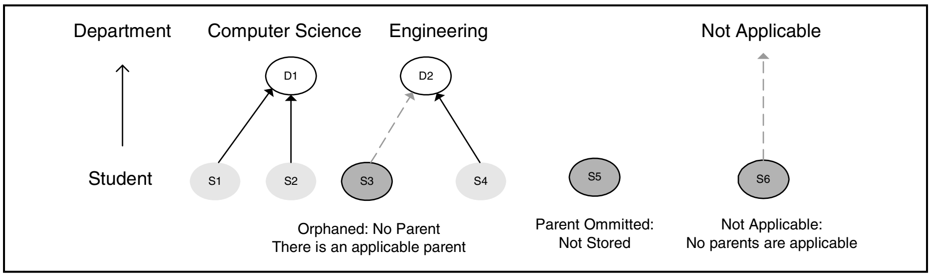


Fig. 3. Three Cases of Incomplete Hierarchy

Additionally, we will also differentiate null values that occur when there are no applicable parents. We label these missing values as *Not Applicable-incomplete*. In the student-department example, there are three situations where the hierarchy does not meet the completeness property. First, it is possible that a student was stored in the database, but their department was not stored in the database. Second, it is possible that the student existed and was assigned a department, but was not stored in the database. Finally, it is possible that the student is not and should not be assigned a department, as may be the case with non-matriculated students. In any case, aggregating the data along this hierarchy will be incomplete. Figure 3 depicts an incomplete hierarchy of the three cases.

Thus far, we have looked at situations where one dimensional category is rolled up to a single dimensional category (i.e. student rolled up to department). *Multiple path hierarchies* consist of hierarchies where a lower level category can be aggregated to more than one higher level category. In situations lower level instances are associated with more than one parent which are located in different dimensional attributes. In some cases, instances can only have one parent; while in other cases, instances can have parents in multiple different attributes. For example, Hurtado and Mendelzon [9] describe an example data warehouse schema, whereby an international organization keeps track of shops, which have a parent in either state or province, depending on their country. It is legal to aggregate the sales in a select set of states with those in a select set of provinces. However, if a child has parents in both of the selected attributes, then erroneous summaries can result.

This situation could be further complicated if these parents are part of different heterogeneous dimensions. This situation could happen in multiple data marts. Data marts can be tied together using drill-across techniques when dimensions linked to the facts are either exactly the same or perfect subsets of each other. Abelló, Samos, and Saltor [10] argue that completely conforming dimensions unnecessarily restricts the usage of drill across. They argue that it is possible to drill-across different fact tables if there are derivation, generalization, association, or flow relationships among the related dimensions. While it may be possible to relate fact tables that do not share dimensions, drilling across non-conformed dimensions could result in inaccurate summaries. Abelló et al. [10] note that dimensions evolve over time, and new schemas can be linked to older schemas through flow relationships. However, semantic relationships may differ at different times. For example, in [4], we describe how the dependency between area code and location was eliminated, making it possible to misinterpret comparison queries between facts stored before and after the dependency changed.

3.2 Data Level

Imprecise, biased, and inconsistent data may result in erroneous summaries. This is especially true when some measures stored in data warehouses are derived from *measurement instruments*. Measurement instrument is a term used to describe the method used to collect the base data. Measurement instruments can be physical instruments (e.g. thermometers, photometers, GPS systems, registers, bar-code scanners), or they can be methods for collecting data (e.g. census surveys, inventory counts).

Regardless of the medium used to collect the data, all measurement instruments have some imprecision associated with them. Imprecision refers to the exactness or reliability of the data collected from a measurement instrument. Bias is a measure of the systematic offset or shift of data collected from a measurement instrument. If the values captured are persistently lower than the data in the real world, then the measurement instrument is considered to be negatively biased. Alternatively, if the captured data are persistently higher than the real world value, the measurement instrument is positively biased.

Inconsistency occurs when the method used to record data changes because the physical instruments used to record the data were changed, the software used to capture the data changed, or the procedure used to record or capture the data was altered. Aggregating data that was collected using different measurement instruments (including process changes) can result in erroneous summaries. And, data collected with one measurement instrument may not be comparable with data collected using a different measurement instrument. Unless changes are recorded, decision makers may come to the conclusion that there were changes in the data; when, in fact, the differences were only a result of the measurement instrument.

3.3 Computation Level

Computational inaccuracies are problems that are related to using aggregate operators that are not appropriate for the data, or aggregating data with differing units. Rafanelli, Bezencheck, and Tinini [11] classify three types of meta-data that are considered relevant to summarizability, including the aggregation function (count, sum average etc.), the summary type (real, non-negative integer, etc.), and the phenomenon described by the statistical object (population, income, etc.). Computational problems occur when there is an illegal or problematic interaction between these different components. Specifically, an issue arises when summary objects cannot provide meaningful summaries of the phenomenon described by the statistical object using certain aggregate operators. Several types of computational problems are described below.

Lenz and Shoshani [2] distinguish between stock and flow measurements. Measures that can be characterized as stock measurements, such as inventory levels or bank account balances, are typically non-additive across the time dimension, meaning that these measures cannot be aggregated using the summation operator unless grouped by time. We named these measures temporarily non-additive [4]. Certain stock levels such as measurements of intensity, and pre-aggregated averages, maximums, and minimums, cannot be meaningfully added regardless of the elements by which these measures are grouped [5], [4]. While stock levels cannot be aggregated using the sum operator, all other aggregate operators can be used to aggregate stock measures. Other measures cannot meaningfully be aggregated using either the sum or average operator, such as measures of direction [4].

Certain measures cannot be meaningfully aggregated regardless of the aggregate operator. It is not mathematically permissible to aggregate values that have different units. In a scientific context, an example would be attempting to aggregate a measure of distance with a measure of mass. And, it is not permissible to aggregate two different measures of distance with different units unless the data are translated into a common unit. In a data warehousing context, basket counts cannot always be meaningfully aggregated because these measures aggregate items with different units. We named these measures categorically non-additive [4]. For example, a basket-count may aggregate 1 telephone with 3 packs of bubble-gum for a total of 4 items. In this case, the data are translated into similar units based on an abstraction hierarchy, whereby both bubble-gum and telephones are products. As we discuss in [4], certain basket-counts may be meaningful if there is a low level of abstraction necessary to translate the items into similar units.

In [5], Kimball and Ross note several types of measures that are inherently non-additive. They state that percentages and ratios, such as gross margin, are non-additive, and therefore, when designing systems, both the numerator and denominator should be stored in the fact table. Additionally, they note that it is important to remember when summing a ratio, it is necessary to take the ratio of the sums and not the sums of the ratios. In other words, the numerator and denominators should first be aggregated separately using the sum operator, and then the totals should be divided, yielding the appropriate ratio values.

4 Measuring the Influence of Inaccurate Summaries

Incorrect summaries can result if instances of measures are incorrect, not counted, or counted more than one time in an aggregate summary. It is not always possible to automatically identify and record sources of summarization problems, since many problems are derived from imprecise business process, measurement instruments, or human errors. Therefore, it is important to identify the *influence* that the erroneous results that may not be eliminated during the design process will have on decision making.

We note that not every summarizability violations may lead to inaccurate decisions. Some problems are insignificant, and would have no effect on the decision process; while others are quite significant and could result in severe mistakes. Eliminating all possible inaccurate results can be overly restrictive and effortful; alternatively, ignoring the potential for inaccuracies could lead to serious errors. One approach without making the system less restrictive is to identify summarization problems in conceptual models [12], [8]. While modeling issues is useful for depicting the existence of a problem, it does not show the influence that an issue may have on the decision-making process. The influence of inaccurate summaries on decisions can be classified along two dimensions, extent and impact. Whereby,

- *the extent* refers to how many decisions will be affected by a particular issue; and,
- *the impact* refers to the degree to which an issue will affect a particular decision.

The extent of inaccurate summaries refers to the scope of decisions that are affected by an inaccurate summary. It is not possible to precisely predict the extent of a summarizability issue because the exact queries that will be run against a schema cannot be precisely known ahead of time. The impact of inaccurate summaries is even more difficult to identify or estimate due to many important factors. In this paper we briefly discuss a way of estimating the extent of inaccurate summaries.

The *extent* of summarizability problems can be measured performing an analysis on the types of queries affected. To ensure a comprehensive analysis, each measure should be analyzed along each hierarchy using all aggregate functions. And, potential sources of biased, missing, or duplicate data, such as those described in Section 3, should be noted. Following this process of identifying all possible sources of erroneous data will be time consuming, but there are several ways to facilitate the process. One suggestion is to automate the process of identifying problems. Grumbach and Tininini [13] suggest a method of tracking numerical dependencies and using metadata to automatically aggregate data that are not necessarily complete. Hurtado and Mendelzon's [9] method of using summarization constraints using metadata can also facilitate the process of making the extent of inaccurate summaries visible.

Queries on data warehouses can be classified according to the decisions they are intended to support. Specifically, queries can be categorized into exploratory queries, comparison queries, and benchmark queries. *Exploratory queries* are used to get a general idea of the data. *Comparison queries* are used to compare distinct sets of data from the system, comparing summaries from different groups. And *Benchmark queries* are used to compare sets of data against a specified benchmark. The type of queries affects both the impact that an erroneous summary will have on a decision and the approach for managing the problem. Examples of benchmark queries, comparison queries, and exploratory queries are shown in Figure 4, 5, and 6, respectively.

How many of our branch stores met the profit goal of $5,000,000 last year?

Which regions of the country have cancer rates that are greater than one in a million?

Which programs have not shown a profit in two out of the previous five years?

Fig. 4. Benchmark Queries

Are electronics sales more profitable than appliance sales?

Which region of the country has had the most new customers during the previous six months?

Are there more cases of influenza this year than the average number of cases during the previous 10 years?

Fig. 5. Comparison Queries

How are the sales in North America doing?

How many people have contracted the HIV virus during the past year?

Fig. 6. Exploratory Queries

With comparison analysis, the impact of erroneous summaries will be most apparent if the bias of one group of data is different from the bias of another group. For example, if both appliance sales and electronics sales are positively biased by the same amount, comparisons between the two will not affect the decision, even though the data are inaccurate. However, if the total sales displayed for appliance sales is positively biased and the total sales for electronics is not, then an inaccurate decision could result if the bias is significant enough to change which group was more profitable.

In both benchmark and comparison analyses, erroneous decisions may be made if the error associated with the aggregate summary cause the displayed value to be on a different side of the decision cut-point than the true value. In exploratory analyses, there may not be any definable decision points at all.

In a sense, OLAP queries can be conceptualized as a type of informal statistical test. Statistical tests are used to determine whether groups are different from each other or a benchmark, and use the error to identify whether the output is reliable at a certain significance level. When running statistical tests, rigorous rules for identifying significance are needed. These rules are based on the probability of an output being incorrect based on the characteristics of the dataset. In data warehousing, it is also important to identify the reliability of an output, and similar considerations should be given to identifying whether the error renders an output insignificant for a given test.

5 Managing Inaccurate Summaries

In this section we suggest an approach that can be used to minimize and identify the impact that inaccurate aggregate summaries will have on a given query. We also suggest techniques and metadata that can be used to automatically detect and display their effect on analyses.

5.1 Schema Level

Much of the work on summarizability has focused on either structuring data warehouses in a manner that eliminates the potential for structural violations to occur or making the violations apparent through conceptual modeling. These techniques are useful for eliminating many inaccurate summaries from occurring or impacting decisions. However, there are many situations where systems are not optimally structured or conceptual models are not consulted. In these situations, it is important to make the impact of structural violations apparent at query time.

Scripts can be run that count the number of times a value is included in a summary. This value can be used to identify orphaned incomplete data or duplicate data resulting from non-strict and alternate path hierarchies. Eder, Koncilla, and Mitsche [14]

Table 3. Managing Schema Problems

Type	Queries Impacted	Management
Non-strict	Any query that rolls-up to the parent level of a non-strict hierarchy will be affected only if the lower level values are counted more than one time in the query	Run scripts at query time to identify the number times measures are counted in the summary and identify the total value of any duplicated values.
Incomplete: Orphaned	Any query that aggregates parents of orphaned data will be impacted only if the orphaned values are associated with measures.	Routinely run scripts to identify the extent of orphaned data and the value of the associated measures.
Incomplete: Not Applicable	Any query where it is important to distinguish whether a value is missing, orphaned, or not applicable	Use a not-applicable code when there is not an appropriate dimension member, rather than leaving the cell null.
Incomplete: Missing	Any query with missing data can significantly impact queries, especially when data are systematically missing	Attempt to identify reasons for missing data and extrapolate the impact of the data that were not captured.
Changing Schema Dependencies	Comparison queries that compare temporally dissimilar groups of data	Identify and track all hierarchical dependency changes in metadata.
Alternate Path Hierarchies	Merge queries will be inaccurate if data are counted multiple times. Comparison queries may count a single measure in more than one group.	Run scripts at query time to identify whether measures are counted in multiple groups or multiple times in a single group.
Heterogeneous Dimensions	Drill-across queries where the dimensions are not perfectly conformed	Dimensions should be conformed if significant inaccurate summaries result from heterogeneous dimensions.

describe the applicability of using regression, correlation, Fourier transforms, and principal component analysis to identify sharp changes in the structure of data warehouses. Specifically, they explore the use of these outlier detection algorithms for identifying hierarchical members that split, merge, change, or have moved. When inaccurate summaries result from heterogeneous dimensions, we recommend that Kimball and Ross' [5] suggestion of the conforming the dimensions be followed. Table 3 shows our suggestions for managing schema problems.

5.2 Data Level

Inaccurate summaries could significantly affect decisions in both business applications and in scientific database applications. Typical sources of errors include units, capture biases, errors due to capturing frequencies of stream data, etc. When the method used to capture and record measures changes during the history of data collection, the resultant data can be affected. Therefore, it is important to make heterogeneous measurement instruments visible to decision-makers. To reduce the likelihood of

Table 4. Managing Data Problems

Type	Queries Impacted	Management
Biased	Any exploratory query that aggregates biased measures Comparison queries where the groups are biased in different directions	Track the positive or negative bias associated with each measure, method, or measurement instrument.
Imprecise	Any query where the aggregate value is close to a decision point	Display the level of precision associated with a summary. Indicate the likelihood that an aggregate value is significantly above or below that value.
Inconsistent	Comparison queries that aggregate measures captured using dissimilar methods or measurement instruments	Store method code indicating to track how a particular measure was captured. Also, track how the dimensional members were captured.

erroneous conclusions based on these summaries, each different measurement instrument should be stored in metadata along with the associated measures. When aggregate queries are run, these metadata should be accessed. And, if there are multiple measurements within a single sub-cube or among different sub-cubes, then there is the potential that the measurement instrument affected the aggregate results.

When biases are known, the strength, direction, and extent of the bias should be stored in metadata. These metadata should then be accessed to adjust data based on these biases. There are several likely sources of bias, including data being duplicated, missing, or inaccurate. Queries that pull biased data should automatically adjust the data to eliminate the bias. We distinguish between three types of biases: missing, duplicate, or shifted.

Often, biases in a data set, however, cannot be precisely known. In these cases, it is also important to track the precision associated with the measurements. These precision values can be used to display a summary output that displays a range of values that could be representative of the true value, rather than a single imprecise value.

The impact of the imprecision and bias must be identified if decisions are going to be based off of these results. The total bias and imprecision should be combined to determine the offset and error associated with a summary. This information can then be used to measure the impact that these issues will have on a decision. Data problems can occur from imprecise, biased, or inconsistent data. To manage these problems, we suggest storing the error and offset for values associated with each measurement instrument. The method used to capture the data should be stored and used to distinguish among values captured using different processes or systems. Table 4 summarizes our suggestions for managing data issues.

5.3 Computation Level

Computational problems may impact decisions, especially when the violations are not apparent to the decision maker. Aggregating basket counts may result in an erroneous summary if dissimilar items are counted together. Averages performed on non-normally distributed data may give an incorrect perception of the central tendency.

And when specific rules are prescribed for aggregating data, erroneous decision may be made if these rules are not apparent to the decision-maker.

All measures have associated units, such as dollars, degrees, inches, or product types. The units must always be stored in metadata. It is best to store the units as a dimensional field or in a conformed dimension called Unit in the data warehouse. The Unit dimension should have also necessary conversion rules. Aggregate operations cannot be performed on measures with differing units unless they can be converted into a common unit. It is also important to check inter-set heterogeneity in comparison queries. The greatest impact will occur when the heterogeneity of the units is not apparent to persons performing the query. This type of error may occur if a single measure stores currency with multiple financial units that have near one-to-one exchange rates.

If queries are found to have data with more than one different unit, they must be transformed into similar measures. In cases where units can be assimilated computationally, this can be done automatically by storing conversion units in meta-data or in Unit dimension. Analysts will be able to aggregate the data by simply choosing the units for the data.

Many other computational problems result from performing illegal operations, and therefore should not be permitted. Systems should track whether fields are fractions, measures of direction, and stock values; and, queries that attempt to improperly aggregate these data should be prohibited. Table 5 depicts our specific suggestions for managing computation problems.

Table 5. Managing Computation Problem

Type	**Queries Impacted**	**Management**
Illegal Operations	Queries aggregating measures with heterogeneous units that appear to have similar units are most likely to be misinterpreted	Units for all measures should be stored in metadata and in a conformed dimension; scripts should be run at ETL stage or query time to convert units if heterogeneous units exist. Queries should be grouped by the units so that no summaries will aggregate measures with different units.
	Queries aggregating measures that are derived from fractions, such as Gross Margin Return on Investment (GMROI)	When using the sum operator, fractions should be aggregating by taking the quotient of the sums, rather than the sums of the quotients.
Type Compatibility	Sum Queries that aggregate snapshots of stock measures. Any query that aggregates data using an inappropriate aggregate operator, such as measures of direction.	Appropriate aggregate operators for each measure should be stored in metadata.
Statistical Requirements	Aggregations that are used for statistical calculations that have specific requirements	Show alerts when aggregate summaries are based on very limited number of instances. Analysis tools should show distribution of data and descriptive statistics for the summaries.

6 Conclusions

In this paper, we presented a taxonomy of inaccurate summary factors and practical rules for handling them. We discussed these issues from the perspectives of schema, data, and computation. We proposed several methods that can be used to identify problems based on the type of queries that will be run. Finally, we suggested metadata and practical rules that can be used to manage inaccurate summaries.

We note that not all summarization problems can be eliminated from OLAP systems. Furthermore, methods for eliminating and managing summarization problems can be effortful. Therefore, it is important to prioritize problems based on how likely they are to impact decisions.

The following heuristic rules can be used to quickly identify problems that have the potential to impact decisions:

- *Follow design guidelines wherever possible*
  Conforming dimensions and making dimensions orthogonal is an important step to reducing the likelihood for misinterpretation of aggregate summaries.
- *Identify inconsistencies*
  Beware of aggregating data that has been collected through different methods or collection instruments. It is also important to identify measures that are collected using the same method that are coded differently.
- *Link all measures to their corresponding units*
  The units of all measures should be stored either in the data warehouse or in metadata. Additionally, if the units associated with measures are part of a hierarchy, the associated hierarchy should also be stored. Measures should only be aggregated with measures that have similar units.
- *Make imprecision and bias visible*
  Where it is not advantageous to completely eliminate the potential for inaccurate summaries, systems should allow decision-makers to make more informed decisions by making error and offset values accessible through links to visual or tabular outputs.
- *Track the dimensions along which measures are non-additive and non-summarizable*
  OLAP systems should then display alerts or prevent queries that attempt to improperly summarize these measures.
- *Make computational assumptions and operations visible*
  OLAP tools often hide the computational aspects of aggregating data, such as rounding rules, equations, and statistical assumptions. This information should be directly accessible through OLAP tools so analysts can quickly and easily identify potential computational problems.

This paper contributes to a comprehensive understanding of summarizability and their impact on decision-making. Identifying source of errors and learning how to manage them can reduce unnecessary effort of imposing overly restrictive rules to eliminate all summarizability violations. Our paper gives designers a means to manage and prioritize inaccurate summary problems.

References

1. Martyn, T.: Reconsidering Multi-Dimensional Schemas. *SIGMOD Record.* Vol. 33, No. 1. ACM Press. New York, NY (2004) 83 – 88.
2. Lenz, H-J. and Shoshani, I.: Summarizability in OLAP and Statistical Data Bases. *Ninth Int'l Conf. on Scientific and Statistical Database Management* (1997) 132-143.
3. Shoshani, A.: OLAP and Statistical Databases: Similarities and Differences. *Proc. of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems,* Tucson, Arizona (1997) 185 – 196.
4. Horner, J., Song, I.-Y., and Chen, P.: An Analysis of Additivity in OLAP Systems. *DOLAP'04*, Washington, DC, USA (2004) 83-91.
5. Kimball, R. and Ross, M.: *The Data Warehouse Toolkit*: Second Edition. John Wiley and Sons, Inc. (2002).
6. Lehner, W., Albrecht, J. and Wedekind, H.: Normal Forms for Multidimensional Databases. *Proc. of the 10th International Conference on Scientific and Statistical Data Management (SSDBM'98)*, Capri, Italy (1998) 63-72.
7. Hüsemann, B., Lechtenbörger, J., and Vossen, G.: Conceptual Data Warehouse Design. *Proc. of International Workshop on Design and Management of Data Warehouses* (2000) 6
8. Tryfona, N., Busborg, F., and Borch Christiansen, J.: StarER: A Conceptual Model for Data Warehouse Design. *DOLAP '99*, Kansas City, MO, USA (1999) 3-8.
9. Hurtado, C and Mendelzon, A.: OLAP Dimension Constraints" *ACM PODS 2002*, Madison, WI, USA (2002) 169-179.
10. Abelló, A. Samos, J., and Saltor, F.: On Relationships Offering New Drill-across Possibilities. *DOLAP '02*. McLean, VA, USA (2002) 7-13.
11. Rafanelli, M., Bezencheck, A., and Tininini, L.: The Aggregate Data Problem: A System for their Definition and Management. *SIGMOD Record*. Vol. 25, No. 4 (1996) 8-13.
12. Trujillo, J., Palomar, M. Gomez, J., and Song, I.-Y.: Designing Data Warehouses with OO Conceptual Models. *IEEE Computer*. Vol. 34, No 12. (2001) 66-75.
13. Grumbach, S., Tininini, L.: On the Content of Materialized Aggregate Views. *ACM PODS*, Dallas, TX, USA (2000) 47-57.
14. Eder, J., Koncilia, C. Mitsche. D.: Automatic Detection of Structural Changes in Data Warehouses. *DaWaK* (2003) 119-128.

XCM: Conceptual Modeling for Dynamic Domains

Luis González Jiménez

Department of Business and Economics,
Universidad de La Rioja, Cigüeña, 60, 26004 - Logroño, Spain

Abstract. ERM remains the tool of choice for conceptual data modeling. It relies on entities and relations to model the domain of interest and so does essentially (differences in notation and terminology aside) object oriented modeling. Consistently with this perspective, prevailing "temporal models" are based on facts and their associated valid and transaction times, rather than events and the corresponding occurrence times. Arguably the approach is, in both cases, inadequate for design independent modeling of dynamic (i.e., time-varying) domains and possibly complicates the definition of the system's behavior.

This paper puts forward an alternative, XCM, that purports to be richer and more domain-oriented, specially where history and forecasting support requirements arising from the temporal nature of the domain of interest are concerned, and that may have a positive influence in other aspects of conceptual modeling.

Keywords: Conceptual Modeling, Entity-Relationship Model, Temporal Databases.

1 Introduction and Motivation

Despite the popularity of Object Oriented Modeling (OOM), the Entity Relationship Model (ERM) [3] remains the most widely used conceptual data model. Arguably, its success derives in large measure from its application to the specification and design of relational databases, so far the most common in practice. Though the ERM has often been revisited, few modifications have gained widespread acceptance, notably its extended version which introduces the category concept for the representation of specialization and generalization [5].

Prominent among the motivations behind many proposed extensions to the ERM is the need to incorporate into database design in general [22] and conceptual data modeling (CDM) in particular [9] the requirements that arise from the temporal nature of the domain of interest.

From an ontological perspective, existing temporal data models are based on the concept of fact, rather than on the idea of event. It is assumed that the domain of interest is best contemplated along the axis of time as a succession of states. States have extent over time. Facts (about objects) are true for an interval of time. The (instantaneous) occurrence of events results in facts becoming or ceasing to be true. Hence, events and states are duals: states can be represented by their delimiting events and events are implied by states. The usual approach is to "timestamp"– with valid times (indicating when was a fact true in reality), transaction times (that tell when was a fact current in the database) or both– entities, relations and attributes of either [13,14]. This approach is consistent with the ERM's ontology. According to Chen's seminal paper "The entity-relationship model adopts the more natural view that the real world consists of entities and relationships" and "an entity is a 'thing' which can be distinctly identified. A specific person, company, or *event* is an example of an entity." [3]. That is, the ERM's ontology does not tell things that *are* (entities proper) from things that *happen* (events). Nor does it consequently differentiate relations among entities from those associating them to events or relating events to one another.

ERM adopts the more natural view of a *static world*. However, *dynamic domains* are the rule rather than the exception and CDM should provide the means to model time and time-varying information in a natural way. Fact-based models do not provide those means because the way ordinary people look at a changing domain is not in terms of "what is the period of time within which this fact was true?"; rather, it is in terms of "when did this happen?". Tellingly, none of the proposals for temporal extension to the ERM surveyed in [9] involves the addition of the *event* either as a new construct or as a variety of those constituting the ontology of the ERM. This does not mean that the notion of event is absent from requirements engineering and high-level system design. Event and other dynamic concepts – action, activity, state transition, and process – are central in goal and agent-oriented models (e.g.: KAOS [4], RML [10], TROPOS [2,6], or AOR [20]) and in (Event-Condition-Action) business rules conceptual modeling [15,21]. However, events are rarely modeled as persistent data objects and are frequently identified with state changes or with actions.

Indeed, the notion that domain dynamics may very well be captured in terms of states, state transitions and processes (leading from one state to another) is pervasive [16]. XCM takes a slightly different approach to the matter. The problem at hand may be considered from two angles. First, how to model a changing domain and what happens within it, regardless of time. Second, how to incorporate into the model a measure of "temporal support", so that information about the domain's past and, if necessary, about its future may be provided by the IS. Research in temporal databases is essentially if not exclusively concerned with the second angle. On the other hand, XCM (where CM stands for conceptual modeling) begins by considering the problem from the first angle and comes up with a different perception of the domain. The latter is a view of a *dynamic world* and should arguably be closer to users and domain experts (collectively referred to hereinafter as "the domain observer") view of the world. The departing point for this purpose is to redefine the concept of *entity* so as to exclude *events* and to do so while differentiating between events and their effect on the state (likewise, from XCM's perspective, actions are not events, though they may prompt them to happen). This rather finer point has its merits. In [19] discrete processes are described as sequences of states (periods of inactivity) and events (changes in the state), while continuous processes are bounded by their initiation and cessation. Through differentiation of events and state changes, XCM allows to model non-atomic events causing more than one state change or no change at all. In addition and by considering that certain events (not associated with entities' lifespan) may have a deferred and/or time-bound effect on the state, it allows to model events' aftereffect. Finally, defining states as fully derived from events makes possible, at least at a conceptual level, to model continuous change.

2 Theoretical Foundations

As already noted, Chen's paper [3] states that: "An entity is a 'thing' which can be distinctly identified." Because the two propositions *things exist* and *things happen* are true in reality, yet the things about which either proposition is true are different, Chen's claim may be reformulated in the following terms:

Definition 1. *An entity is a thing that exists (independently) and has duration. An event is a thing that happens (independently) and is instantaneous. Relationships are associations among entities, events or both. Relations between entities exist (and last), while those among events or between entities and events are instantaneous. To avoid any ambiguity, the latter will always be referred to as instantaneous relations, while those associating entities will be referred to just as relations.*

Consistently with **Definition 1**, the following three axioms may be formulated.

Axiom 1. *The state of the domain of interest at an instant may be described, at a certain level of abstraction, by the existing entities and relations among entities, and the values of their attributes.*

Entities attributes represent features of the former that the domain's observer considers relevant excluding markers of their association with other entities (foreign keys, FK).

Relations represent associations among entities which may or may not include, in addition to markers of the participating entities (FK), attributes representing relations' features relevant to the observer.

For the sake of clarity, let us elaborate a bit on the implications of the second paragraph of **Axiom 1**. XCM considers that, for conceptual data modeling purposes, the possibility of mapping relations to entities' attributes (as FK) should be ignored. Data objects representing relations would contain all the relevant information including both, which ones are the associated entities and the relation's *descriptors* (attributes that are not foreign keys) if any.

Axiom 2. *The state of the domain is subject to the following classes of changes:*

1. *appearance of a new entity;*
2. *disappearance of an entity;*
3. *an entity's attribute (FK excluded) takes value (for the first time);*
4. *change in the value of an entity's attribute (FK excluded);*
5. *appearance of a new relation;*

6. *a relation's attribute takes value (for the first time);*
7. *change in the value of a relation's attribute; and*
8. *disappearance of a relation.*

Axiom 3. *The original domain state is a vacuum and any subsequent state is the result of the accumulation of state changes.*

It is now possible to define events in a very precise fashion.

Definition 2. *An event is an instantaneous burst of activity involving entities, either actively or passively. Events may or may not result in perceptible changes to the state, but state changes always result from events. On the other hand, (occurred) events are not themselves subject to changes because the past cannot be altered.*

Events always have an effect on the state because if anything happens, something necessarily changes, temporarily or for good. **Definition 2** does not contradict this claim. It is possible that an event is both, perceptible and relevant (by and to the domain's observer), while its effect on the domain is not perceptible, irrelevant or both. On the other hand, an event may cause more than one change to the state, and the same state change may be caused by different (types of) events (e.g., the balance of a customer may change as a result of a sale, a payment or a rebate).

Definition 2 also states that events are instantaneous. Yet, *macroevents* (or processes) are defined in [8] as events with duration; that is, events that occur over an interval of time. This contradiction is only apparent. XCM assumes that macroevents do not exist, events being instantaneous by definition. Rather they are chains of events and the states that obtain in between, or events whose effect on the domain is either or both, deferred and limited to a known time period. For instance, assume that a train has its departure today at 9:30 p.m, it travels overnight non-stop and is expected to reach its destination tomorrow at 9:30 a.m. How many events occur (and when as measured with a discrete time scale)? To answer this question it is necessary to define the precision of time measurements; i.e., the basic time unit (or *granule* [8]). If time is measured in minutes, there are 720 instants between today at 9:30 p.m. and tomorrow at 9:30 a.m., and that is the maximum number of events that may be defined. Yet, to know what has happened and what is happening (and with more exactitude than that which is provided by minutes), a much smaller number of events are necessary:

– A first option is to make do with two events: departure (event time: today at 9:30 p.m.) and arrival (event time: tomorrow at 9:30 a.m.); in between these two events, the train is travelling.
– The second option involves just one event: departure (event time: today at 9:30 p.m.). A second time magnitude that may be either "travelling time" (12 hours) or "arrival time" (tomorrow at 9:30 a.m.), will respectively indicate the time-bound or deferred effect.

Both options preserve the "instantaneous nature", but the second one involves the use of foreseen information, while the first one does not.

Let us leave aside for the moment the possibility of making forecasts and turn to the domain observer and wonder what he or she may be interested in. Essentially: in the current state of the domain, in its past states and in what has happened in it. Possibly, if the observer is willing to have information about what has occurred, he or she will also want to know when did it happen. Likewise, with few exceptions, past states will be interesting only if accurately located in time.

Based on this admittedly loose description of the possible information interests of a domain observer the following definition may be formulated.

Definition 3. ***Historical information*** *may be defined as data (either raw or processed) about events occurred in the domain at a point in time or within a period of time, about state changes caused by occurred events, or about the state of the domain at a given point in time in the past. Consistently, it may be said that a database provides history support (HS) if it can supply the necessary flow of data for the hosting information system to produce historical information.*

XCM follows the basic principle that states are the result of accumulated state changes and these are the consequence of events. Databases designed and implemented to comply with conceptual data models written in accordance with this basic principle should yield historical information as above defined. This claim is substantiated in **Section 3.2**.

Let us now go back to the possibility of forecasting (see **Figure 1**). Obviously, the future cannot be foretold, yet it may be foreseen. Foreseeing and forecasting, as used herein, are meant to include, besides more or less educated guesses on the future, the planning or scheduling of events and changes in the state. In the real world, future events and state changes as such do not exist. Forecasts do. These will later on be proved either right, partially right or completely wrong.

The following definition will be useful to discuss the possible interests of the observer in this regard.

Definition 4. ***Forecasts*** *may be defined as data whose contents are predictions by the observer of the domain in the form of:*

1. *delayed and/or time-bound state changes caused by occurred events; and*
2. *foreseen events (and their effect on the state which may be immediate, but also delayed and/or time-bound).*

Definition 5. ***Forecasting information*** *may be defined as forecasts (raw or processed) as already defined, together with data about the (expected) state of the domain at a given point in time in the future. Consistently, it may be said that a database provides forecasting support (FS) if it can supply the necessary flow of data for the hosting information system to produce forecasting information.*

3 XCM Model and Graphical Notation

This **Section** defines XCM's *morphology* and *syntax*, assuming that:

1. the database time scope contains the domain's history and foreseen future (see **Figure 1**);

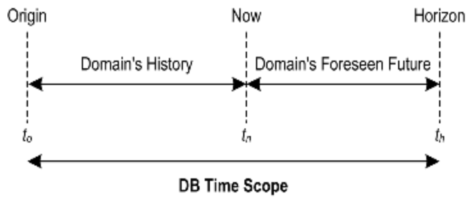


Fig. 1. DB's time scope for total HS and FS

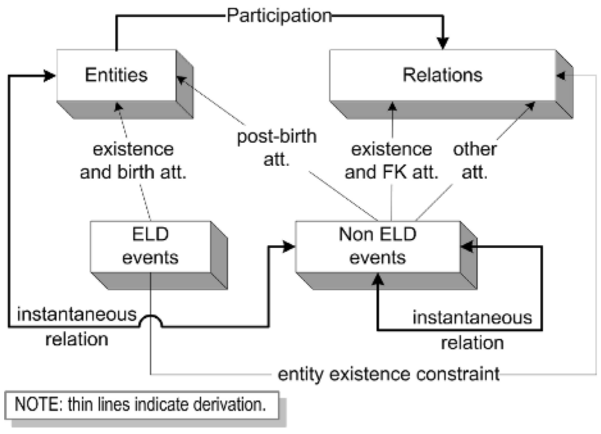


Fig. 2. Events' Classes and State-components

2. history and forecasting support are required for the whole domain of interest; and
3. the continuous model of time applies.

The first two assumptions imply that the database must contain the records of all the events occurred or expected to occur within the indicated time scope (shown in **Figure 1**), including the time of their occurrence, and that every single change suffered by the state during that period must be derivable from recorded events.

How to adapt XCM when either or all of above conditions do not hold shall be briefly discussed in **Section 3.3**.

Figure 2 shows the essence of XCM's morphology and syntax. In particular, it depicts the two classes of events that will be defined, and their rapport to the state components: entities and relations.

3.1 Events: Classes and Defining Features

XCM assumes events to belong to either of the two classes defined below. Essentially, they differ in how they modify the state of the domain. To fully appreciate the significance of their configuration and properties, it is important to note that the state (as defined in **Axiom 1**) at a given point in time will be derived from data objects representing events of these two classes.

Definition 6. ***Entity-lifespan delimiting (ELD) events*** *are those whose effect on the state is either*

1. *the appearance in the domain of a new entity, including its attributes' values if they are both, constant (i.e., they take values only once) and known at that moment ("birth attributes"); or*
2. *the disappearance from the domain of an entity.*

XCM models ELD events (**Figure 3**) according to the definition above and with the following defining **features**:

– Both classes of ELD events (appearance and disappearance) associated with any given entity will be modeled as one single data object.
– The attributes (entity) lifespan starting time (LST) and lifespan ending time (LET) are event occurrence times that indicate respectively the points in time at which an entity appeared or disappeared (from the domain of interest). LET will usually be an optional attribute because its value will not be known beforehand. For obvious reasons, it shall always hold that $LST < LET$.
– Neither the LST nor the LET can be used as identifiers (primary keys). ELD events should be identified by the same attribute as the derived entity.
– Entities resulting (derived) from ELD events are part of the state within the timeframe thus delimited (by LST and LET) and only within it may they be part to non-ELD events (defined below).
– For every instance of an ELD event such that LST $\leq t$ and $LET > t$ or *null*, an instance of the associated entity is part of the state at t, and the values of the entity's birth attributes are the values of the same attributes in the ELD event (derivation does not require proper computation).
– By definition, ELD events cannot participate in relations, either instantaneous or otherwise.

Definition 7. ***Non ELD events*** *are those whose effect on the state is any change or changes other than those caused by ELD events or that do not alter the state of the domain. State changes caused by events of this class may be limited in time and will be either instantaneous or deferred.*

XCM models non-ELD events (**Figure 4**) consistently with above definition and with the following defining **features**:

– The attribute event occurrence time (EOT) indicates the point in time when a non-ELD event has occurred or is expected to occur (second type of forecast enumerated in **Definition 4**). In principle, the EOT is an identifier (or primary key). This is theoretically (hence conceptually) possible as far as the continuous model of time applies and it is assumed that no two events may happen exactly at the same point in time. However, this possibility (identification by the EOT) poses some problems that will be elaborated on at the end of this section.

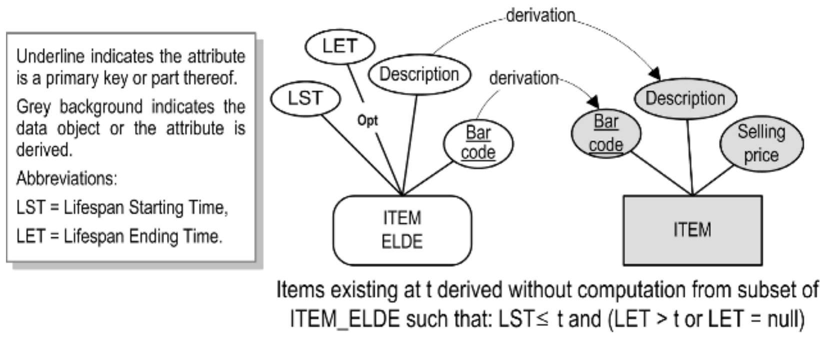

Underline indicates the attribute is a primary key or part thereof.
Grey background indicates the data object or the attribute is derived.
Abbreviations:
LST = Lifespan Starting Time,
LET = Lifespan Ending Time.
LET
LST
Opt
Description
Bar code
derivation
derivation
Description
Bar code
Selling price
ITEM ELDE
ITEM
Items existing at t derived without computation from subset of ITEM_ELDE such that: LST≤ t and (LET > t or LET = null)

Fig. 3. ELD event

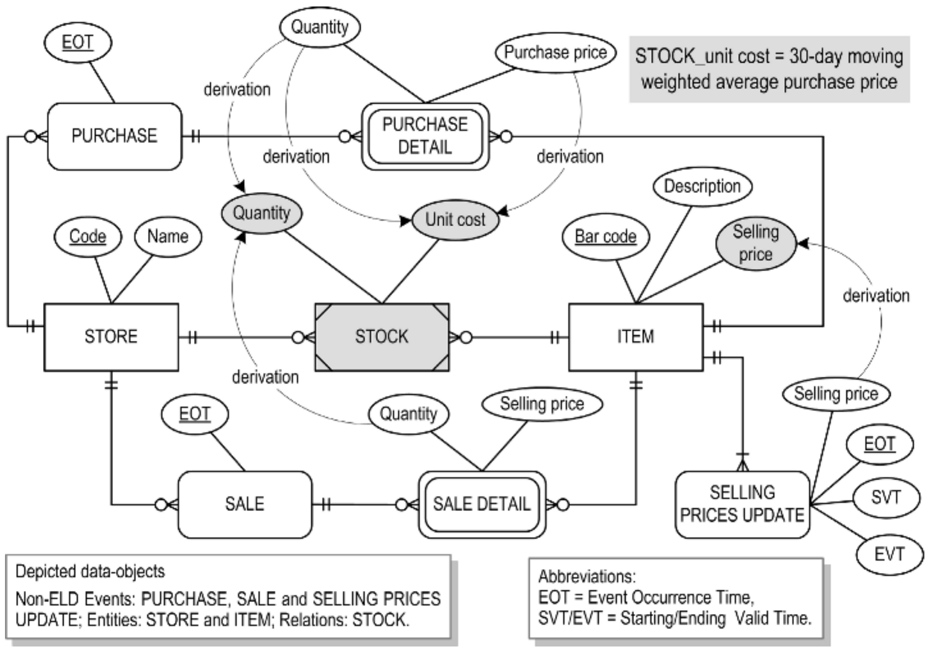


Fig. 4. Non-ELD events and derivation

– If the effect of a non-ELD event is either deferred, time-bound or both (first type of forecast enumerated in **Definition 4**), attributes indicating the associated time magnitudes must be added. Hence, the attributes starting and ending valid times (SVT and EVT) respectively indicate the points in time where the effect (or one of several effects) of a non-ELD event on the state will or is expected to take place (SVT) and cease to take place (EVT). For instance, an item price established through an instance of the event SELLING PRICES UPDATE (**Figure 4**) will be valid only for the period of time

bounded by the values of the event's SVT and EVT. The temporal relations between EOT, SVT and EVT are extremely important and will also be discussed at the end of this section.
- Data objects representing non ELD events will provide the data required for derivation of the values of entities' postbirth attributes (e.g., the ITEM selling price in **Figure 4**) and of existing relations and values of their attributes (for instance STOCK and its two descriptors, quantity and unit cost in **Figure 4**). More often than not, derivation in this case will be complex and require actual computation. Not only nor mainly because of the temporal constraints on the event's validity, but because the same event may affect more than one entity or relation and these may in turn be affected by different (types of) events. In the case of **Figure 4**, the stock of an item at a particular store and at a certain point in time would be computed adding all the quantities of the item purchased for the store and deducting the item quantities sold at that store. Computation of the unit cost would require, for each stock, adding the quantities of the associated item bought within the period $[t - 29\ days, t]$ for the associated store, tallying the corresponding amounts (quantity times price), and dividing the total resulting amount by the total resulting quantity.
- Consistently with **Definition 1**, non ELD events may only participate in instantaneous relations. For instance, in the running example (**Figure 4**) there cannot exist a relation linking PURCHASE and STOCK, but it would be valid one associating PURCHASE to another non-ELD event like PAYMENT (not included in the schema of the figure).

Besides the defining characteristics enumerated above for each class of events, both ELD and non-ELD events have the following properties in common:

- Attributes of events are always constant (i.e., non-derived).
- Notwithstanding errors correction and foreseen data update, within the DB time scope data objects associated to events cannot be deleted or modified.
- Foreseen events will be modeled in exactly the same way as occurred events. However, their occurrence times (EOT, LST or LET, as applicable) will be greater than the current time.
- When suitable, both classes of events may be represented using complex constructs. Namely, supertype/subtype hierarchies and combinations of strong and weak components.

To close this section, a brief discussion on the temporal features of non-ELD events is befitting. **Figure 5** will be illustrate the analysis of the different possibilities regarding the order of precedence of the three time magnitudes associated with these events plus the *transaction time*, defined in this context as the point in time when the event is registered in the database. Numbers in the figure indicate different temporal orders relative to the EOT.

Let us assume that delayed update is not legal, so that the transaction time (TT) of an event has to be either equal or earlier (smaller) than the event's occurrence time (EOT).

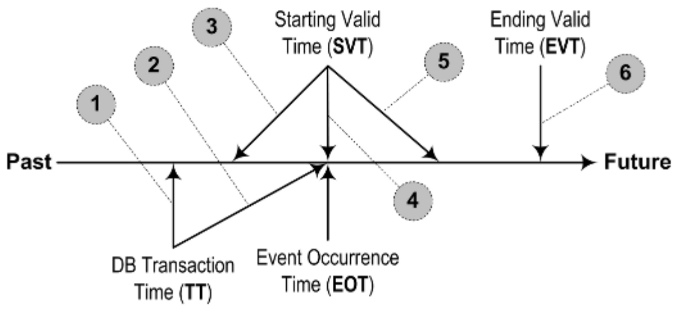


Fig. 5. Relations between occurrence, valid and transaction times

The first "basic" order is that of occurred events ($TT = EOT$, number **2** in **Figure 5**) whose effect on the state is not time bound (they have no EVT) nor deferred ($SVT = EOT$, number **4** in **Figure 5**). This order corresponds to events that are not forecasts and the SVT should be omitted.

The second "basic" order encompasses occurred events whose effect is deferred (number **5** in **Figure 5**, $SVT > EOT$) and/or time-bound (number **6** in **Figure 5**, $EVT > SVT$ or, if the event has no SVT, $EVT > EOT$). These events are forecasts of the first type enumerated in **Definition 4** and there is something to be noted for those with EVT. Assume that the value of an event's EVT is t; further assume that the value of an entity's postbirth attribute is derived from it (e.g., the ITEM selling price in the case of **Figure 4**). At t, unless there is another event (of the same type and associated to the same entity) such that $SVT \leq t$ and $EVT > t$, the value of the entity's postbirth attribute will be null. There is nothing wrong with this because it mirrors reality, but it should be taken into account when modeling business rules.

These two "basic" orders pose no problems as far as consistency with the configuration and features discussed in this section is concerned.

Foreseen events provide the third case of temporal order (number **1** in **Figure 5**, $TT < EOT$). These events may, in addition, have deferred and/or time-bound effects on the state (numbers **5** and **6** in **Figure 5**).

This third possibility creates a small difficulty. The EOT takes a foreseen value until the event actually occurs, hence it is subject to changes (resulting in either anticipation or delay of the occurrence time). This causes a problem of data integrity since the EOT identifies non-ELD events (in principle). The safest and easiest solution is to use in this particular case an alternative identifier. In addition it must be noted that in the case of events that always begin as forecasts, the second "basic" order with $EOT = \textit{time of making the forecast}$ is possibly more adequate and does not entail inconsistency problems.

Finally, there is a fourth case, indicated by number **3** in **Figure 5** ($SVT < EOT$). This possibility does not exist in the physical world: events that have not yet happened cannot have any effect on the state. However, this case may arise, for instance, in the context of business transactions (e.g., a pay-rise given on March applicable as of the first of January); actually, in an extreme case, it might also happen that $EVT < EOT$. This temporal order creates the same sort of consistency problems that delayed update does. If delayed update is legal (i.e., events with $EOT < TT$ are valid), then the current state may not be fully known (by the system hosting the database). Likewise, if $SVT < EOT$, then the current state at $t \in [SVT, EOT)$ is not fully described. It is unrealistic to assume that delayed update may be completely barred, it should be addressed when modeling queries (e.g., giving indication to users that the information provided may be not up-to-date) and business rules, and so should this fourth case.

3.2 Derivation of the State and HS Provided

The following table summarizes the morphology of XCM data objects.

Class	Temporal attributes	Attributes	Foreign Keys (of)
ELD event	LST, LET	Non-derived	None
Non ELD event	EOT, SVT (optional), EVT (optional)	Non-derived	Participating entities, other non-ELDE
Entity	Any (optional)	Derived	None
Relation	Any (optional)	Derived	Associated entities

Let us now recall and elaborate a little on the assumptions formulated at the beginning of this **Section 3**:

1. the database time scope contains the domain's history and foreseen future**,** i.e., every single event (ELD and non-ELD) occurred or expected to occur since the origin until the horizon is registered in the DB;
2. the continuous model of time applies, i.e., all the time magnitudes in the database – EOT, LST, LET, SVT and EVT – are positive real numbers.

Let $[t_o, t_h]$ be the DB time scope (**Figure 1**). The state of the domain at a given point, $t \in [t_o, t_h]$, may be described by the existing entities and relations among entities, and the values of their attributes (**Axiom 1**). All of which results from events (of both classes) occurred on or before t and whose effect on the state is valid in t. The latter applies for non-ELD events with either or both, starting and ending valid times and for derivation of relations (from non-ELDE events) which is subject to the associated entities' existence (i.e., relations' existence may depend indirectly on ELD events).

Derivation of the entities existing at t is straightforward: there is one and only one for each ELD event such that $LST \leq t$ and LET is greater than t or *null*; and the value of their "birth attributes"is derived as already indicated (see **Figure 3**).

In all other cases, derivation will follow a set of *derivation rules* (DR), which are *static business rules* (as opposed to dynamic or Event-Condition-Action, ECA, business rules [1,12]) that have to be defined and that embody the effect of non-ELD events on the state. Whether DR are modeled as part of the data-objects definitions (in the data dictionary), as independent objects (business rules to be precise), or combining both alternatives (e.g., as business rules if they may vary over time or across the domain and as part of the data-objects definitions otherwise), should be decided taking into consideration domain's characteristics, observer's requirements and mappability to design components.

It must be noted that there are two *general constraints* that apply for derivation (and, hence, need not be explicit in derivation rules' definitions):

- to derive the state at *t*, only those non-ELD events occurred on or before *t* are relevant and
- in application of the *entity existence constraint*, only relations associating entities that exist at *t* have to be derived, likewise only attributes of entities and relations existing at such time must be derived.

Specific DR may be very diverse, however, there are some basic "metarules" (or rules to be complied with when formulating derivation rules). Namely:

- DR rules must identify clearly:
    - the derived element: entity postbirth attribute, relation (existence, expressed as foreign keys' values), or relation descriptor (derivation of two or more elements associated with the same data object may be expressed in one single rule);
    - the non-ELD event type or types that determine the value of the derived element, let us call these event types the *sources* for derivation;
    - *constraints*, temporal (if they have starting and/or ending valid times) and otherwise on the members of the *determinant set(s) of events*, which are constraints-compliant subsets of the sources.
- DR will also state unambiguously how to compute each instance of the derived element in terms of the determinant at any point in time (provided it is part of the DB timescope):
    - if the derived element is an entity's attribute or relation's descriptor, the rule will define a mathematical formula for computation of the attribute's value as a function of the values of one or more attributes of the events in the determinant set(s);
    - if the derived element is a relation (existence and foreign keys' values), the rule will specify what attribute of which event in the determinant set(s) yields the value of each foreign key making up the relation; consistently, the specified event attributes must be identifiers of the related entities.
- Precisely to avoid ambiguity, DR should be formal. However, it makes sense to accompany rules' formal definitions with their expressions in natural language. The latter would be 'intentional rules' and the former 'operational rules' as defined in [15], not as different classes of (implementation-free) rules, but as alternative expressions (informal and formal), formulated with different approaches (business context/processes), that belong in two different stages of IS analysis (intentional/operational). In the normal course of the conceptual modeling stage of a project, intentional rules would be elicited and analyzed and later on redefined in formal form.
- Finally, derivation rules can and should always be declarative.

A database designed and implemented to comply with a conceptual model thus defined would fulfill the first two assumptions formulated at the beginning of this **Section 3 and** satisfy **Definitions 3** and **5**. The information provided by the database would include all the events occurred during the time interval $[t_o, t_h]$, and the states at any point in time within the interval.

3.3 Partial Forecasting and History Support

For a database to provide total HS and FS the three assumptions at the beginning of this **Section 3** have to be met. However, the third assumption cannot hold in practice because, for implementation purposes, the continuous model of time is not feasible, available clocks being able to provide but a discrete measure of time. On this matter, for non-critical systems the discrete model would seem to be sufficiently precise. Within this frame, it would be a matter of domain's observer preferences (i.e., requirements) what granularity(ies) should be used. However, use of the discrete model poses two problems:

– Non-ELD events should be identifiable by their occurrence time (EOT), and it is evident that only the continuous model guarantees that the EOT is a candidate key.
– In addition, chronological ordering of events occurred at the same granule may be required.

Both problems can be solved using a *complex timestamp* for the EOT that expresses both the absolute and the relative occurrence times. The latter being relative to other events occurred within the same granule (month, day, hour, etcetera).

A second limitation refers to the possible existence of a DB time-lower bound that is beyond the beginning of domain's history (so that the first assumption does not hold either). For design and implementation purposes, this possibility should be taken for granted. Unless it is the first information system that is being developed, events occurred within the domain from its very beginning will be irrelevant if not unavailable. In addition, once the system is in operation, *vacuuming* capabilities (for the periodical physical removal of historical data in a disciplined manner) [18] will be required. Unless this limitation is treated as a logical design problem, the conceptual model has to be modified as follows:

– the modeled DB should be assumed to contain all the required information about the *initial state*: existing entities and relations, and values of their attributes as at the point in time constituting the DB time-lower bound;
– the modeled DB must also be assumed to contain:
  • non-ELD events occurred before the DB time-lower bound but with starting and/or ending valid times greater than it, and
  • ELD events for entities that are part of the initial state;
– finally, derivation rules have to be defined taking into account the additional information included in the DB.

Concerning the second assumption, more often than not, the preferences of the domain's observer regarding history support will not be homogeneous across the domain. For instance, the observer may be interested in the clinical history of a patient and in his/her current personal data but not in his/her personal history (previous addresses, phone numbers, etcetera). Likewise, the required forecasting support (FS) will vary, both as regards the time scope and the covered domain area. Limited FS poses no problem as far as the discussed model is concerned. Limited domain coverage for HS purposes may be solved, at least in part, by allowing deletion of ELD events for entities that do not require HS, and/or modification of their attributes if no HS is required for the derived entities' attributes. Likewise, binary relations without descriptors may be excluded from the derivation mechanism as long as temporal consistency is preserved and if no HS/FS is required or if they are not subject to changes. Further simplification of the model would require allowing further updates (modification of attributes' values and/or deletion of data objects) of the initial state, of non-ELD events, or both. In this case, it might be better to use the alternative discussed in [7] (REERM, Reenhanced Entity Relationship Model), a precursor of XCM better suited when HS requirements are very limited.

4 Summary and Directions

In addition to a natural (i.e., closer to observer's perception) way of modeling dynamic domains, XCM provides the constructs and the syntax to smoothly incorporate history and forecasting support requirements. Moreover, XCM may have a positive influence in other aspects of CDM. For instance, working simultaneously and interactively on and with a list of events and another one of entities should improve elicitation and analysis of information about the domain. So would likely do to work with *event clusters* instead of *entity clusters*. A unit (department, division, etcetera) within an organization is usually concerned with a certain set of events, while entities tend to be "transversal", to be present in events concerning different units.

On first acquaintance, XCM may seem a bit awkward to apply in practice. Two considerations should be made in this regard. First, it has to be noted that, more often than not, domains are strongly dynamic. This implies that the different types of events (of the non-ELDE class) that occur in the domain outnumber by far the types of entities present and, in addition, the number of instances of the former is also much bigger. Second, observers of such domains will possibly have a stronger interest (which should translate into requirements) on information about events than on information about states (and in the latter case, some degree of HS and FS should be expected to be required). Where these two conditions hold, XCM is better and easier to apply than the usual (fact-based) approach and facilitates conceptual modeling in general, which also encompasses implementation-free definition of the IS behavior, including dynamic (Event-Condition-Action, ECA, [1,12]) and static (consistency, [11]) business rules, (concept-based) queries [17] and use-cases.

References

1. Assche, F. Van, Layzell, P.J., Loucopoulos, P. and Speltincx, G. 1988. Information Systems Development: a Rule-Based Approach. Journal of Knowledge Based Systems (September): 227-234.
2. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., Perini, A. 2004. TROPOS: An Agent-Oriented Software Development Methodology. Journal of Autonomous Agents and Multi-Agent Systems: 203-236.
3. Chen, P.P. 1976. The Entity-Relationship Model Toward a Unified View of Data. ACM Transactions on Database Systems, 1(1): 9-36.
4. Dardenne, A., van Lamsweerde, A. and S. Fickas. 1993. Goal-Directed Requirements Acquisition. Science of Computer Programming, 20: 3-50.
5. Elmasri, R., Hevner, A., and Weeldreyer, J. 1985. The Category Concept: An Extension to the Entity-Relationship Model. Data Knowledge Eng., 1(1): 75-116.
6. Fuxman, A., Liu, L., Mylopoulos, J., Pistore, M., Roveri, M., Traverso, P. 2004. Specifying and Analyzing Early Requirements in Tropos. Requirements Engineering, 9(2): 132-150.
7. González Jiménez, L. 2005. REERM: Reenhancing the entity-relationship model. Data & Knowledge Eng, In Press, Corrected Proof, Available online 13 June 2005, (http://www.sciencedirect.com/science/article/B6TYX-4GCWY60-1/2/5468b3374f3b31d93773e68fc0cc3d25).
8. Gregersen, H. and Jensen, C.S. et al (eds.). 1998. The Consensus Glossary of Temporal Database Concepts - February 1998 Version. In O. Etzion, S. Jajodia, and S. Sripada (eds.), Temporal Databases: Research and Practice, LNCS 1399, Springer-Verlag: 367-405.
9. Gregersen, H. and Jensen, C.S. 1999. Temporal Entity-Relationship Models - A Survey. IEEE Transaction on Knowledge and Data Engineering, 11(3): 464–497.
10. Greenspan, S., Mylopoulos, J. and Borgida, A. 1986. A requirements modeling language and its logic. Information Systems, 11(1): 9-23.
11. Herbst, H., Knolmayer, G., Myrach, T., Schlesinger, M. 1994. The Specification of Business Rules: A Comparison of Selected Methodologies. In A.A. Verrijn-Stuart, T. W. Olle (eds.), Methods and Associated Tools for the Information System Life Cycle, Elsevier:29-46.
12. Herbst, H. 1996. Business Rules in Systems Analysis: a Meta-Model and Repository System. Information Systems, 21(2): 147-166.
13. Jensen, C.S. and Snodgrass, R.T. 1994. Semantics of Time-Varying Information. Information Systems, 19(4): 33-54.
14. Jensen, C.S., Soo, M.and Snodgrass, R.T. 1994. Unifying temporal data models via a conceptual model. Information Systems, 9(7): 513–547.
15. Kardasis, P. and Loucopoulos, P. 2004. Expressing and organising business rules. Information and Software Technology, 46(11): 701-718.
16. Mylopoulos, J. 1998. Information Modeling in the Time of the Revolution. Information Systems, 23 (3/4): 127-155.
17. Owei, V. and Navathe, S.B. 2001. Enriching the conceptual basis for query formulation through relationship semantics in databases. Information Systems, 26(6): 445-475.
18. Skyt, J., Jensen, C.S. and Mark, L. 2003. A foundation for vacuuming temporal databases. Data and Knowledge Engineering, 4(1): 1-29.
19. Sowa, J.F. 2000. Knowledge Representation: Logical, Philosophical, and Computational Foundations. Brooks Cole Publishing Co.

20. Wagner, G. 2003. The Agent-Object-Relationship metamodel: towards a unified view of state and behavior. Information Systems, 28(5): 475-504.
21. Wan-Kadir, W. M. N. and Loucopoulos, P. 2004. Relating evolving business rules to software design. Journal of Systems Architecture, 50(7): 367-382.
22. Wu, Y., Jajodia, S. and Wang, X.S. 1998. Temporal Database Bibliography Update. In O. Etzion, S. Jajodia, and S. Sripada (eds.), Temporal Databases: Research and Practice, LNCS 1399, Springer-Verlag: 338-366.

Precise Modeling and Verification of Topological Integrity Constraints in Spatial Databases: From an Expressive Power Study to Code Generation Principles

Magali Duboisset[1], François Pinet[1], Myoung-Ah Kang[2], and Michel Schneider[2]

[1] Cemagref, Clermont Ferrand, France
{magali.duboisset, francois.pinet}@cemagref.fr
[2] Laboratory of Computer Science, Modeling and System Optimisation (LIMOS), Blaise Pascal University, Clermont Ferrand, France
{kang, schneider}@isima.fr

Abstract. Recent works underline that the integration of topological relationships into the Object Constraint Language (OCL) is an important field of investigation. The final goal is to provide an expressive language adapted to precisely model alphanumerical and topological constraints. In order to reach this goal, the present paper focuses on the integration of the 9 Intersection Method (9IM) into OCL. We show that this OCL+9IM language is especially suitable for the specification of topological constraints implying composite spatial objects. The expressive power of the language is also studied from a spatial point of view, and the SQL code generation from OCL+9IM expressions is considered. An important validation is related to the use of the language in the context of agricultural information systems.

1 Introduction

The specification of topological integrity constraints in spatial databases is an important task. Indeed, the challenge is to specify precisely the topological relationships between spatial objects but also to help to maintain databases consistent in identifying objects that don't verify the specified topological relationships.

The formalization of spatial relationships is an active research field and numerous works deal with the recognition of pertinent topological relationships. The core models in this domain are the Calculus Based Model (CBM) [5], the 9-Intersections Method (9IM) [9] and the Region Connection Calculus (RCC) [6]. All these approaches satisfy the requirements that they provide a sound and complete set of topological relationships between two spatial objects. These topological relationships have a wide range of applications. In the context of databases, CBM has been integrated into the Structured Query Language (SQL) of PostGres [4], and 9IM constitutes the basis of Oracle Spatial SQL [14].

In parallel of the spatial relationships definition, an interesting investigation field is related to the modeling of topological integrity constraints in databases. The purpose of these constraints is to check the quality of spatial data and to monitor the consistency of information. This paragraph gives an example of concrete database-oriented topological constraints; it is based on the consistency between database associations and spatial data. In the conceptual model of figure 1, each town hall building is associated to a town by the relationship "`postal_code`". In practice, this association corresponds to a foreign key in the relational database physical schema. This is illustrated by the corresponding physical schema of figure 2. An interesting topological integrity constraint could be "each town hall building *b* associated to a town *t* (by `postal_code`) must be spatially inside *t*". For example, in figure 2, the buildings b2 and b3 are associated (by the foreign key) to the town named "Issoire", and consequently the geometry of the buildings b2 and b3 must be inside the geometry of the town "Issoire".

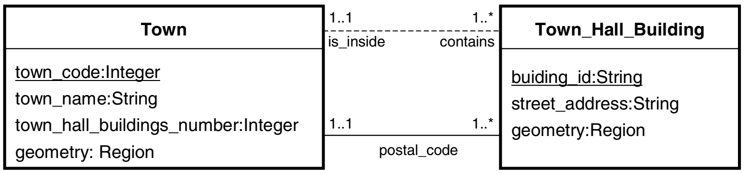
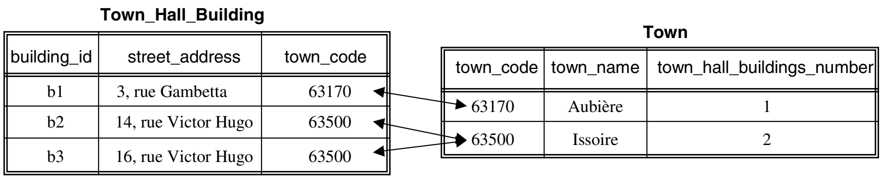


Fig. 1. "Town hall building" example



Town_Hall_Building

building_id	street_address	town_code
b1	3, rue Gambetta	63170
b2	14, rue Victor Hugo	63500
b3	16, rue Victor Hugo	63500

Town

town_code	town_name	town_hall_buildings_number
63170	Aubière	1
63500	Issoire	2

Fig. 2. Instance example of the "town hall building" physical schema

Currently specific mechanisms to describe topological constraints are often used in the object-oriented formalisms dedicated to spatial database design. In [1,10,11,15], the topological constraints are usually represented by a special relationship in class diagrams that describe database conceptual schemas. In these propositions, binary relationships are drawn between two classes in order to define that two types of data must verify a specific topological integrity constraint. This method is illustrated in figure 1 by the relationship "`contains...is_inside`" between town hall building and town classes; according to the multiplicity, this example indicates that: a) a town hall building is inside one town and b) a town contains one or several buildings composing a town hall. This relationship is not an association of the database; in fact, it corresponds to a topologic constraint. This type of relationships can be useful to express end-user basic constraints but it remains too limited to specify numerous real topological integrity constraints notably topological constraints depending on database associations. For instance, the "town hall building" constraint presented in the paragraph below, cannot be expressed by this type of modeling because the "town hall building" constraint depends on the association "`postal_code`" (i.e. there exists a specific topological relationship between a town hall building and a town if these two objects are associated by `postal_code`).

Thus, the non-ambiguous formalization of concrete topological constraints requires a specific database-oriented constraint language. Consequently, this paper underlines the needs to use an expressive formal language to describe precisely topological constraints and to generate automatically reliable constraint checking mechanisms inside databases. From a practical point of view, it is relevant to use only one language to express both topological and purely alphanumerical constraints. Thus, the main idea presented in this paper is to adapt an existing constraint language in order to offer the capability to express topological constraints.

At present, the constraint language par excellence is the Object Constraint Language (OCL) [12,17]. It is an important part of the Unified Modeling Language (UML) which is the standard formalism for information systems design, accepted both by the industrial domain and the scientific community. OCL has been originally developed by IBM and the standard is currently maintained by the Object Management Group. A growing number of information system designers use this constraint language in complement of their class diagram models. Moreover, OCL provides the capability to express easily constraints on database associations by using the concept of “navigation”. OCL is suitable for information systems engineers and is especially dedicated to formal constraints specification without side effects. Moreover, several theoretical and practical tools have been developed to convert an OCL constraint into a SQL query that checks if the OCL constraint is satisfied or not. OCL is a language especially dedicated to the constraints modeling and consequently, it is really easier for database designers to write integrity constraints in OCL than to write directly the corresponding SQL queries; in fact, OCL provides means to naturally describe constraints. For all these reasons, and as mentioned in [10], it seems important to precisely study the integration of spatial features into OCL in order to specify topological constraints.

A final goal of the work proposed in the present paper is to allow designers to specify topological constraints in OCL, independently of the platforms, and then to generate equivalent integrity checking mechanisms into different relational DBMS. In this context, another advantage of OCL is that it can be considered as a platform independent language. Indeed, all OCL specifications are expressed at a conceptual level i.e. the same specification level as the one presented in figure 1.

An important approach to characterize topological relationships rests on the 9-Intersection Method. Thus, this paper aims at extending OCL with 9IM. The paper is organized as follows. After a short overview of OCL (section 2), we propose the integration of the 9IM into OCL. We show that the produced language is especially suitable for the checking of topological relationships on composite spatial objects (section 3). We also assess the expressive power of OCL with 9IM from a topological point of view (section 4). An existing software named OCL2SQL has been extended in order to provide a reliable code generation tool dedicated to produce database triggers from topological constraints written in OCL with 9IM (section 5). An important validation of the works is related to the use of the language in the context of agricultural information systems (section 6).

2 Overview of OCL

The Object Constraint Language provides a framework for precisely defining constraints on a UML model in a formal way. OCL is textual and integrates several concepts issued from classical object-oriented languages. OCL is used to specify invariant i.e. a condition that "must be true for all instances of a class at any time" [17]. The examples presented in this section are based on figure 1.

Example 1. The following constraint specifies that the attribute named `town_hall_buildings_number` must be lower than 100.

```
context Town inv:
 self.town_hall_buildings_number < 100
```

In this example, `self` denotes an instance of the `Town` class i.e. the class declared in the "`context`". The OCL constraint must be "true" for each `Town` instance (i.e. each `self`). □

Example 2. OCL provides a functionality to count the number of instances. For example, the next expression specifies the following invariant: "the number of towns is lower or equal than the number of town hall buildings".

```
Town.allInstances()->size() <=
Town_Hall_Building.allInstances()->size()
```

The `allInstances` function returns a collection containing all the instances of a class. For example, "`Town.allInstances()`" returns the collection containing all `Town` instances. The "`->size()`" function returns the elements number of a collection. □

Example 3. More complex constraints can be built by using navigations along the associations between classes. In figure 1, the UML association "`postal_code`" is used in order to link `Town_Hall_Building` with `Town`. The next constraint illustrates the use of navigation in OCL by defining that for each `Town` instance `I`, the `town_hall_buildings_number` attribute is equal to the number of `Town_Hall_Building` instances associated to `I` by "`postal_code`". For instance, in figure 2, the `town_hall_buildings_number` attribute value of the town "Issoire" is equal to 2 because two buildings (b2 and b3) are associated to "Issoire".

```
context Town inv:
 self.town_hall_buildings_number
 = self.Town_Hall_Building->size()
```

Thus, `self` notation represents an instance of the `Town` class and the expression "`self.Town_Hall_Building`" returns a collection containing all the `Town_Hall_Building` instances associated to `self` by the relationship "`postal_code`". The function "`->size()`" returns the size of this collection. □

Example 4. Universal and existential quantifiers are denoted in OCL by `forAll` and `exists`. The logical implication can be expressed by `implies`. The next expression exemplifies these functionalities by specifying that each town has a proper town code. Let `t1` and `t2` be two towns. If `t1` and `t2` are not the same town then their town codes must be different.

```
Town.allInstances->forAll(t1,t2|
t1<>t2  implies  t1.town_code<>t2.town_code)
```
□

3 Integrating 9IM into OCL

3.1 Overview of 9IM

In 9IM, the classification of topological relationships is based on the intersection of the boundaries, the interiors and the exteriors of two spatial objects [9]. Each spatial object can be a point, a line or a simple region. The result of the intersection may be empty ($\varnothing$) or not ($\neg\varnothing$). A°, ∂A and A^- denote the interior, the boundary and the exterior of a spatial object A. Each topological relationship between two spatial objects A and B is represented by a 3x3 matrix; each matrix corresponds to a combination of intersections between A°, ∂A, A^- and B°, ∂B, B^-:

$$M = \begin{array}{ccc} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{array}$$

There are $2^9 = 512$ theoretical matrixes for two spatial objects but inconsistent matrixes can be removed i.e. matrixes corresponding to impossible cases. For instance, there are only 8 possible matrixes for two simple regions i.e. 8 possible relationships (see figure 3) [9].

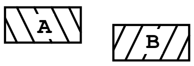
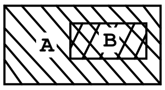
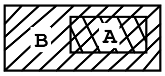
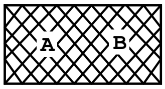
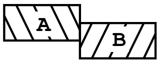
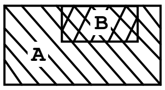
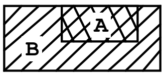
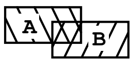
 $\begin{matrix} \varnothing & \varnothing & \neg\varnothing \\ \varnothing & \varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *disjoint*, *B*⟩**	 $\begin{matrix} \neg\varnothing & \neg\varnothing & \neg\varnothing \\ \varnothing & \varnothing & \neg\varnothing \\ \varnothing & \varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *contains*, *B*⟩**	 $\begin{matrix} \neg\varnothing & \varnothing & \varnothing \\ \neg\varnothing & \varnothing & \varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *inside*, *B*⟩**	 $\begin{matrix} \neg\varnothing & \varnothing & \varnothing \\ \varnothing & \neg\varnothing & \varnothing \\ \varnothing & \varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *equal*, *B*⟩**
 $\begin{matrix} \varnothing & \varnothing & \neg\varnothing \\ \varnothing & \neg\varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *meet*, *B*⟩**	 $\begin{matrix} \neg\varnothing & \neg\varnothing & \neg\varnothing \\ \varnothing & \neg\varnothing & \neg\varnothing \\ \varnothing & \varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *covers*, *B*⟩**	 $\begin{matrix} \neg\varnothing & \varnothing & \varnothing \\ \neg\varnothing & \neg\varnothing & \varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *coveredBy*, *B*⟩**	 $\begin{matrix} \neg\varnothing & \neg\varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \\ \neg\varnothing & \neg\varnothing & \neg\varnothing \end{matrix}$ **⟨*A*, *overlap*, *B*⟩**

Fig. 3. 8 possible topological relationships between two simple regions [9]

3.2 OCL$_{9IM}$

In this section we propose to integrate into OCL, the 8 relationships described in figure 3. We also propose to use the OCL set-based operations to decompose composite geometries. We call the extended language, OCL$_{9IM}$. The purpose is to enable database designers to model precise topological constraints on a database schema modeled with UML. In this paper, we investigate the integration of relationships between regions.

Definition 1. Region abstract model
A simple region is a closed connected point set without hole in a 2-dimensional space R^2. A composite region is a set $CR = \{R_1, \ldots, R_i, \ldots, R_n\}$ where R_i is a simple region also called “part” of *CR*. We define that: $\forall i \neq j$, $R_i^\circ \cap R_j^\circ = \emptyset$. □

Definition 2. Database schema with regions
The minimal set of concepts required to model a database conceptual schema with regions in UML is composed of the following entity-relationship notations: firstly, classes with attributes, and secondly, associations with multiplicities. An object identifier of a class *C* is the smallest set of attributes that identifies uniquely an instance of *C*. In this paper, the attribute names of an object identifier appear underlined in classes. Each attribute has a type; `Region` is the simple region type and `Set(Region)` is the composite region type. If a class *C* contains an attribute *a* having the type `Region` or `Set(Region)`, *C* is a spatial class and *a* is a geometry attribute. □

Definition 3. OCL$_{9IM}$
The language OCL$_{9IM}$ is OCL in which 8 new operations are integrated; one operation for each 9IM topological relationship on regions. The general form of these operations is: `A->topo_relationship(B)`

Thus, `topo_relationship` can be: `disjoint`, `contains`, `inside`, `equal`, `meet`, `covers`, `coveredBy`, `overlap`. We define that `A` and `B` are the parameters of the operations. The type of `A` and `B` must be `Region`. These operations return true or false (a boolean) depending on whether the topological relation between `A` and `B` is true or false. □

Example 5. Considering the constraint example presented in introduction: “each town hall building *b* associated to a town *t* must be spatially inside *t*” (see figure 1 and 2). This constraint can be easily expressed in OCL$_{9IM}$ as follows.

```
context Town_Hall_Building inv:
  self.geometry->inside(self.Town.geometry)
```

The constraint must be satisfied for each `Town_Hall_Building` instance (denoted by `self`). The expression “`self.Town`” returns the `Town` instance associated to `self` by “`postal_code`”. □

Definition 4. The 9IM relationships integrated into OCL require non-composite regions. But a composite region can be viewed as a set of regions. Thus, we define that in OCL$_{9IM}$, standard set-based OCL operations can be applied on a composite region in order to “decompose” it into several simple regions. Then, it becomes possible to apply the 9IM relationships on the produced simple regions (i.e. the parts). Thus, standard set-based OCL operations such as `size`, `forAll`, `exists` or `select` [12,17] can be applied on each attribute having the `Set(Region)` type.

The general syntax is: `geometry_attribute->set_based_operation(...)` □

In fact, it becomes possible to check topological relations involving composite geometries by integrating the eight 9IM relationships into OCL and in using the standard OCL operations on sets. For instance, in the conceptual model of figure 4, the geometry attributes of the `Downtown_Buildings_Lot` class and of the `Downtown_Area` class have the `Set(Region)` type. In other words, each of these attributes stores a composite region. Consequently, as presented in definition 4, set-based OCL operations can be applied on these attributes. This is illustrated by the next constraint (based on figure 4).

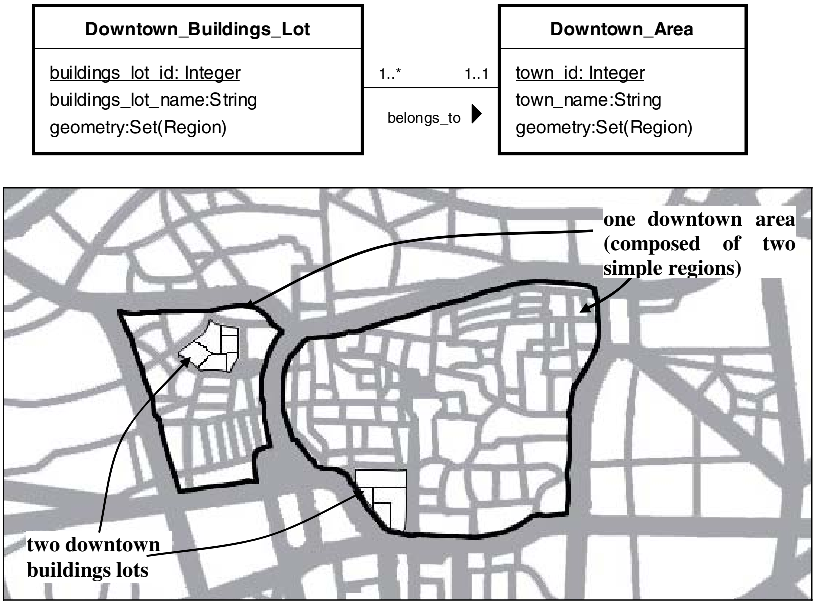

Downtown_Buildings_Lot
buildings_lot_id: Integer
buildings_lot_name:String
geometry:Set(Region)
1..*
1..1
belongs_to
Downtown_Area
town_id: Integer
town_name:String
geometry:Set(Region)
one downtown area (composed of two simple regions)
two downtown buildings lots

Fig. 4. "Downtown building lot" example

Example 6. Let *L* be a downtown building lot and let *D* be a downtown area. If *L* and *D* are associated by "`belongs_to`" then for each part `b` of *L*, there must exist a part `d` of *D* such as (`b` is inside `d`) or (`b` is covered by `d`). This constraint is written as follows in OCL$_{9IM}$.

```
context Downtown_Building_Lot inv:
 self.geometry->forAll(b| self.Downtown_Area.geometry->
 exists(d| b->inside(d) or b->coveredBy(d)) )
```

`geometry` attributes have the `Set(Region)` type and consequently, `b` and `d` have the `Region` type. □

As presented in example 7, we can, also specify the number of parts implied in a topological relationship.

Example 7. This constraint uses the relationship “each part of `A` meets two parts of `B`”.

```
context Class_1 inv:
  self.A->
  forAll(A_i| Class_2.allInstances()->forAll(I| I.B->
  select(B_j| A_i->meet(B_j))->size() = 2) )
```

The `Class_1` (resp. `Class_2`) has an attribute named `A` (resp. `B`). The type of the attributes `A` and `B` is `Set(Region)`. In the example, the OCL operation “`B->select(c)`” returns all elements (i.e. all parts) of the `B` attribute value that satisfy the condition `c`. □

3.3 OCL$_{9IM+ADV}$

In order to facilitate the specification of constraints on composite geometries, we propose to integrate into OCL$_{9IM}$, new operations based on the adverb model (ADV) presented in [3]. The ADV method provides interesting concepts to express easily and intuitively topological relationships between two composite regions. It offers the possibility to add an adverb to each of the 8 classical relationships presented in section 3.1. The logic-based semantics of the 7 adverbs proposed in [3] and an example of their use are presented in table 1. In this table, *topo_relationship* denotes a 9IM relationship between two simple regions (*disjoint*, *contains*, *inside*, *equal*, *meet*, *covers*, *coveredBy*, *overlap*). *topo_relationship$_{rev}$* is the converse relationship of a *topo_relationship*, in the case of contains/inside and covers/coveredBy. For the other relations, *topo_relationship$_{rev}$* = *topo_relationship*.

Moreover, 9IM+ADV denotes the relationships on composite regions that it is possible to express by adding each of the seven presented adverbs to each of the eight 9IM relationships.

Definition 5. OCL$_{9IM+ADV}$
We define OCL$_{9IM+ADV}$ as follows. The language OCL$_{9IM+ADV}$ integrates new operations into OCL$_{9IM}$. The general form of these operations is:

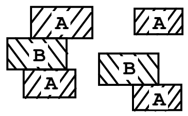
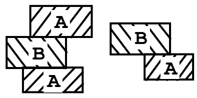
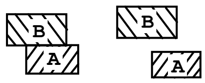
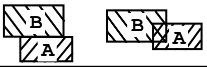
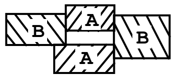
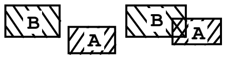
```
A->topo_relationship("adverb",B)
```

Thus, `topo_relationship` can be: `disjoint`, `contains`, `inside`, `equal`, `meet`, `covers`, `coveredBy`, `overlap`. The correct values for the `"adverb"` parameter are: `"mostly"`, `"mostlyRev"`, `"completely"`, `"partially"`, `"occasionally"`, `"entirely"`, `"never"`. The type of `A` and `B` must be `Set(Region)`. These operations return true or false (a boolean) depending on whether the 9IM+ADV topological relation between `A` and `B` is true or false. If `A` or `B` is a set which contains no element, the operation returns false. □

Example 8. The constraint of example 6 can be expressed more directly in OCL$_{9IM+ADV}$ without using `forAll` and `exists` operators.

```
context Downtown_Building_Lot inv:
  self.geometry->inside("mostlyRev",self.Downtown.geometry) or
  self.geometry->coveredBy("mostlyRev",self.Downtown.geometry)
```
□

Table 1. Semantics of the seven adverbs [3]

Logic-based semantics of the seven adverbs	Examples with *meet*
mostly - A *mostly topo_relationship* B $\forall j \in 1..m, \exists i \in 1..n \mid \langle A_i, \mathit{topo\_relationship}, B_j \rangle$	
mostly	
completely - A *completely topo_relationship* B $(\forall j \in 1..m, \exists i \in 1..n \mid \langle A_i, \mathit{topo\_relationship}, B_j \rangle \wedge$ $(\forall i \in 1..n, \exists j \in 1..m \mid \langle B_j, \mathit{topo\_relationship}_{rev}, A_i \rangle)$	
partially - A *partially topo_relationship* B $\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \mathit{topo\_relationship}, B_j \rangle \wedge$ $(\forall r \in 1..n, \forall s \in 1..m \mid \langle A_r, \mathit{disjoint}, B_s \rangle$ $\vee \langle A_r, \mathit{topo\_relationship}, B_s \rangle)$	
occasionally - A *occasionaly topo_relationship* B $\exists i \in 1..n, \exists j \in 1..m \mid \langle A_i, \mathit{topo\_relationship}, B_j \rangle$	
entirely - A *entirely topo_relationship* B $\forall i \in 1..n, \forall j \in 1..m, \mid \langle A_i, \mathit{topo\_relationship}, B_j \rangle$ $\wedge \langle B_j, \mathit{topo\_relationship}_{rev}, A_i \rangle$	
never - A *never topo_relationship* B $\forall i \in 1..n, \forall j \in 1..m, \mid \neg \langle A_i, \mathit{topo\_relationship}, B_j \rangle$	

4 Detailed Study of Expressive Power

We show in the previous section that the integration of 9IM into OCL provides a language enabling designers to specify relationships on composite regions. An important work is to study precisely the expressive power of OCL_{9IM} and $OCL_{9IM+ADV}$. In other words, what is precisely the set of relationships that the proposed languages can distinguish? Firstly, we compare the expressive power of OCL_{9IM} and $OCL_{9IM+ADV}$ (section 4.1). In a second step, we investigate a method to evaluate the expressive power of the proposed languages (sections 4.2).

4.1 Expressivity Comparison Between OCL_{9IM} and $OCL_{9IM+ADV}$

ADV provides an excellent abstraction for designers in order to specify topological constraints implying composite regions. We demonstrate in this section that all $OCL_{9IM+ADV}$ expressions can be rewritten into OCL_{9IM} without semantic loss. In fact, we show that OCL_{9IM} and $OCL_{9IM+ADV}$ have exactly the same power of expression.

Table 2. OCL$_{9IM+ADV}$ to OCL$_{9IM}$. `A` and `B` have the `Set(Region)` type.

mostly – `A->topo_relationship("mostly",B)` can be rewritten as follows: `B->forAll(B_j|A->exists(A_i|A_i->topo_relationship(B_j)))`
mostly$_{rev}$ – `A->topo_relationship("mostlyRev",B)` can be rewritten as follows: `A->forAll(A_i|B->exists(B_j|A_i->topo_relationship(B_j)))`
completely – `A->topo_relationship("completely",B)` can be rewritten as follows: `B->forAll(B_j|A->exists(A_i|A_i->topo_relationship(B_j))) and A->forAll(A_i|B->exists(B_j|B_j->topo_relationship`$_{rev}$`(A_i)))`
partially – `A->topo_relationship("partially",B)` can be rewritten as follows: `A->exists(A_i|B->exists(B_j|A_i->topo_relationship(B_j))) and A->forAll(A_r|B->forAll(B_s|A_r->topo_relationship(B_s) or A_r->disjoint(B_s)))`
occasionally – `A->topo_relationship("occasionally",B)` can be rewritten as follows: `A->exists(A_i|B->exists(B_j|A_i->topo_relationship(B_j)))`
entirely – `A->topo_relationship("entirely",B)` can be rewritten as follows: `A->forAll(A_i|B->forAll(B_j|A_i->topo_relationship(B_j) and B_j->topo_relationship`$_{rev}$`(A_i)))`
never – `A->topo_relationship("never",B)` can be rewritten as follows: `A->forAll(A_i|B->forAll(B_j|not A_i->topo_relationship(B_j)))`

Theorem 1. OCL$_{9IM}$ $\subset$ OCL$_{9IM+ADV}$. All constraints expressed in OCL$_{9IM}$ can be also expressed in OCL$_{9IM+ADV.}$

Demonstration**.** Trivial. OCL$_{9IM+ADV}$ is based on OCL$_{9IM}$. OCL$_{9IM+ADV}$ simply adds new operations to OCL$_{9IM}$. All constraint expressions that belong to OCL$_{9IM}$ also belong to OCL$_{9IM+ADV}$. □

Theorem 2. OCL$_{9IM+ADV}$ $\subset$ OCL$_{9IM}$. All constraints expressed in OCL$_{9IM+ADV}$ can be also expressed in OCL$_{9IM}$.

Demonstration (Sketch). An expression using a 9IM+ADV operation of OCL$_{9IM+ADV}$ can be rewritten in an expression of OCL$_{9IM}$ using 9IM operations and OCL set-based operations. See table 2. This mapping is based on a conversion between the logic-based specification of table 1 and OCL expressions. Indeed, this conversion becomes possible thanks to the integration of 9IM into OCL and to the use of set-based OCL operations on geometries, as defined in section 3.2. □

Corollary. OCL$_{\text{9IM}}$ $\equiv$ OCL$_{\text{9IM+ADV}}$
The expressive powers of OCL$_{\text{9IM}}$ and OCL$_{\text{9IM+ADV}}$ are equivalent. □

4.2 *n*×*m* Matrix Approach

An approach to enumerate relationships between composite regions is described in [4]. The main principle of this approach is to consider a *n*×*m* matrix where *n* is the parts number of a composite region *A*, and *m* is the parts number of a composite region *B*; the rows of the matrix correspond to the parts of *A* and the columns of the matrix correspond to the parts of *B*. The matrix describes all relationships between parts of *A* and parts of *B*. More precisely, a topological scene is represented by means of a matrix in which the element in position (*i*,*j*) gives the relationship between the *i*-th row's simple region and the *j*-th column's simple region. Figure 5 and table 3 exemplify this type of matrixes.

In OCL$_{9IM}$, the parts of a composite region cannot be named or numbered as in the $n\times m$ matrixes, but it is possible to define the number of rows or columns implied in a specific topological relationship i.e. the number of parts of A or B that are implied in a specific 9IM relationship. For instance, as presented in the matrix of table 3, two parts of A (i.e. two rows) are implied in the "meet" relationship and one part of A (i.e. one row) is implied in the "overlap" relationship.

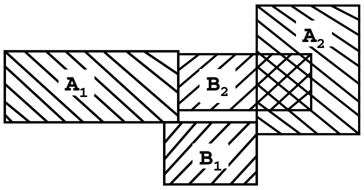


Fig. 5. A topological configuration between two composite regions A and B

Table 3. The $n\times m$ topology matrix for the composite regions of Figure 5 (O=*overlap*, M=*meet*)

	B_1	B_2
A_1	M	M
A_2	M	O

Theorem 3. Let A and B be two composite regions. The number of parts of A (resp. B) that are implied in a specific 9IM relationship with parts of B (resp. A) can be defined in OCL$_{9IM}$. The general form of the OCL$_{9IM}$ constraint describing this parts number is presented in definition 6. □

Definition 6. The general form of the OCL$_{9IM}$ expression describing for each *relationship$_p$*, the number of parts of `A` that are implied in *relationship$_p$* with `B` is the following.

```
A->select( part_of_A | B->exists(part_of_B |
     part_of_A->relationship_1(part_of_B)) )->size() = s_1
 and
 ...
A->select( part_of_A | B->exists(part_of_B |
     part_of_A->relationship_p(part_of_B)) )->size() = s_p
 ...
 and
A->select( part_of_A | B->exists(part_of_B |
     part_of_A->relationship_z(part_of_B)) )->size() = s_z
```

`A` and `B` have the `Set(Region)` type. For example, `A` and `B` can be "`self.geometryA`" and "`self.association.geometryB`".

relationship$_p$ is a 9IM relationship operation name (`disjoint`, `contains`, `inside`, `equal`, `meet`, `covers`, `coveredBy`, `overlap`) and s_p is the corresponding parts number for the p-th relationship. `A` and `B` can be inverted in the OCL expressions, in order to describe the number of parts of `B` that are implied in a 9IM relationship with parts of A. Notice that topological relationships between parts of a same composite region can be also considered when A and B have the same attribute value. □

5 Code Generation

An important challenge is to reduce the gap between the conceptual description of constraints and their implementation inside a spatial database. Thus, a goal of the works proposed in the present paper is to enable designers to specify topological constraints in OCL independently of the platforms, and then to generate equivalent integrity checking mechanisms into different relational DBMS. This is the reason why we set up an extension of the tool named OCL2SQL to translate OCL_{9IM} constraints into database triggers. The corresponding architecture is schematized in figure 6. OCL_{9IM} is a platform independent language allowing the expression of constraints at a high abstraction level.

Indeed, we extended an existing tool named OCL2SQL in order to produce a topological integrity checking mechanism in spatial databases. The open source OCL2SQL program is a powerful generator [7,8]; it offers the capability to generate automatically from an OCL expression *c*, a SQL query selecting all data that don't satisfy *c*. Once integrated inside a database trigger (on data insertion, deletion and update), the query provides guards that guarantee the consistency of databases. Indeed, when a data modification occurs, the trigger checks if the generated SQL query returns tuples; if it's not the case then the update is accepted, else the data modification is rejected. By this technique, it becomes impossible to insert data that violate a constraint.

We extended the standard conversion rules of OCL2SQL in order to translate OCL_{9IM} constraints into Spatial SQL.

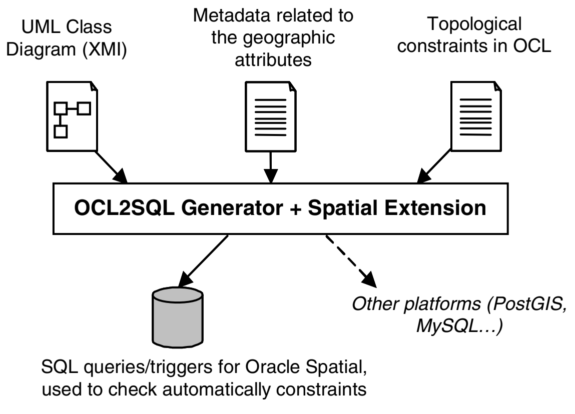

UML Class Diagram (XMI)
Metadata related to the geographic attributes
Topological constraints in OCL
OCL2SQL Generator + Spatial Extension
Other platforms (PostGIS, MySQL...)
SQL queries/triggers for Oracle Spatial, used to check automatically constraints

Fig. 6. From the topological constraints specification to integrity checking mechanisms. Metadata are related to information on geographic data (coordinate system...). OCL2SQL has been developed by [7,8].

5.1 Code Generation from OCL$_{9IM}$ Constraints Implying Simple Regions

We included the 9IM operations into OCL2SQL by adding the new proposed OCL syntax and by providing in a first step, the automatic generation for the Spatial SQL supported by Oracle. To be able to generate code for a specific DBMS, a direct mapping between the 9IM operations of OCL$_{9IM}$ and SQL operations must be possible. For example, concerning the relationships between two simple regions, we defined the possible mapping for Oracle SQL [14] and OpenGIS SQL [13] (table 4).

Table 4. Mapping rules from OCL$_{9IM}$ operations to Oracle Spatial SQL and OpenGIS SQL

OCL$_{9IM}$	**Oracle Spatial SQL**	**OpenGIS SQL**
disjoint	not ANYINTERACT	Disjoint(A, B)
contains	CONTAINS	Relate(A, B, '111FF1FF1')
inside	INSIDE	Relate (A, B, '1FF1FF111')
equal	EQUAL	Equals(A, B)
meet	TOUCH	Touches(A, B)
covers	COVERS	Relate(A, B, '111F11FF1')
coveredBy	COVEREDBY	Relate(A, B, '1FF11F111')
overlap	OVERLAPBDYINTERSECT	Overlaps(A, B)

In table 4, the *within* OpenGIS predicate includes *equal*, *coveredBy* and *inside* of 9IM. It can't be used for the mapping of the *inside* OCL$_{9IM}$ predicate. *covers* and *coveredBy* are also not defined in OpenGIS SQL. Thus we use the OpenGIS predicate *relate*. It takes as input a pattern matrix representing the set of acceptable values for the DE-9IM matrix (Dimensionally Extended 9IM) [13] for the two geometries on which it is formulated. The pattern matrix consists of a set of 9 pattern-values, one for each cell in the matrix. "F" means that the intersection for the cell is null, "1" that it is not null.

Example 9. Converting the constraint of example 5 into Spatial SQL. Firstly, a mapping between conceptual models of figure 1 and physical database schemas must be achieved. The corresponding physical schema is:

```
Town(town_code,town_name,town_hall_buildings_number,geometry)
Town_Hall_Building(building_id,street_address,geometry,
#town_code)
```

At the physical schema level, the `geometry` attribute type is `Region` in the two tables. The OCL$_{9IM}$ constraint of example 5 can be translated into Spatial SQL as follows (the queries select data that don't satisfy the constraint).

Oracle Spatial SQL:

```
select * from Town_Hall_Building self, Town T
where T.town_code = self.town_code
and not (MDSYS.SDO_RELATE(self.geometry, T.geometry,
'mask=INSIDE querytype=WINDOW')= 'true');
```

OpenGIS SQL:

```
select * from Town_Hall_Building self, Town T
where T.town_code = self.town_code
and not Relate (self.geometry, T.geometry, '1FF1FF111') = 1;   □
```

5.2 Code Generation from OCL$_{9IM}$ Constraints Implying Composite Regions

The use of attributes having the `Set(Region)` type in UML diagram can simplify the modeling of composite regions at a conceptual level [2,10,15]. However, as presented in this subsection, it is not straightforward to handle attributes having a `Set(Region)` type in relational physical schemas with SQL. Concerning the composite regions, we can consider two main types of physical schemas: one in which a spatial class is mapped into one table, and one in which a spatial class is mapped into two tables.

1) Mapping one spatial class with one table. If the target DBMS supports the `Set(Region)` type, a first method is to convert each spatial class of the conceptual model into only one table in the database; this table corresponds to the class itself and each composite geometry is stored in one geometry attribute value. In this case, the type of the geometry attribute of the physical schema is `Set(Region)`. Example 10 illustrates this type of physical schemas. This possibility is interesting but the storage of all the parts of a composite region in the same attribute could lead to practical difficulties notably concerning the access of the different parts in SQL. For example, in considering this data structure, writing a SQL query to compute the area of the smallest part of a composite region is difficult for the database users. This SQL query implies the use of a decomposition operation [13]. For these reasons, we don't investigate this first mapping in our work, and we advocate the use of the second mapping (i.e. from a spatial class to two tables).

Example 10. Mapping one spatial class with one table. In using this type of conversions, the physical database schema of the conceptual model presented in figure 4 is:

```
Downtown_Area(town_id,town_name,geometry)
Downtown_Buildings_Lot(buildings_lot_id,buildings_lot_name,
geometry,#town_id)
```

The type of the `geometry` attributes is `Set(Region)`. □

2) Mapping one spatial class with two tables. A spatial class of the conceptual model is converted into two tables in the physical schema of the database. The first table is the class itself. Each tuple of the second table stores a part of the regions. Thus, it becomes possible to easily reach every part of composite geometries with SQL thanks to the "part" table. Example 11 illustrates this type of physical schemas. We implemented into OCL2SQL the mapping rules related to the translation of the set-based OCL operations into SQL Spatial, in the case of the use of these operations on a composite region attribute. To understand this method, the application of these mapping rules to an attribute of "self" is illustrated in table 5. The techniques used for these mapping rules are similar to the ones used for the standard implementation of "forAll", "exists" and "select" in the original version of OCL2SQL.

Table 5. Using set-based OCL operations to decompose a composite geometry: mapping rules from OCL$_{9IM}$ to Oracle Spatial SQL and example of application to an attribute of “self”. Physical database schema for composite spatial data: `T(t_id,...)`, `T_Part(part_id, geo_part, #t_id)`. The `e` function translate an OCL expression into SQL.

```
OCL: self.composite_geo_attrib->forAll(x|bool_exp_with_x)
SQL: select * from T where not
               (not exists(select part_id from T_Part
               where T_Part.t_id=T.t_id
               minus
               select part_id from T_Part where e(bool_exp_with_x)))
```

```
OCL: self.composite_geo_attrib->exists(x|bool_exp_with_x)
SQL: select * from T where not
            (exists(select part_id from T_Part
            where T_Part.t_id=T.t_id
            intersect
            select part_id from T_Part where e(bool_exp_with_x)))
```

```
OCL: self.composite_geo_attrib->select(x|bool_exp_with_x)
SQL: select * from T where not
     (select part_id from T_Part
     where T_Part.t_id=T.t_id
     minus
     select part_id from T_Part where not e(bool_exp_with_x))
```

Example 11. Mapping one spatial class with two tables. In using this type of conversions, the physical database schema of the conceptual model presented in figure 4 is as follows.

```
Downtown_Area(town_id,town_name)
Downtown_Area_Part(part_id,geo_part,#town_id)
Downtown_Buildings_Lot(buildings_lot_id,buildings_lot_name,
#town_id)
Downtown_Buildings_Lot_Part(part_id,geo_part,#buildings_lot_id)
```

The `geo_part` attribute type is `Region`. In using the mapping rules, the OCL$_{9IM}$ constraint of example 6 can be translated into SQL as follows.

Oracle Spatial SQL:

```
select * from Downtown_Buildings_Lot self where not(
not exists ( select part_id from Downtown_Buildings_Lot_Part
DBL_Part_1 where
DBL_Part_1.buildings_lot_id=self.buildings_lot_id
minus select part_id from Downtown_Buildings_Lot_Part DBL_Part_2
where exists( select part_id from Downtown_Area_Part DA_Part_1
where DA_Part_1.town_id=self.town_id
intersect select part_id from Downtown_Area_Part DA_Part_2 where
MDSYS.SDO_RELATE(DBL_Part_2.geo_part, DA_Part_2.geo_part,
'mask=INSIDE querytype=WINDOW')= 'true' or
MDSYS.SDO_RELATE(DBL_Part_2.geo_part, DA_Part_2.geo_part,
'mask=COVEREDBY querytype=WINDOW')= 'true');
```

OpenGIS SQL:

```
select * from Downtown_Buildings_Lot self where not(
not exists ( select part_id from Downtown_Buildings_Lot_Part
DBL_Part_1 where
DBL_Part_1.buildings_lot_id=self.buildings_lot_id minus
```

```
select part_id from Downtown_Buildings_Lot_Part DBL_Part_2 where
exists( select part_id from Downtown_Area_Part DA_Part_1
where DA_Part_1.town_id=self.town_id intersect
select part_id from Downtown_Area_Part DA_Part_2 where
Relate(DBL_Part_2.geo_part, DA_Part_2.geo_part, '1FF1FF111')=1
or Relate(DBL_Part_2.geo_part, DA_Part_2.geo_part,
'111F11FF1')=1;                                                 □
```

The conversion from $OCL_{9IM+ADV}$ to Spatial SQL can be also considered by translating $OCL_{9IM+ADV}$ constraints into OCL_{9IM} constraints (see section 4.1).

6 Case Study in Agriculture

A first version of the spatial extension of the OCL2SQL code generator has been developed for Oracle Spatial. This first version allows the automatic generation of Spatial SQL queries from OCL_{9IM} constraints. In order to validate the spatial extension of OCL2SQL, a final goal is to use it in the Cemagref institute, during the iterative development process of an agricultural information system integrating a spatial database. Information of this database must be exported toward other systems, and consequently it is really important to avoid data inconsistencies before exporting information. Thus, in order to validate the development of this project, the purpose is to check with this tool, if the different beta versions of this system under-construction produce inconsistencies in the associated spatial database. The checking process is schematized in figure 7.

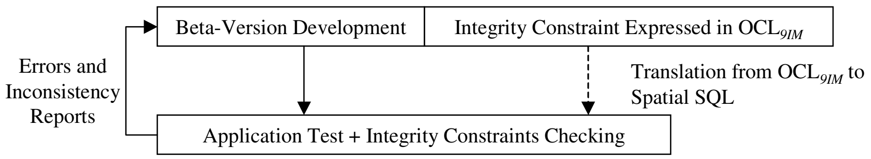

Beta-Version Development
Integrity Constraint Expressed in OCL9IM
Errors and Inconsistency Reports
Translation from OCL9IM to Spatial SQL
Application Test + Integrity Constraints Checking

Fig. 7. Example of development process with OCL_{9IM}

The purpose of the information system is to monitor agricultural spreading of sludge, and the associated database stores the traceability of agricultural practices. Indeed, in agriculture, the sewage sludge spreading is considered as a good way to recycle waste issued from sewage plants; this technique consists in depositing sludge directly on fields. This type of low cost practices gives the possibility not only to recycle waste, but also to fertilize the ground. In spite of its numerous advantages, the sewage sludge spreading must be monitored in order to avoid ground and waterway pollution. Indeed, too intensive practices could lead to an environmental deterioration. This could affect: a) areas that are close to the location where sewage sludge has been spread, and b) extended areas including hydrographical networks. This is the reason why a specific regulation has been defined e.g. for each farm, allowed spreading areas must be defined in order to indicate precisely where sewage sludge could be spread without risk. To facilitate the monitoring of these activities, farms have to record areas where spreading had finally been carried out. The concentration of sewage must also be carefully monitored and governmental institutions usually organize ground analysis in different locations. The final version of the information system will store all data related to the management and the monitoring of agricultural spreading practices. The main spatial data on which constraints can be applied concern: allowed spreading areas, parcels where spreading had finally been carried out, locations where the ground analysis is applied.

7 Conclusion and Perspectives

To sum up, the present paper proposes the integration of 9IM into OCL, in order to define precisely topological constraints in databases. We show that the produced language named OCL_{9IM} is especially suitable for the modeling of topological constraints on composite spatial objects (section 3.2).

In order to simplify the syntax of these constraints, we also introduce $OCL_{9IM+ADV}$ which corresponds to the integration of 9IM+ADV into OCL (section 3.3). We show that it's easier to express constraints on composite objects with $OCL_{9IM+ADV}$ than with OCL_{9IM}. Nevertheless, OCL_{9IM} and $OCL_{9IM+ADV}$ have exactly the same expressive power (section 4.1). In other words, the integration of 9IM into OCL also provides the possibility to express the 9IM+ADV relationships.

Section 4.2 studies a method to delimitate the expressive power of OCL_{9IM} from a topological point of view. The first proposition presented in this section opened up a new field of investigation related to the refinement of the OCL_{9IM} expressive power study.

An important final goal is to enable designers to specify topological constraints in OCL_{9IM} independently of the platforms, and then to generate equivalent integrity checking mechanisms into different relational DBMS (section 5). We extended OCL2SQL in order to translate OCL_{9IM} constraints into Spatial SQL. From a general point of view, OCL2SQL is a very interesting and flexible open source tool to experiment new database-oriented extensions of OCL.

Several first tests of Spatial SQL code generation have yielded very good results for the DBMS considered in our works (Oracle Spatial). An important validation of the works is related to the use of OCL_{9IM} during the development of agricultural information systems in the Cemagref institute (section 6).

This paper focuses on topological relationships between regions. In the future, we will generalize the proposed approach in order to also consider topological relationships between different types of geometries (points, lines, regions with holes). In order to reach this goal, we will study the comparison between our approach and the interesting model presented in [16]. Indeed, the authors of [16] propose unified semantics based on spatial intersection and spatial difference operations, in order to define relationships implying spatial composite objects having different spatial types. These works consider not only binary relationships but also *n*-ary relationships. Moreover, [18] is another reference to consider in the field of spatial relationships between heterogeneous set of geometries. These works provide different topological predicates for this type of relationships. Concerning the code generation tool, the current target platform is Oracle Spatial but other DBMS will be considered (PostGIS for instance). Another important field of investigation is also related to the development of a specific tool to help end-users to write easily OCL_{9IM} constraints.

References

1. Borges, K., Laender, A., Clodoveu, D.: Spatial Data Integrity Constraints in Object Oriented Geographic Data Modeling. In: Proc. of the Int. Symposium on Geographic Information System. ACM Press. USA (1999) 1-6
2. Brodeur J., Bédard Y., Proulx M.J.: Modelling Geospatial Application Databases using UML-based Repositories Aligned with International Standards in Geomatics. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (2000) 39-46
3. Claramunt C.: Extending Ladkin's Algebra on Non-convex Intervals towards an Algebra on Union-of Regions. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (2000) 9-14
4. Clementini E., Di Felice P., Califano G.: Composite Regions in Topological Queries. Information Systems, Vol.20(7). (1995) 579-594
5. Clementini E., Di Felice P., Oosterom P.: A Small Set of Formal Topological Relationships For End-User Interaction, Int. Symposium on Advances in Spatial Databases (SSD'93), Singapore (1993) 277-295
6. David A. Randell, Zhan Cui, Anthony G. Cohn: A Spatial Logic based on Regions and Connection, Int. Conference on Principles of Knowledge Representation and Reasoning (KR'92), USA (1992) 165-176
7. Demuth B., Hußmann H.: Using UML/OCL Constraints for Relational Database Design. Proc. of the Conference on the Unified Modelling Language, USA (1999) 598-613
8. Demuth B., Hußmann H., Loecher S.: OCL as a Specification Language for Business Rules in Database Applications. Proc. of the Conference on the Unified Modelling Language, USA (2001) 104-117
9. Egenhofer M., Franzosa R.: Point-Set Topological Spatial Relations. Int. Journal of Geographical Information Systems, Vol.5(2). (1991) 161-174
10. Friis-Christensen A., Tryfona N., Jensen C.: Requirements and Research Issues in Geographic Data Modeling. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (2001) 2-8
11. Kösters G., Pagel B., Six H.: GIS-Application Development with GeoOOA. Int. Journal of Geographical Information Science, Vol.11(4). (1997) 307-335
12. OMG: Unified Modeling Language: OCL, version 2.0. OMG Specification
13. OpenGIS: Simple Features Specification for SQL. OpenGIS Specification
14. Oracle Corp: Oracle Spatial: User's Guide and Reference. Oracle Documentation
15. Parent C., Spaccapietra S., Zimanyi E.: Spatio-Temporal Conceptual Models: Data Structures + Space + Time. Proc. of the Int. ACM Symposium on Advances in Geographic Information Systems, USA (1999) 26-33
16. Price R., Tryfona N., Jensen C.: Modeling Topological Constraints in Spatial Part-Whole Relationships. Proc. of the Int. Conference on Conceptual Modeling (ER'01), Japan (2001) 27-40
17. Schmid B., Warmer J., Clark T.: Object Modeling with the OCL: The Rationale Behind the Object Constraint Language. Springer Verlag (2002) 281p
18. Zhong Z., Jing N., Chen L., Wu Q.: Representing Topological Relationships Among Heterogeneous Geometry-Collection Features. Journal of Computer Science and Technology, Vol.19(3). (2004) 280-289

Topological Relationships Between Complex Lines and Complex Regions

Markus Schneider[1,⋆] and Thomas Behr[2]

[1] University of Florida,
Dept. of Computer & Information Science & Engineering,
Gainesville, FL 32611, USA
mschneid@cise.ufl.edu
[2] Fern Universität Hagen, Praktische Informatik IV,
58084 Hagen, Germany
thomas.behr@fernuni-hagen.de

Abstract. Topological relationships between spatial objects in the two-dimensional space have been investigated for a long time in a number of disciplines like artificial intelligence, cognitive science, linguistics, and robotics. In the context of spatial databases and geographical information systems, as predicates they especially support the design of suitable query languages for spatial data retrieval and analysis. But so far, they have only been defined for simplified abstractions of spatial objects like continuous lines and simple regions. With the introduction of *complex spatial data types* in spatial data models and extensions of commercial database systems, an issue arises regarding the design, definition, and number of topological relationships operating on these complex types. This paper first introduces a formally defined, conceptual model of general and versatile spatial data types for *complex lines* and *complex regions*. Based on the well known 9-intersection model, it then formally determines the complete set of mutually exclusive topological relationships between complex lines and complex regions. Completeness and mutual exclusion are shown by a proof technique called *proof-by-constraint-and-drawing*.

Keywords: Topological predicate, topological constraint rule, proof-by-constraint-and-drawing, complex spatial data type, 9-intersection model.

1 Introduction

For a long time, the study of topological relationships between objects in two-dimensional space has been a multi-disciplinary research issue involving disciplines like artificial intelligence, cognitive science, geographical information systems (GIS), linguistics, psychology, robotics, spatial database systems, and qualitative spatial reasoning. From a database and GIS perspective, their development has been stimulated by the necessity of formally defined topological predicates as filter conditions for spatial selections and spatial joins in spatial query languages and as a support for spatial data retrieval and analysis tasks.

⋆ The first author was partially supported by the National Science Foundation under grant number NSF-CAREER-IIS-0347574.

Topological relationships like *overlap*, *inside*, or *meet* describe purely qualitative properties that characterize the relative positions of spatial objects and are preserved under affine transformations. A restriction and shortcoming of current topological models is that topological relationships have so far only been determined for simplified abstractions of spatial objects like simple lines and simple regions. Simple lines are one-dimensional continuous features embedded in the plane with two end points, and simple regions are two-dimensional point sets topologically equivalent to a closed disc. Unfortunately, these simple geometric structures are insufficient to cover the variety and complexity of geographic reality. Universal and versatile type specifications are needed for (more) complex spatial objects that can be leveraged in many different applications. With regard to *complex lines*, we permit arbitrary, finite collections of one-dimensional curves, i.e., spatially embedded networks possibly consisting of several disjoint connected components, as line objects (e.g., to model the ramifications of the Nile Delta). With regard to *complex regions*, the two main extensions relate to separations of the exterior (holes) and to separations of the interior (multiple components). For example, countries (like Italy) can be made up of multiple components (like the mainland and the offshore islands) and can have holes (like the Vatican). Hence, a first goal of this paper is to give a formal definition of spatial data types for complex lines and complex regions.

With the integration of *complex spatial data types* into spatial type systems from a formal perspective as well as into GIS and spatial extension packages of commercial database systems from an application perspective, an issue arises regarding the design, definition, and number of topological relationships operating on these complex types. This is of interest simply from a theoretical point of view but has especially impact on the aforementioned disciplines and on spatial selections and spatial joins. Hence, a second goal is to explore and derive the possible topological relationships between all combinations of complex spatial data types. In this paper, we show the derivation mechanism for complex lines and complex regions on the basis of the well known 9-intersection model. For this purpose, we draw up collections of constraints specifying conditions for valid topological relationships and satisfying the properties of *completeness* and *exclusiveness*. The property of completeness ensures a full coverage of all topological situations on the basis of the 9-intersection model. The property of exclusiveness ensures that two different relationships cannot hold for the same two spatial objects.

The remainder of the paper is organized as follows: Section 2 discusses related work on complex lines, complex regions, and topological relationships. Section 3 formalizes the spatial concepts of complex lines and complex regions. Section 4 explains our general strategy, called the *Proof-By-Constraint-And-Drawing Method*, for deriving topological relationships from the 9-intersection model. As an example, Section 5 identifies all topological relationships between complex lines and complex regions. Finally, Section 6 draws some conclusions and discusses future work.

2 Related Work

In the past, numerous data models and query languages for spatial data have been proposed with the aim of formulating and processing spatial queries in databases and GIS [7]. *Spatial data types* (see [10] for a survey) like *point*, *line*, or *region* provide fundamental abstractions for modeling the structure of geometric entities, their relationships, properties, and operations. A few models [1,8,9,10] have been developed towards complex spatial objects. All these approaches allow multiple object components. In some approaches object components are allowed to intersect [1,9]. Some approaches are based on a finite geometric domain [8,10] whereas we define our data types in the infinite Euclidean plane.

Topological predicates have so far only been determined for *simple* object structures like continuous lines and simple regions. An important approach for characterizing them rests on the so-called 9-*intersection model* [3], which leverages point set theory and point set topology [6] as its formal framework. For example, a complete collection of 19 mutually exclusive topological relationships has been determined between a simple line and a simple region [4]. The model is based on the nine possible intersections of boundary (∂A), interior (A°), and exterior (A^-) of a spatial object A with the corresponding components of another object B. Each intersection is tested with regard to the topologically invariant criteria of emptiness and non-emptiness. The topological relationship between two spatial objects A and B can be expressed by evaluating the well known intersection matrix in Table 1. For this matrix $2^9 = 512$ different configurations are possible from which only a certain subset makes sense depending on the *definition* and *combination* of spatial data types. For each combination of spatial types this means that each of its predicates is associated with a unique intersection matrix so that all predicates are mutually exclusive and complete with regard to the topologically invariant criteria of emptiness and non-emptiness.

Table 1. The 9-intersection matrix

$$\begin{pmatrix} A^\circ \cap B^\circ \neq \varnothing & A^\circ \cap \partial B \neq \varnothing & A^\circ \cap B^- \neq \varnothing \\ \partial A \cap B^\circ \neq \varnothing & \partial A \cap \partial B \neq \varnothing & \partial A \cap B^- \neq \varnothing \\ A^- \cap B^\circ \neq \varnothing & A^- \cap \partial B \neq \varnothing & A^- \cap B^- \neq \varnothing \end{pmatrix}$$

Surprisingly, topological predicates have so far not been defined on complex spatial objects. So far, two works [2,5] have given a definition of topological relationships between two more complex regions. But either their region definition only allows sets of disjoint simple regions without holes [2] or only single simple regions with holes [5]. The results are also problematic in the sense that they either depend on the number of components or on the number of holes.

3 Complex Lines and Complex Regions

This section defines the underlying spatial data model for our topological predicates. We strive for a very general, abstract definition of complex lines and com-

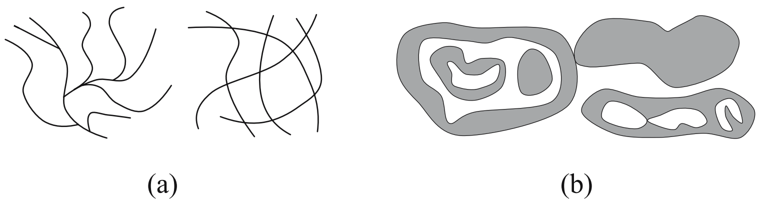


Fig. 1. Examples of a complex line object (a) and a complex region object (b)

plex regions (see Figure 1) in the Euclidean plane $\mathbb{R}^2$. Our formal framework are basic concepts of point set theory and point set topology [6]. The task is to determine those point sets that are admissible for complex line (Section 3.1) and complex region (Section 3.2) objects. The definitions we give contribute to an "unstructured" object definition which solely determines the point set of a line or region. Due to space restrictions, we do not identify structural components. But a complex line represents a spatially embedded network possibly consisting of several connected components, and a complex region represents a multi-part region possibly with holes. For both spatial data types we specify the topological notions of *boundary*, *interior*, *exterior*, and *closure* since these notions are later needed for the specification of topological relationships.

3.1 Complex Lines

Before we start with a definition for complex lines (Figure 1a), we need a few definitions of some well-known and needed topological concepts. We assume the existence of the Euclidean distance function $d : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ with $d(p, q) = d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. With the notion of distance, we can now proceed to define what is meant by a *neighborhood* of a point in $\mathbb{R}^2$.

Definition 1. *Let $q \in \mathbb{R}^2$ and $\epsilon \in \mathbb{R}^+$. The set $N_\epsilon(q) = \{p \in \mathbb{R}^2 \mid d(p, q) < \epsilon\}$ is called the* open neighborhood of radius ϵ and center q. *Any open neighborhood with center q is denoted by $N(q)$.* □

We are now able to define the notion of a *continuous mapping* which preserves neighborhood relations between mapped points in two spaces of the plane. Hence, the property of continuity of this mapping ensures the maintenance of the closure and connectivity of the mapping domain for its image. These mappings are also called *topological transformations* and include translation, rotation, and scaling.

Definition 2. *Let $X \subset \mathbb{R}$ and $f : X \to \mathbb{R}^2$. Then f is said to be* continuous at a point $x_0 \in X$ *if, given an arbitrary number $\epsilon > 0$, there exists a number $\delta > 0$ (usually depending on ϵ) such that for every $x \in N_\delta(x_0) \cap X$ we obtain that $f(x) \in N_\epsilon(f(x_0))$. The mapping f is said to be* continuous on X *if it is continuous at every point of X.* □

For a function $f : X \to Y$ and a set $A \subseteq X$ we introduce the notation $f(A) = \{f(x) \mid x \in A\}$. Definition 2 enables us to give an unstructured definition for complex lines as the union of the images of a finite number of continuous mappings.

Definition 3. *The spatial data type line is defined as*

$$\begin{aligned} line = \{L \subset \mathbb{R}^2 \mid & \textit{(i)} \quad L = \textstyle\bigcup_{i=1}^{n} f_i([0,1]) \textit{ with } n \in \mathbb{N}_0 \\ & \textit{(ii)} \ \forall\, 1 \leq i \leq n : f_i : [0,1] \to \mathbb{R}^2 \textit{ is a continuous mapping} \\ & \textit{(iii)} \ \forall\, 1 \leq i \leq n : |f_i([0,1])| > 1\} \end{aligned}$$

We call a value of this type complex line *and the image of a continuous mapping* continuous line*.* □

The first condition also allows a line object to be the empty point set ($n = 0$ in Definition 3). The third condition avoids degenerate line objects consisting only of a single point.

The boundary of a complex line L is the set of its end points minus those end points that are shared by several continuous lines. The shared points belong to the interior of a complex line. Based on Definition 3, let $E(L) = \bigcup_{i=1}^{n}\{f_i(0), f_i(1)\}$ be the set of end points of all continuous lines. We obtain

$$\begin{aligned} \partial L = E(L) - \{p \in E(L) \mid & \, card(\{f_i \mid 1 \leq i \leq m \ \wedge \ f_i(0) = p\}) + \\ & card(\{f_i \mid 1 \leq i \leq m \ \wedge \ f_i(1) = p\}) \neq 1\} \end{aligned}$$

Let $L \neq \varnothing$. It is possible that ∂L is empty (e.g., if L is a closed continuous line). The closure $\overline{L}$ of L is the set of all points of L including the end points. Therefore $\overline{L} = L$ holds. For the interior of L we obtain $L^{\circ} = \overline{L} - \partial L = L - \partial L \neq \varnothing$, and for the exterior we get $L^{-} = \mathbb{R}^2 - L$, since $\mathbb{R}^2$ is the embedding space.

3.2 Complex Regions

Regions are embedded into the two-dimensional Euclidean space $\mathbb{R}^2$ and modeled as special infinite point sets. We briefly introduce some needed concepts from point set topology in $\mathbb{R}^2$.

Definition 4. *Let $X \subseteq \mathbb{R}^2$ and $q \in \mathbb{R}^2$. q is an* interior point *of X if there exists a neighborhood N such that $N(q) \subseteq X$. q is an* exterior point *of X if there exists a neighborhood N such that $N(q) \cap X = \varnothing$. q is a* boundary point *of X if q is neither an interior nor exterior point of X. q is a* closure point *of X if q is either an interior or boundary point of X.*

The set of all interior points of X is called the interior *of X and is denoted by X°. The set of all exterior points of X is called the* exterior *of X and is denoted by X^{-}. The set of all boundary points of X is called the* boundary *of X and is denoted by ∂X. The set of all closure points of X is called the* closure *of X and is denoted by $\overline{X}$.*

A point q is a limit point *of X if for every neighborhood $N(q)$ holds that $(N - \{q\}) \cap X \neq \varnothing$. X is called an* open set *in $\mathbb{R}^2$ if $X = X^{\circ}$. X is called a* closed set *in $\mathbb{R}^2$ if every limit point of X is a point of X.* □

It follows from the definition that every interior point of X is a limit point of X. Thus, limit points need not be boundary points. The converse is also true. A boundary point of X need not be a limit point; it is then called an *isolated* point of X. For the closure of X we obtain that $\overline{X} = \partial X \cup X^{\circ}$.

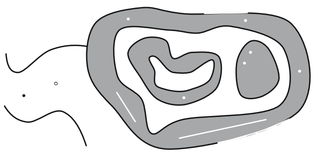


Fig. 2. Examples of possible geometric anomalies of a region object

It is obvious that arbitrary point sets do not necessarily form a region. But open and closed point sets in $\mathbb{R}^2$ are also inadequate models for complex regions since they can suffer from undesired geometric anomalies (Figure 2). A complex region defined as an open point set runs into the problem that it may have missing lines and points in the form of cuts and punctures. At any rate, its boundary is missing. A complex region defined as a closed point set admits isolated or dangling point and line features. *Regular closed* point sets [12] avoid these anomalies.

Definition 5. *Let $X \subseteq \mathbb{R}^2$. X is called a* regular closed set *if, and only if, $X = \overline{X^\circ}$.* □

The effect of the *interior* operation is to eliminate dangling points, dangling lines, and boundary parts. The effect of the *closure* operation is to eliminate cuts and punctures by appropriately supplementing points and to add the boundary.

For the specification of the *region* data type, definitions are needed for bounded and connected sets.

Definition 6. *(i) Two sets $X, Y \subseteq \mathbb{R}^2$ are said to be* separated *if, and only if, $X \cap \overline{Y} = \varnothing = \overline{X} \cap Y$. A set $X \subseteq \mathbb{R}^2$ is* connected *if, and only if, it is not the union of two non-empty separated sets. (ii) Let $q = (x, y) \in \mathbb{R}^2$. Then the* length *or* norm *of q is defined as $||q|| = \sqrt{x^2 + y^2}$. (iii) A set $X \subseteq \mathbb{R}^2$ is said to be* bounded *if there exists a number $r \in \mathbb{R}^+$ such that $||q|| < r$ for every $q \in X$.* □

We are now able to give an unstructured type definition for complex regions:

Definition 7. *The spatial data type region is defined as*

$$\begin{array}{lll} \textit{region} = \{R \subset \mathbb{R}^2 \mid & (i) & R \text{ is regular closed} \\ & (ii) & R \text{ is bounded} \\ & (iii) & \text{The number of connected sets of } R \text{ is finite}\} \end{array}$$

We call a value of this type complex region. □

A region object can also be the empty object (empty set). Let $F = \bigcup_{i=1}^{n} F_i$ be a non-empty region with faces $\{F_1, \ldots, F_n\}$. Then the boundary of F is given as $\partial F = \bigcup_{i=1}^{n} \partial F_i$ ($\neq \varnothing$), and the interior of F is given as $F^\circ = \bigcup_{i=1}^{n} F_i^\circ = F - \partial F$ ($\neq \varnothing$). Further, we obtain $\overline{F} = \partial F \cup F^\circ = F$ and $F^- = \mathbb{R}^2 - \overline{F} = \mathbb{R}^2 - F$ ($\neq \varnothing$).

4 The Proof-by-Constraint-and-Drawing Method

An apparently promising approach to deriving topological relationships is to leverage the component view of a spatial object. But research on region objects in this direction reveals that considering components leads to rather complicated and impractical models. We demonstrate this by first considering two simple regions A and B with n and m holes, respectively. If we take into account the regions A and B *without* holes and call them A^* and B^*, respectively, the total number of topological relationships that can be specified between A^* and its holes with B^* and its holes amounts to $(n+m+2)^2$ [5]. It has also been shown in [5] that this number can be reduced to $mn+m+n+1$. The problems of this approach are the dependency on the number of holes and the resulting large number of topological relationships.

We are confronted with a similar problem if we take another strategy and have a look on the topological relationships between two complex regions A and B with n and m faces, respectively, possibly with holes. Each face of A is in relationship with any face of B. This gives us a total of $8^{n \cdot m}$ possible topological configurations if we take the eight topological relationships between two simple regions with holes, as they are specified in [11], as the basis. As a result, the total number of relationships between the faces of two complex regions depends on the numbers of faces, is therefore not bounded by a constant, and increases in an exponential way. This approach is obviously not manageable and thus not acceptable.

Hence, the comparison of structural components of the objects with respect to their topological relationships does not seem to be an adequate and efficient method. Often, such a detailed investigation is not desired and thus even unnecessary. For instance, if two regions intersect (according to some definition), the number of intersecting face pairs, as long as it is greater than 0, is irrelevant since it does not influence the fact of intersection. Consequently, the analysis of topological relationships between two complex spatial objects requires a more general strategy.

Our strategy is simple and yet very general and expressive. Instead of applying the 9-intersection model to point sets belonging to simple spatial objects, we extend it to point sets belonging to complex spatial objects. Due to the special features of the objects (point, linear, areal properties), the embedding space (here: $\mathbb{R}^2$), the relation between the objects and the embedding space (e.g., it makes a difference whether we consider a point in $\mathbb{R}$ or in $\mathbb{R}^2$), and the employed spatial data model (e.g., discrete, continuous), a number of topological configurations cannot exist and have to be excluded. For each pair of complex spatial data types, our goal is to determine topological constraints that have to be satisfied. These serve as criteria for excluding all impossible configurations. The approach taken employs a proof technique which we call *Proof-By-Constraint-And-Drawing*. It starts with the 512 possible matrices and is a two-step process:

(i) For each type combination we give the formalization of a collection of topological *constraint* rules for existing relationships in terms of the nine intersections. For each constraint rule we give reasons for its validity, correctness, and meaningfulness. The evaluation of each constraint rule gradually reduces the set of the currently valid matrices by all those matrices not fulfilling the constraint rule under consideration.

(ii) The existence of topological relationships given by the remaining matrices is verified by realizing prototypical spatial configurations in $\mathbb{R}^2$, i.e., these configurations can be *drawn* in the plane.

Still open issues relate to the evaluation order, completeness, and minimality of the collection of constraint rules. Each constraint rule is a predicate that is matched with all intersection matrices under consideration. All constraint rules must be satisfied together so that they represent a conjunction of predicates. To say it in other words, constraint rules are all formulated in conjunctive normal form. Since the conjunction (logical *and*) operator is commutative and associative, the *evaluation order* of the constraint rules is irrelevant; the final result is always the same.

The *completeness* of the collection of constraints is directly ensured by the second step of the two-step process provided that spatial configurations for all remaining matrices can be drawn.

The aspect of *minimality* addresses the possible redundancy of constraint rules. Redundancy can arise for two reasons. First, several constraint rules may be correlated in the sense that one of them is more general than the others, i.e., it eliminates at least the matrices excluded by all the other, covered constraints. This can be easily checked by analyzing the constraint rules themselves and searching for the most non-restrictive and common constraint rule. Even then the same matrix can be excluded by several constraint rules simultaneously. Second, a constraint rule can be covered by some combination of other constraint rules. This can be checked by a comparison of the matrix collection fulfilling all n constraint rules with the matrix collection fulfilling $n - 1$ constraint rules. If both collections are equal, then the omitted constraint rule was implied by the combination of the other constraint rules and is therefore redundant. In this paper, we are not so much interested in the aspect of minimality since our goal is to identify the topologically invalid intersection matrices (predicates) so that the valid matrices remain.

5 Topological Relationships for the Complex Line/Complex Region Case

Leveraging the proof technique developed in the last section, we develop constraint rules for the identification of all topological relationships between a complex line and a complex region. In the following, we assume that A is a non-empty object of type *line* and B is a non-empty object of type *region*.

Lemma 1. *The exteriors of a complex line and a complex region always intersect with each other, i.e.,*

$$A^- \cap B^- \neq \varnothing$$

Proof. We know that $\overline{A} \cup A^- = \mathbb{R}^2$ and $\overline{B} \cup B^- = \mathbb{R}^2$. Hence, $A^- \cap B^-$ is only empty if either (i) $\overline{A} = \mathbb{R}^2$, or (ii) $\overline{B} = \mathbb{R}^2$, or (iii) $\overline{A} \cup \overline{B} = \mathbb{R}^2$. The situations are all impossible, since A, B, and hence $A \cup B$ are bounded, but the Euclidean plane $\mathbb{R}^2$ is unbounded. $\square$

Lemma 2. *The interior of a complex region always intersects the exterior of a complex line, i.e.,*

$$A^- \cap B^\circ \neq \varnothing$$

Proof. Assuming that this constraint rule is wrong. Then $A^- \cap B^\circ = \varnothing$, and we can conclude that $A \supseteq B^\circ$. From this we obtain that $\forall p \in B^\circ\ \exists \epsilon \in \mathbb{R}^+ : N_\epsilon(p) \subseteq B^\circ \Rightarrow N_\epsilon(p) \subseteq A$. This leads to a contradiction since $\forall p \in B^\circ\ \forall \epsilon \in \mathbb{R}^+ : N_\epsilon(p) \nsubseteq A$. $\square$

Intuitively, a line object as a one-dimensional, linear entity cannot cover a region object, which is a two-dimensional, areal entity.

Lemma 3. *The interior or the exterior of a complex line intersects the boundary of a complex region, i.e.,*

$$A^\circ \cap \partial B \neq \varnothing \ \vee \ A^- \cap \partial B \neq \varnothing$$

Proof. We know that $\partial B \neq \varnothing$ and that hence $\mathbb{R}^2 \cap \partial B \neq \varnothing$. Since $A^\circ \cup \partial A \cup A^- = \mathbb{R}^2$, we obtain that $(A^\circ \cup \partial A \cup A^-) \cap \partial B \neq \varnothing$. This leads to $A^\circ \cap \partial B \neq \varnothing \ \vee \ \partial A \cap \partial B \neq \varnothing \ \vee \ A^- \cap \partial B \neq \varnothing$. Since ∂A is a finite point set and ∂B is an infinite point set, either $\partial A \subset \partial B$ or $\partial A \cap \partial B = \varnothing$. This means that the constraint rule $A^\circ \cap \partial B \neq \varnothing \ \vee \ A^- \cap \partial B \neq \varnothing$ must hold. $\square$

Lemma 4. *The interior of a complex line intersects at least one part of a complex region, i.e.,*

$$A^\circ \cap \partial B \neq \varnothing \ \vee \ A^\circ \cap B^\circ \neq \varnothing \ \vee \ A^\circ \cap B^- \neq \varnothing$$

Proof. We know that $A^\circ \cup A^- = \mathbb{R}^2$ and that $\partial B \cup B^\circ \cup B^- = \mathbb{R}^2$. Since only non-empty object parts of both objects are taken into account, we obtain $A^\circ \cap \mathbb{R}^2 = A^\circ \cap (\partial B \cup B^\circ \cup B^-) \neq \varnothing$. This statement is equivalent to the constraint rule. $\square$

Lemma 5. *If the boundary of a complex line intersects the interior of a complex region, also its interior intersects the interior of the complex region, i.e.,*

$$\begin{aligned} &(\partial A \cap B^\circ \neq \varnothing \Rightarrow A^\circ \cap B^\circ \neq \varnothing) \\ \Leftrightarrow\ &(\partial A \cap B^\circ = \varnothing \ \vee \ A^\circ \cap B^\circ \neq \varnothing) \end{aligned}$$

Proof. Without loss of generality, let $p \in \partial A \cap B^\circ$. Since $p \in B^\circ$, an $\epsilon \in \mathbb{R}^+$ exists such that $N_\epsilon(p) \subset B^\circ$. Fixing such an ϵ, and because a continuous curve with an infinite number of points starts in p, we obtain that $N_\epsilon(p) \cap A^\circ \neq \varnothing$. This leads to the conclusion that $A^\circ \cap B^\circ \neq \varnothing$. $\square$

Lemma 6. *If the boundary of a complex line intersects the exterior of a complex region, also its interior intersects the exterior of the complex region, i.e.,*

$$\begin{aligned}&(\partial A \cap B^- \neq \varnothing \Rightarrow A^\circ \cap B^- \neq \varnothing)\\ \Leftrightarrow\ &(\partial A \cap B^- = \varnothing\ \vee\ A^\circ \cap B^- \neq \varnothing)\end{aligned}$$

Proof. The argumentation is analogous to the argumentation for the constraint rule in Lemma 5. □

Lemma 7. *If the boundary of a complex line intersects the boundary of a complex region, also its exterior intersects the boundary of the complex region, i.e.,*

$$\begin{aligned}&(\partial A \cap \partial B \neq \varnothing \Rightarrow A^- \cap \partial B \neq \varnothing)\\ \Leftrightarrow\ &(\partial A \cap \partial B = \varnothing\ \vee\ A^- \cap \partial B \neq \varnothing)\end{aligned}$$

Proof. The boundary of a region B is a line object L whose components are all closed curves. Hence, this line object only consists of interior points ($L = L^\circ$). Without loss of generality, let P be an endpoint of the boundary of A located on L. From P exactly one curve of A starts or ends. Either P divides a curve of L into two subcurves, or P is endpoint of more than one curve of L. Hence, in P at least two curves of L end. Since the curve of A can coincide with at most one of the curves of L, at least one of the curves of L must be situated in the exterior of A. □

An evaluation of all 512 3×3-intersection matrices against these seven constraint rules with the aid of a simple test program reveals that 43 matrices satisfy these rules and thus represent the possible topological relationships between a complex line and a complex region. The matrices and their geometric interpretations are shown in Table 2. Between a *simple* line and a *simple* region we can distinguish 19 topological relationships [3]. These topological predicates are contained in the 43 general ones and correspond to the matrices 2-4, 7, 11-13, 15-17, 28, 30, 31, 35-37, 39, 41, and 42, respectively.

Finally, we can summarize our result as follows:

Theorem 1. *Based on the 9-intersection model, 43 different topological relationships can be identified between a complex line object and a complex region object.*

Proof. The argumentation is based on the *Proof-By-Constraint-And-Drawing* method described in Section 4. The constraint rules, whose correctness has been shown in Lemmas 1 to 7, reduce the number of the 512 possible intersection matrices to 43 matrices. The ability to draw prototypes of the corresponding 43 topological configurations proves that the constraint rules are complete. □

Table 2 in the Appendix shows for each topological predicate a prototypical configuration as a drawing.

6 Conclusions and Future Work

In this paper we have given a very general definition of spatial data types for complex lines and complex regions in the two-dimensional Euclidean space on the basis of point set theory and point set topology. Further, we have developed a proof technique called *Proof-By-Constraint-And-Drawing* which enables the derivation of a complete collection of mutually exclusive topological relationships between all combinations of complex spatial data types. We have demonstrated this mechanism by deriving all 43 topological relationships between a complex line and a complex region.

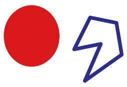
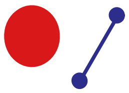
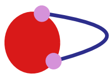
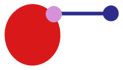
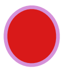
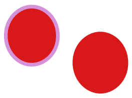
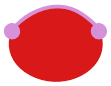
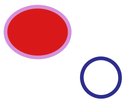
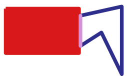
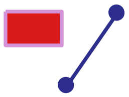
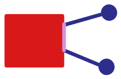
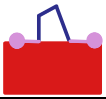
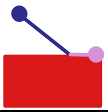
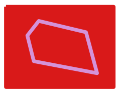
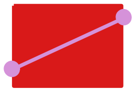
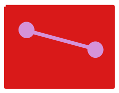
Future work will relate to the derivation of topological predicates for all other combinations of complex spatial data types. Further, the efficient implementation of the large numbers of predicates that have to be expected will be another topic.

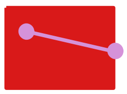
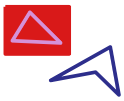
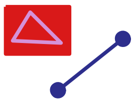
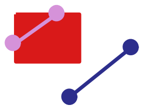
References

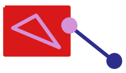
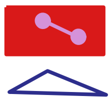
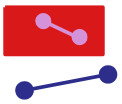
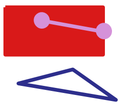
1. E. Clementini and P. Di Felice. A Model for Representing Topological Relationships between Complex Geometric Features in Spatial Databases. *Information Systems*, 90(1-4):121–136, 1996.
2. E. Clementini, P. Di Felice, and G. Califano. Composite Regions in Topological Queries. *Information Systems*, 20(7):579–594, 1995.
3. M. J. Egenhofer and J. Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical Report 90-12, National Center for Geographic Information and Analysis, University of California, Santa Barbara, 1990.
4. M. J. Egenhofer and D. Mark. Modeling Conceptual Neighborhoods of Topological Line-Region Relations. *Int. Journal of Geographical Information Systems*, 9(5):555–565, 1995.
5. M.J. Egenhofer, E. Clementini, and P. Di Felice. Topological Relations between Regions with Holes. *Int. Journal of Geographical Information Systems*, 8(2):128–142, 1994.
6. S. Gaal. *Point Set Topology.* Academic Press, 1964.
7. R. H. Güting. An Introduction to Spatial Database Systems. *VLDB Journal*, 3(4):357–399, 1994.
8. R. H. Güting and M. Schneider. Realm-Based Spatial Data Types: The ROSE Algebra. *VLDB Journal*, 4:100–143, 1995.
9. OGC Abstract Specification. OpenGIS Consortium (OGC), 1999. URL: http://www.opengis.org/techno/specs.htm.
10. M. Schneider. *Spatial Data Types for Database Systems - Finite Resolution Geometry for Geographic Information Systems*, volume LNCS 1288. Springer-Verlag, Berlin Heidelberg, 1997.
11. M. Schneider. A Design of Topological Predicates for Complex Crisp and Fuzzy Regions. *Int. Conf. on Conceptual Modeling*, pp. 103–116, 2001.
12. R. B. Tilove. Set Membership Classification: A Unified Approach to Geometric Intersection Problems. *IEEE Trans. on Computers*, C-29:874–883, 1980.

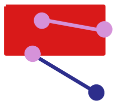
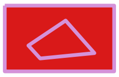
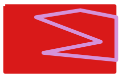
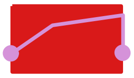
Appendix

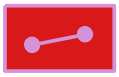
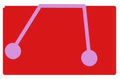
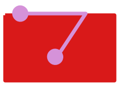
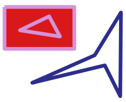
Table 2. The 43 topological relationships between a complex line and a complex region

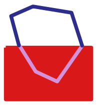
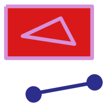
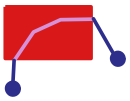
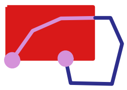
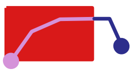
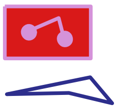
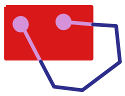
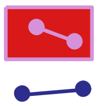
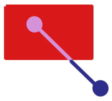
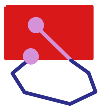
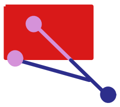
Matrix 1	Matrix 2	Matrix 3	Matrix 4
$\begin{pmatrix}0&0&1\\0&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&0&1\\0&0&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&0&1\\0&1&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&0&1\\0&1&1\\1&1&1\end{pmatrix}$
			
Matrix 5	Matrix 6	Matrix 7	Matrix 8
$\begin{pmatrix}0&1&0\\0&0&0\\1&0&1\end{pmatrix}$	$\begin{pmatrix}0&1&0\\0&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&1&0\\0&1&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&1&1\\0&0&0\\1&0&1\end{pmatrix}$
			
Matrix 9	Matrix 10	Matrix 11	Matrix 12
$\begin{pmatrix}0&1&1\\0&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&1&1\\0&0&1\\1&0&1\end{pmatrix}$	$\begin{pmatrix}0&1&1\\0&0&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}0&1&1\\0&1&0\\1&1&1\end{pmatrix}$
			
Matrix 13	Matrix 14	Matrix 15	Matrix 16
$\begin{pmatrix}0&1&1\\0&1&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&0\\0&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&0\\0&1&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&0\\1&0&0\\1&1&1\end{pmatrix}$
			

Matrix 17	Matrix 18	Matrix 19	Matrix 20
$\begin{pmatrix}1&0&0\\1&1&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\0&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\0&0&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\0&1&0\\1&1&1\end{pmatrix}$
			

Matrix 21	Matrix 22	Matrix 23	Matrix 24
$\begin{pmatrix}1&0&1\\0&1&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\1&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\1&0&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&0&1\\1&1&0\\1&1&1\end{pmatrix}$
			

Matrix 25	Matrix 26	Matrix 27	Matrix 28
$\begin{pmatrix}1&0&1\\1&1&1\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&1&0\\0&0&0\\1&0&1\end{pmatrix}$	$\begin{pmatrix}1&1&0\\0&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&1&0\\0&1&0\\1&1&1\end{pmatrix}$
			

Matrix 29	Matrix 30	Matrix 31	Matrix 32
$\begin{pmatrix}1&1&0\\1&0&0\\1&0&1\end{pmatrix}$	$\begin{pmatrix}1&1&0\\1&0&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&1&0\\1&1&0\\1&1&1\end{pmatrix}$	$\begin{pmatrix}1&1&1\\0&0&0\\1&0&1\end{pmatrix}$
			

Matrix 33	Matrix 34	Matrix 35	Matrix 36
$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$
			
Matrix 37	Matrix 38	Matrix 39	Matrix 40
$\begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$
			
Matrix 41	Matrix 42	Matrix 43	
$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$	
			

Author Index