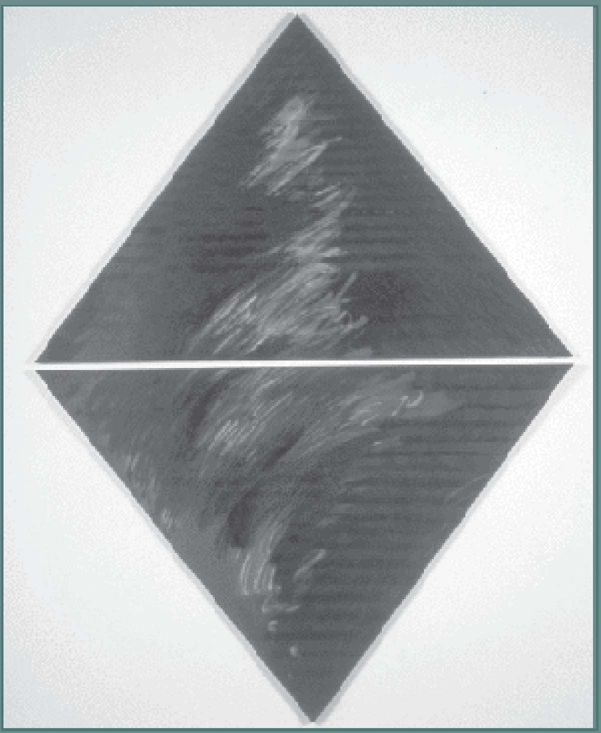
Advanced Topics in Database Research

Volume 3



Keng Siau

Advanced Topics in Database Research

Volume 3

Keng Siau
University of Nebraska-Lincoln, USA

IDEA GROUP PUBLISHING
Hershey • London • Melbourne • Singapore

Acquisitions Editor: Mehdi Khosrow-Pour
Senior Managing Editor: Jan Travers
Managing Editor: Amanda Appicello
Development Editor: Michele Rossi
Copy Editor: Ingrid Widitz
Typesetter: Sara Reed
Cover Design: Idea Group Inc.
Printed at: Yurchak Printing Inc.

Published in the United States of America by
Idea Group Publishing (an imprint of Idea Group Inc.)
701 E. Chocolate Avenue, Suite 200
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@idea-group.com
Web site: http://www.idea-group.com

and in the United Kingdom by
Idea Group Publishing (an imprint of Idea Group Inc.)
3 Henrietta Street
Covent Garden
London WC2E 8LU
Tel: 44 20 7240 0856
Fax: 44 20 7379 3313
Web site: http://www.eurospan.co.uk

Advanced Topics in Database Research, Volume 3 is part of the Idea Group Publishing series named *Advanced Topics in Database Research* (Series ISSN 1537-9299)

ISBN 1-59140-255-7
ISBN Paperback 1-59140-296-4
eISBN 1-59140-256-5

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Advanced Topics in Database Research

Volume 3

Table of Contents

SECTION I: ANALYSIS OF DEVELOPMENT METHODOLOGIES

SECTION II: DATABASE DESIGN AND DEVELOPMENT: ISSUES AND SOLUTIONS

SECTION III: DATABASE DESIGN AND DEVELOPMENT: APPLICATIONS

Preface

The *Advanced Topics in Database Research* book series has been recognized as an outstanding academic book series in the fields of database, software engineering, as well as systems analysis and design. The goal of the book series is to provide researchers and practitioners easy access to excellent chapters which address the latest research issues in the field of database (the term "database" is used here broadly).

This is the third volume of the *Advanced Topics in Database Research* book series. This book consists of 16 excellent chapters ranging from theoretical database issues to practical applications of database techniques. In terms of research methodology, the chapters vary from meta-modeling to empirical case studies. Although the topics are broad, the book provides a sample of some of the best research work done in the database area. The contributing authors represent almost every part of the globe. We have authors from the USA, Canada, The Netherlands, Spain, Chile, Hungary, Israel, Lebanon, Korea, and China.

The book is divided into three sections: (I) Analysis of Development Methodologies; (II) Database Design and Development: Issues and Solutions; and (III) Database Design and Development: Applications. In the following, we briefly describe each chapter:

Section I: Analysis of Development Methodologies consists of three chapters.

Chapter I, "Agile Development Methods and Component-Orientation: A Review and Analysis," presents and analyzes the state-of-the-art agile methods used in the agile development process. Different conceptual foundations and practical uses of these methods, as well as their limitations, are listed and discussed. Service-based component concepts applied at the level of modeling, architectural design, and development are proposed to ensure and strengthen agile development principles and practices. The paper also introduces necessary agility to more traditional development.

Chapter II, "Comparing Metamodels for ER, ORM and UML Data Models," gives a concrete metamodel analysis of the three main database modeling techniques used in the industry — Entity Relationship (ER), Object Role Modeling (ORM), and Unified Modeling Language (UML). ORM is used as the metamodeling language because of its great expressibility and clarity. Discussions based on the metamodel analysis are detailed in the chapter.

Chapter III, “An Evaluation Framework for Component-Based and Service-Oriented System Development Methodologies,” presents an evaluation framework that highlights the extent to which a particular method is component-based and service-oriented. The framework is then applied to evaluate a few popular Component-Based Development (CBD) methods. Based on the evaluation, improvements to these methods are proposed to provide a consistent, systematic, and integrated CBD and Web-Service (WS) methodology support throughout the system life cycle.

Section II: Database Design and Development: Issues and Solutions consists of seven chapters.

Chapter IV, “Improving the Understandability of Dynamic Semantics: An Enhanced Metamodel for UML State Machines,” introduces an approach to improve the understandability of the dynamic semantics of languages involved in the representation of behavior. Using a two-layer architecture as the starting point, a metamodel of UML State Machines is proposed.

Chapter V, “Metrics for Workflow Design: How an Information Processing View on Business Processes Helps to Make Good Designs,” introduces a cohesion metric for the identification of weakly cohesive activities in a workflow design. A heuristic method based on the cohesion metric is presented to decide between various workflow design alternatives. Both theoretical and empirical evaluations positively support the soundness of the metric.

Chapter VI, “Fuzzy Aggregations and Fuzzy Specializations in Eindhoven Fuzzy EER Model,” uses fuzzy quantifiers and fuzzy degrees in the context of fuzzy sets and fuzzy query systems for understanding semantic aspects in database concepts. The study is aimed to relax some constraints and other aspects that have not been studied in previous works. The study also extends the Enhanced Entity-Relationship (EER) model with fuzzy capabilities.

Chapter VII, “Normalization of Relations with Nulls in Candidate Keys: Traditional and Domain Key Normal Forms,” discusses normalization of relations when the candidate keys of a relation have missing information represented by nulls. The related limitations of Boyce-Codd Normal Form (BCNF) and Domain Key Normal Form (DKNF) can be solved by incorporating the concept of entity integrity rule into the respective definitions.

Chapter VIII, “Regression Test Selection for Database Applications,” discusses the difficulties caused by some database applications’ features during maintenance activities, especially for regression testing that follows modification to database applications. The chapter proposes a two-phase regression testing methodology for selecting regression tests and for further reduction in the number of these tests.

Chapter IX, “An Attempt to Establish a Correspondence between Development Methods and Problem Domains,” discusses the issue of development method adaptation. Then it introduces a new approach to calculate the fitness of methods to specific problems.

Chapter X, “Toward an Extended Framework for Human Factors Research on Data Modeling,” summarizes the past human factors research on conceptual data modeling. In addition to analyzing the variables used in earlier studies and summarizing the results of this stream of research, the authors propose a new framework to help both scholars and practitioners in this area.

Section III: Database Design and Development: Applications consists of six chapters.

Chapter XI, “Using DEMO and ORM in Concert: A Case Study,” examines the role of Demo Engineering Methodology for Organizations (DEMO) and Object-Role Modeling (ORM) in conceptually modeling business processes. An exploratory case study of applying the two methods in concert is provided.

Chapter XII, “Revisiting Workflow Modeling with Statecharts,” proposes the use of Harel’s statecharts in business workflows modeling. The authors developed algorithms that link desirable properties of active database system—non-termination, non-confluence, and not-observable determinism—to problems in workflow management systems.

Chapter XIII, “Framework for the Rapid Development of Modeling Environments,” presents Generic Modeling Environment (GME) as a framework for rapid development of modeling environments. The chapter also compares GME with other tools in terms of metamodeling, constraint management, visualization, and extensibility.

Chapter XIV, “Federated Process Framework for Transparent Process Monitoring in Business Process Outsourcing,” proposes a federated process framework and its system architecture. The architecture provides a conceptual design for effective implementation of process information sharing that supports the autonomy and agility of insourcing companies. The framework was developed using an object-oriented database and Extensible Markup Language.

Chapter XV, “Online Analytical Mining for Web Access Patterns,” offers an architecture to store the derived web user access paths in a data warehouse and to facilitate its view maintainability by use of a metadata. The architecture of online analytical mining uses the frame model metadata to study the user surfing behavior. Performance studies were done to demonstrate the effectiveness and efficiency of the proposed architecture.

Chapter XVI, “Modeling Motion: Building Blocks of a Motion Database,” introduces a binary-based model for the representation and storage of motion data. The model enables the communication, storage, and analysis of patterns of motion. The comparison with a standard motion system that is based on key frames indicates a significant advantage of the proposed model.

These 16 chapters provide a sample of the state-of-the-art research in the field of database. We hope that both scholars and practitioners will find the book a useful reference for their work.

Keng Siau
University of Nebraska-Lincoln, USA
November 2003

SECTION I:

ANALYSIS OF DEVELOPMENT METHODOLOGIES

Chapter I

Agile Development Methods and Component-Orientation: A Review and Analysis

Zoran Stojanovic, Delft University of Technology, The Netherlands

Ajantha Dahanayake, Delft University of Technology, The Netherlands

Henk Sol, Delft University of Technology, The Netherlands

ABSTRACT

Agile software development methods have been proposed as the way to address the problem of delivering high-quality software on time under constantly and rapidly changing requirements in business and IT environments. An agile development process is characterized by extensive coding practice, intensive communication between stakeholders, fast iterative cycles, small and flexible teams, and minimal efforts in system modeling and architectural design. This paper presents the state-of-the-art of agile methods and analyzes them along the selected criteria that highlight different aspects of their theory and practice. Certain limitations of agile methods are identified. The chapter presents the component paradigm as a way of balancing traditional (model-driven or plan-driven) and agile development, depending on the project settings. Service-based component concepts applied at the level of modeling, architectural design and development can ensure and strengthen agile development principles and practices, and at the same time introduce necessary agility to more traditional development. By using components, the software development process can easily scale in size, robustness, and the level of details. This provides an effective balance between the requirements for agility in software development and needs for a disciplined, design-driven way of building complex software.

INTRODUCTION

EXtreme Programming (XP) and other Agile Methodologies (AMs) have started to gain considerable interest in the IT community during the last several years. They have been proposed as a way to build quality software systems fast and be able to easily adapt to rapidly and frequently changing requirements in the environment. Agile processes are focused on early, fast and frequent production of working code through the fast iterations and small increments. The processes are characterized by intensive communication between participants, rapid feedback, simple design and frequent testing. By their proponents, the software code is the main deliverable, while the role of system analysis, design and documentation in software development and maintenance is de-emphasized and to some extent ignored.

A number of processes claiming to be “agile” have been proposed so far. The best examples are eXtreme Programming (XP) (Beck, 2000), Scrum (Schwaber & Beedle, 2002), Feature-Driven Development (FDD) (Palmer & Felsing, 2002), Adaptive Software Development (ASD) (Highsmith, 2000), Crystal methods family (Cockburn, 2002) and DSDM (Stapleton, 2003). There have been attempts in applying agile values, principles and practices in earlier phases of the software life cycle, such as analysis and design, under the initiatives called Agile Modeling (Ambler, 2002) and eXtreme Modeling (Extreme, 2003). Efforts have been made to investigate how the Unified Modeling Language (UML) can be used in an agile process, as well as how to use the Rational Unified Process (RUP) (Jacobson, Booch & Rumbaugh, 1999) in an agile manner (Larman, 2001; Ambler, 2002). The authors of the listed agile approaches have formed the Agile Alliance and published the Agile Manifesto that represents a condensed definition of principles and goals of agile software development (Agile Alliance, 2001). These principles are:

- Individuals and interactions over processes and tools,
- Working software over comprehensive documentation,
- Customer collaboration over contract negotiation, and
- Responding to change over following a plan.

Agile Development (AD) paradigm challenges many of the common assumptions in software development. One of the most controversial is its rejection of significant efforts in up-front design in favor of a more evolutionary approach. According to its critics this is very similar to the so-called code-and-fix hacking strategy in software development. XP and other AMs minimize the role of common design techniques in traditional software development such as frameworks, design patterns, modeling tool support, modeling languages, model repositories and reusability. On the other hand, AD supporters claim that their methodologies include just enough design efforts for the project to be successful, and AD design is actually done in a different way than in traditional software processes. For example, in XP simple metaphor-like design, refactoring, architecture prototypes, and test-based design are used in an evolutionary way for software design purposes. These characteristics of XP and other AMs are opposite to the current initiatives and paradigms in software development, such as Model-Driven Development (MDD) (OMG, 2003). While both AD and MDD claim to address the challenges of high change rates, short time-to-market, increased return-on-investment and high quality software, their proposed solutions are actually very dissimilar. The question is whether principles and practices of both development paradigms can be combined in order to take the benefits of both approaches.

The aim of this chapter is to present the state-of-the-art of agile methodologies and analyze them according to the set of selected criteria. Special attention is paid on how modeling and architectural design are addressed in the current agile methodology practice as well as what kind of support to modeling and design activities exists in the selected set of methodologies. The paper further proposes how concepts of component-based modeling, design and development can help in bridging the gap between model-driven and agile development. The paper shows how components can ensure and strengthen AD principles and practices, provide simple and flexible component-oriented architectural design, as well as help in overcoming the limitations of the agile methodologies, such as reusability, outsourcing, large teams and software, and safety critical software development.

THE STATE-OF-THE-ART OF AGILE METHODS

In this section, different agile methodologies are presented and analyzed according to the set of criteria. Although all agile methodologies share similar concepts, principles and practice, their focus, scope and nature are varied. Some agile methodologies such as Scrum, Adaptive Software Development, Crystal family and Dynamic Systems Development Method are primarily focused on the project management and teamwork aspects. These methods do not particularly treat any specific software development practice including any modeling and design activities. These methods will be presented rather briefly, while the methods covering software modeling, design and development practice (XP, FDD, Agile Modeling and Extreme Modeling) will be covered in more detail.

Extreme Programming

Extreme Programming (XP) is a lightweight development methodology defined by Kent Beck (Beck, 1999; Jeffries, Anderson & Hendrickson, 2001) that has received much attention during the last years. XP is the most documented, popular and widely used agile methodology. XP empowers developers to confidently respond to changing customer requirements, even late in the life cycle. XP also emphasizes teamwork. Managers, customers, and developers are all part of a team dedicated to delivering quality software. The foundation of XP represents the four values:

- Communication,
- Feedback,
- Simplicity, and
- Courage.

The five basic XP principles are used as the guide for development: *Rapid Feedback*, *Assume Simplicity*, *Incremental Change*, *Embracing Change*, and *Quality Work*. The four basic XP activities are coding, testing, listening, and designing. Based on these values, principles and activities the basic XP practices are derived:

- *Planning Game:* Quickly determine the scope of the next release by combining business priorities and technical estimates.

- *Small Releases:* Put a simple system into production quickly, then release new versions on a very short cycle.
- *Metaphor:* Guide all development with a simple shared story of how the whole system works.
- *Simple Design:* The system should be designed as simply as possible at any given moment. Extra complexity is removed as soon as it is discovered.
- *Testing:* Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished.
- *Refactoring:* Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.
- *Pair Programming:* All production code is written with two programmers at one machine.
- *Collective Ownership:* Anyone can change any code anywhere in the system at any time.
- *Continuous Integration:* Integrate and build the system many times a day, every time a task is completed.
- *40-hour Week:* Work no more than 40 hours a week as a rule. Never work overtime a second week in a row.
- *On-site Customer:* Include a real, live user on the team, available full-time to answer questions.
- *Coding Standards:* Programmers write all code in accordance with rules emphasizing communication through code.

Many of these practices are old, tried and tested techniques, but often forgotten by many, including most planned processes. XP integrates them into a synergistic whole where each one is reinforced by the others.

XP defines the following lifecycle phases of an ideal project: *Exploration*, *Planning*, *Iterations to First Release*, *Productioning*, *Maintenance* and *Death*. According to this, XP defines the main human roles in a typical XP project: Programmer, Customer, Tester, Tracker, Coach, Consultant and Big Boss. XP is perfect for small to medium teams; the team size should be between two and 12 project members. Communication and coordination between project members should be enabled at all times, so they should be even physically collocated. However, the geographical distribution of teams is not necessarily outside the scope of XP in the case it includes two teams working on related projects with limited interaction (Beck, 1999). Similar to other agile methodologies, XP minimizes the efforts invested in modeling and up-front architectural design. For exploration and planning purposes XP uses user stories, which are the light, textual version of use cases, while for design purposes the XP team uses Class-Responsibility-Collaborator (CRC) cards, sketches of, e.g., classes, prose text and refactoring. For representing a system architecture XP uses a metaphor—a simple textual representation of the basic elements of the system and their relationships. Along with that XP can use so-called architecture spikes that provide quick explorations into the nature of a potential solution. XP does not require any tool support, except a simple whiteboard for drawing necessary sketches as well as Story and CRC cards.

Feature-Driven Development

Feature-Driven Development (FDD) is a software development process for producing frequent, tangible, working results (Coad, Lefebvre & DeLuca, 1999; Palmer & Felsing, 2002). FDD was used for the first time in the development work of a large and complex banking application project in the late 90's. FDD consists of five sequential processes during which the design and building of the system is carried out: *Develop an overall model*, *Build a Feature list*, *Plan by Feature*, *Design by Feature*, and *Build by Feature*. While the first three processes are sequential, the next two (Design and Build) are the iterative part of the FDD. FDD concentrates on the concept of feature. A feature is a small, client-valued function expressed in the form <action><result><object>. As there may be a lot of features, FDD recommends that features are organized into a hierarchical list: major feature set, feature set, and feature steps. Features are determined from the formal requirements specification. FDD suggests that each feature should take no more than two weeks to design and develop.

FDD defines the six key human roles and nine supporting roles. The main role is that of Chief Programmer(s), who is responsible for planning the schedule of features, allocating the ownership of classes, delivering a feature set, ensuring quality of all products, and leading the feature teams. The FDD development approach has been applied to projects of various sizes (from 50 people up to 250), claming to contain just enough process to ensure scalability. Unlike some other agile methodologies, FDD claims to be suitable for the development of critical systems.

As following the classical object-oriented paradigm, all diagrams and notation in FDD are based on the UML with supporting textual documents for representing, e.g., the feature list. For domain object model, a class diagram is used, specifying operations and attributes of the classes and associations between them. For the list of features a textual notation is used. The features can be derived from use cases of the system. In designing the system a sequence diagram can be used for associating features to particular objects/classes. FDD does not suggest using any specific tool support, but it can be assumed that for object modeling and sequence diagrams a UML-based tool can be used. The domain object model through different levels of detail serves as an architecture blueprint of the system. When the features are later listed it is natural to map them onto the existing domain classes. That is a data-centric view of the domain and may not be the best structure for the solution.

Agile Modeling

Agile Modeling (AM) has been proposed as an attempt to apply AD and XP principles to modeling and design activities (Ambler, 2002). As in the case of XP, Agile Modeling is a collection of values, principles and practices for modeling software that can be applied in an effective and lightweight manner. The agility and effectiveness of the modeling are achieved by the fact that the models are as simple as possible, easily understandable, sufficiently detailed, accurate and consistent. AM is not a prescriptive process, i.e., it does not define detailed procedure to create a given type of model; instead it provides advice on how to effectively model and produce a quality product that matches business needs. The focus of AM is on effective modeling and documentation. It does not include programming and testing activities, project management, and system deployment and maintenance. Therefore AM should be used with another complete method such as XP, DSDM or RUP, where it can provide a way of effective and agile modeling and design (Figure 1).

Figure 1: AM enhances other software processes

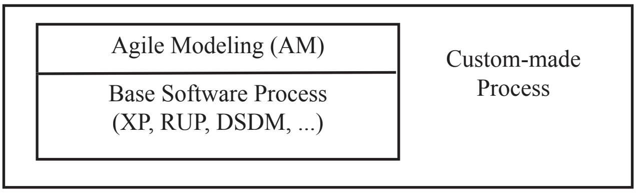


Table 1: Core and supplementary principles of Agile Modeling

Core AM principles	**Supplementary AM principles**
Software is your primary goal	Content is more important than presentation
Enabling the next effort is secondary goal	Know your models
Travel light	Everyone can learn from everyone else
Assume simplicity	Local adaptation
Embrace change	Open and honest communication
Incremental change	Work with people's instincts
Model with a purpose	
Multiple models	
Quality work	
Rapid feedback	
Maximize stakeholder investment	

The AM methodology is a collection of practices, guided principles and values. The values of AM, similar to those of XP, are *communication*, *simplicity*, *feedback*, *courage*, and *humility*. The keys to modeling success according to AM are to have effective communication between all project stakeholders, to strive to develop the simplest solution possible that meets all of your needs, to obtain feedback regarding your efforts often and early, to have the courage to make and stick to your decisions, and to have the humility to admit that you may not know everything. The core and supplementary AM principles are derived from the XP (Table 1).

The heart of AM is its practices that are guided by the AM values and principles. AM core practices are organized into four categories:

(1) Iterative and Incremental Modeling
  - Apply the right artifacts,
  - Create several models in parallel,
  - Iterate to another artifact,
  - Model in small increments.

(2) Teamwork
  - Model with others,
  - Active stakeholder participation,
  - Collective ownership,
  - Display models publicly.

(3) Simplicity
  - Create simple content,
  - Depict models simply,
  - Use the simplest tools.

(4) Validation
  - Consider testability,
  - Prove it with code.

AM includes supplementary practices that support its core practices and that the team can optionally adopt. They are also organized in categories:

(1) Productivity
  - Apply modeling standards,
  - Apply patterns gently,
  - Reuse existing resources.

(2) Documentation
  - Discard temporary models,
  - Formalize contract models,
  - Update only when it hurts.

(3) Motivation
  - Model to communicate,
  - Model to understand.

It is obvious that AM practices are not new; they are techniques that modelers have been following for years, but Scott Ambler, the author of AM, claims that they have been packaged for the first time together and represented as a modeling framework. AM differentiates between two types of modeling tools: simple tools (sheet paper, whiteboards and index cards) and advanced CASE tools. Both of them have their advantages and disadvantages in relation with the AM principles and practices. Generally agile modelers should select the simplest tools that are best suited for the particular project regarding the added value and investments in learning and working. AM does not precisely define human roles, but makes suggestions on how agile work areas should look and how to organize an effective AM team.

Since many AM principles and practices are derived from the XP ones and map straight to XP, there is a clear potential of AM to fit well with XP and add value to an XP project. AM can be applied throughout the XP life cycle, especially during *Exploration*, *Planning* and *Iteration to Release* phases. For this purpose, the sketches (or CASE-made diagrams) of use cases, architectural overviews, UI screens, data models, class diagrams and sequence diagrams can be used. Regarding the RUP, it can be noticed that many of the AM principles and practices are already a part of the RUP, although perhaps not as explicitly as stated in AM. This is because the RUP is very flexible, and can be tailored to meet particular needs, making it easy to merge AM practices into the RUP. Both the RUP and AM are based on the incremental and iterative strategy of software development. However, according to Ambler, for the purpose of Agile Modeling the RUP and UML should be extended with other modeling artifacts, e.g., for representing business concepts and processes, navigational flows within the user interface, data models of the physical design of the database, user stories and CRC cards. The agility in using AM on top of the RUP is not achieved by using fewer modeling artifacts (the case is even opposite) but rather by focusing on the practices of *Apply the right artifacts*, *Use the simplest tools* and *Discard temporary models,* among others.

Extreme Modeling

Extreme Modeling (XM) (Extreme, 2003) is a software development process that tries to make a synthesis of model-based processes and Extreme Programming (XP), but in a different way than agile modeling. XM is still a subject of investigation and research. Several papers are published on this topic, and there is a dedicated web site with basic information. XM unites UML-based modeling principles and XP and combines their advantages by applying the tenets of XP to the modeling phase. For the successful integration of the two, there are two basic requirements that have to be met: models need to be executable and they must be testable. Therefore XM requires intensive support by an integrated tool that is able to execute UML models, test models, support the transition from models to code and keep the code and model in sync. According to the authors of XM, a critical set of the necessary tools already exists. At the University of Hamburg, an implementation based on an open source UML tool called Argo/UML and a Petri nets tool called Renew has been developed. It is currently able to execute state, activity, collaboration and sequence diagrams. The translation of these UML diagrams to the corresponding Petri nets representation works for almost all complex diagram elements, including forks/joins, complex states, history states, transition guards and actions. This allows the execution and visualization of UML diagrams as well as to express tests on models. Recently, XM has stopped using Petri nets as an intermediary step between UML models and code, and now translates directly from models to code. XM represents a promising approach and has a close relationship to the OMG's MDA initiative (OMG, 2003). XM is strongly based on the required tool support, which is the matter of further research and investigation.

XM tries to bridge the gap between traditional development and code-focused extreme programming by introducing executable and testable models that are supported by advanced tools. That includes transformations of models of different levels of abstractions, as well as an extensive code generation based on these models. The approach of Extreme Modeling is in line with some other approaches focused on the concept of executable models, such as Executable UML (Mellor & Balcer, 2002). Executable models are the main products of the development process, translated directly into bits using compilation software. In this sense, the models are actually the code. The models are exact graphical representations of the software structure. This will certainly represent one of the major research directions in software engineering in the future.

Scrum

Scrum is an empirical approach applying the ideas of industrial process control theory to systems development with the ideas of flexibility, adaptability and productivity (Schwaber & Beedle, 2002). It does not define any specific software development techniques for the design and implementation phase. Scrum concentrates on how the team members should function in order to produce the system in a constantly changing environment. There have been some efforts recently about integrating Scrum and XP, where Scrum should provide the project management framework. Scrum process defines three main phases: pre-game, development and post-game. Development phases should be done in seven to 30 days-long iteration cycles called *sprints*. Scrum keeps two stacks of cards containing features that should be developed. One stack has an ordered set of features for the whole system, and the other has those features to be executed in the current 30-day sprint. Scrum involves frequent management activities and daily meetings called scrums in order to identify and correct any deficiencies or obstacles in the development process. Scrum defines team members' roles and responsibilities and is suitable for small teams up to 10 people.

Adaptive Software Development

Adaptive Software Development (ASD) was developed by Jim Highsmith and published in Higsmith (2000). Many of ASD principles are derived from Highsmith's earlier research on iterative development methods. ASD focuses mainly on the problems in developing complex, large systems. The method strongly encourages incremental, iterative development with constant prototyping. ASD suggests the importance of collaborating teams and teamwork and building an adaptive organizational culture, but proposes very few practices for day-to-day software development work. That is why there is a space for this method to be accompanied with the development practice of XP, for example. ASD process includes three phases—*Speculate*, *Collaborate* and *Learn*—performed in the cycles. ASD is explicitly feature-based (or component-based) rather than task-oriented, which means that the focus is on results and products rather than the tasks for producing them. ASD does not propose the team structure in details, and does not enforce that the team member must be co-located like most other agile methodologies.

Crystal Method Family

The Crystal family of methodologies includes a number of different methodologies, as well as principles for tailoring them to fit into the varying nature of different projects (Cockburn, 2002). The family consists of four methods—Clear, Yellow, Orange, and Red—with the principle, 'the darker the color, the heavier the methodology'. There are certain rules, features and values that are common to all Crystal methods. The projects always use incremental development cycles with a length between one and three months. The emphasis is on communication and cooperation of people. Crystal methodologies do not limit any development practices, and therefore can easily adopt XP practices. The Crystal methods use the common work products from XP, such as stories/use cases, screen drafts and design sketches. Crystal Clear is suited for small projects and small co-located teams (up to six developers) with precisely defined roles. Cockburn's Crystal Family is now merged with Highsmith's Adaptive Systems Development.

Dynamic Systems Development Method

Dynamic Systems Development Method (DSDM) was developed in 1994 in United Kingdom as a framework for rapid application development (RAD) (Stapleton, 1997, 2003). DSDM is a non-profit and non-proprietary framework maintained by the DSDM Consortium (DSDM, 2003). DSDM has underlying principles that include active user interaction, frequent deliveries, empowered teams and testing throughout the life cycle. Three major phases of DSDM—functional iteration, design-and-build iteration and implementation—are themselves iterative processes. DSDM suggests making a series of prototypes in short cycles to gradually solve the problems that are not precisely defined in advance, or not stable during that time. DSDM does not offer detailed documentation of its work products; rather, a brief description, list of purposes and several quality criteria. The DSDM process assumes that the time is fixed for the life of a project and project resources are fixed as far as possible, while the requirements that will be satisfied are allowed to change, which is largely opposite to traditional development processes. Although there is evidence (white papers) about combining DSDM with UML and RUP, supporting materials and white papers are available only for consortium partners for an annual cost. DSDM defines 15 roles for users and developers. The team consists of between two and six members, and several teams can exist in the project. According to its authors and users, DSDM has proved to be one of the most successful frameworks for agile software development. The new book on DSDM (Stapleton, 2003) has been updated to reflect recent changes in the framework, as well as experiences and results in applying it in practice.

ANALYSIS OF AGILE METHODS

In this section we will summarize the main characteristics of presented agile methodologies and provide their comparison. In the sequel, special attention will be put to analyzing the support of modeling and design activities that exist in XP, FDD, AM and XM, while other methodologies are not taken into account because of the lack of a development and modeling practice. According to Sol (1983) the analysis and comparison of software development methods can be approached in five different ways:

- Describe an idealized method and evaluate other methods against it.
- Distil a set of important features inductively from several methods and compare each method against it.
- Formulate a priori hypotheses about the method's requirements and derive a framework from the empirical evidence in several methods.
- Define a meta-language as a frame of reference against which you describe many methods.
- Use a contingency approach and try to relate features of each method to specific problems.

Our goal here is to identify differences and similarities between different agile software development methods. Therefore, we will use the combination of the second and the fourth approach in comparing the methods. The methods will be analyzed through the chosen set of important features concerning the method, its usage and adoption, key process characteristics as well as the support to modeling and architecture design activities.

Comparison of Basic Characteristics

Agile Methodologies will be analyzed and compared using several sets of criteria. Key characteristics, special features and shortcomings of agile methodologies are shown in Table 2. The current state of the methodologies, the level of documentation and their adoption in practice are shown in Table 3, while Table 4 analyzes certain aspects of the development processes of agile methodologies.

Table 2: Characteristics, special features and shortcomings of AMs

	Key characteristics	**Special features**	**Shortcomings**
XP	Customer-driven, frequent releases, pair programming, face-to-face development	Refactoring, test-first development, constant testing, simple design through metaphors	Little about management practice, not scalable, team members must be co-located
FDD	Five basic process steps, short iterations, feature-centered, use UML diagrams, developing features in up to two weeks	Combining features and object modeling, applicable to the projects of various sizes, applicable for developing mission critical systems	Management support needed, not sophisticated, more data-centric view on system
Agile Modeling	Applying agile principles and practices to modeling, XP with modeling and without programming	Can fit well into different processes (XP, DSDM or RUP), use UML, ER, business process modeling, etc.	Not complete process, need other development methods, basically restatements of XP principles for modeling
Extreme Modeling	Integrating model-based and XP principles, executable models, models are code, in line with MDA	Tool support needed, models testable and executable, use UML and translate it directly to code (no more Petri nets)	The method and tools under development, promising but needs more support
Scrum	Small teams up to 10, iterations (sprints) seven to 30 day cycles, three main phases, widely applicable	Daily meetings (scrums), possible integration with XP for development practice, easily customizable	Little about development practice, not sophisticated
ASD	Adaptive organizational culture, collaborative teamwork, three main phases, combined with Crystal family	Non-linear overlapping life cycle phases, component-based, rapid prototyping, members need not be co-located	Lack of software development practice, not detailed team structure
Crystal family	Family of methods, adaptable to different project size and complexity, combined with ASD	Features and values common to the whole family, small teams, 1-3 month cycles, can use XP development practice	Not complete, not enough supporting materials
DSDM	Controlled RAD variant, supported by consortium, represents a framework for development, three main iteration phases	Use of prototyping, several small teams (two to six people), can be combined with RUP and UML	Limited access to supporting materials

Table 3: Current state, documentation and practical experiences of AMs

	Maturity	**Documented**	**Adoption and Experience**
XP	Active	Books, papers, web sites	Applied in practice
FDD	Active	Books, papers, web sites	Applied in practice
Agile Modeling	Still under development	Book, papers, web sites	No evidence of applying
Extreme Modeling	Still under development	Web sites, papers	Applied to some extent
Scrum	Active	Book, web sites	Applied in practice
Adaptive SD	Active	Book, papers, web sites	Applied to some extent
Crystal family	Still under development	Web sites, web documents	Applied to some extent
DSDM	Active	Book, web sites, papers (limited to DSDM members)	Applied in practice

Table 4: Development process characteristics of Agile Methodologies

	Process Support	**Iterative and Incremental**	**Scalability – Size of teams**	**Defined Member Roles**	**Modeling Activities**
XP	Development practice	Yes	Up to 12	Yes	Minimized
FDD	Development and partly management process	Design and build phases	From 50 up to 250	Yes	Minimized
Agile Modeling	Modeling	Yes	No restrictions	Not detailed	Complete
Extreme Modeling	Development process	Yes	No restrictions	N/A	Models are executable and testable
Scrum	Management process	Yes	Up to 10 people	Yes	Not applicable
ASD	Management process	Yes	No restrictions	Not detailed	Not applicable
Crystal family	Partly development and management process	Yes	From six (Clear) up to 40 (Yellow)	Yes	Not applicable
DSDM	Management process	Yes	Possibly many small teams (from two to six)	Yes	Not applicable

Modeling and Design in Agile Methods

In this section, the kind and nature of modeling and design support will be analyzed in XP, FDD, Agile Modeling and Extreme Modeling, since they describe practices that are related to modeling and architecture design, as shown in Table 5.

Table 5: Modeling and design activities in the selected set of Agile Methodologies

	XP	FDD	Agile Modeling	Extreme Modeling
Scope	Development process	Development process	Modeling and design	Development process (not detailed)
Amount of modeling	Low	Low	Middle	Middle
Tool support	No	Not specified (possible UML-supportive)	The simplest possible, e.g., whiteboard	Extensive support (transfer model to code)
Notation	User stories, CRC cards, Sketches of classes	Features, Objects/ Classes, Sequence diagrams	UML + User stories, UI screens, Data modeling, etc.	Formerly Petri nets, now standard UML and code
Architecture design	Metaphor, Architecture Spikes, First Release	Based on object model	Domain packages	Based on object orientation
Requirements elicitation	User stories	Features	Use cases + user stories	Use cases
Using components	No	No	To some extent	To some extent
Business modeling	No	No	Yes	No
Model repositories	No	N/A	No	Yes
Designing large systems	No	To some extent	Yes	Yes
Incremental	Yes	Yes	Yes	Yes
Iterative	Yes (very small increments)	Yes, only two phases	Yes, sequential and iterative	Model and code constantly in sync
Complexity	Low	Low	Middle	Middle – advanced tool support
Model reusability	No	No	No	Yes

Agile vs. Traditional Methods

It is obvious that Agile Methodologies claim to address the challenges of high change rates, short time-to-market, increased return-on-investment and high quality software by emphasizing communication between project stakeholders, iterative and incremental development with the focus on software code, as well as high response to changes in requirements. On the other hand, traditional, more formal methodologies, such as Rational Unified Process (RUP) (Jacobson et al., 1999) and Catalysis (D'Souza & Wills, 1999) suggest rigorous modeling and a clear plan to follow in transferring business requirements into working software. The current Object Management Group initiative over Model-Driven Architecture is in line with that way of thinking. MDA stresses the importance of a platform-independent and platform-specific model to separate abstract domain knowledge from concrete imple-

Table 6: Agile and traditional methods

	Agile methods	Plan-driven methods
Developers	Agile, knowledgeable, co-located, and collaborative	Plan-oriented, adequate skills, access to external knowledge
Customers	Dedicated, knowledgeable, co-located, collaborative, representative and empowered	Access to knowledgeable, collaborative, representative, and empowered customers
Requirements	Largely emergent, rapid change	Knowable early; largely stable
Architecture	Designed for current requirements	Designed for current and foreseeable requirements
Refactoring	Inexpensive	Expensive
Size	Smaller teams and products	Larger teams and products
Primary objective	Rapid value	High assurance

mentation environment. These are so-called plan-driven (or design-driven) methods, where the requirements at the beginning of the project are largely stable so that the fixed, well-thought plan can be followed during the development process (Boehm, 2002). Both types of methodologies, agile and traditional, have their advantages and shortcomings depending on particular project settings. Making a decision of using a particular method in a software development project is in strong relation with the project nature, project environment and involved stakeholders (Boehm, 2002) (Table 6).

Both agile methods and more traditional methods try to handle the software development process under the constant changes in the environment. However, their focus is different as to what types of changes they primarily deal with. Agile methods are focused on potential changes of business requirements that can evolve during the project up to the final release. Therefore, they propose mechanisms to flexibly capture these requirements, while the challenges in making technology choices are left implicit, and are under the responsibility of developers. On the other hand, more traditional, model-driven methods try to preserve efforts made in constructing software architecture and design under the changes in available advanced technology solutions. For these methods, business requirements are more or less fixed, written down in the form of contract between business users and developers. In reality, it is essential to provide an effective strategy and mechanisms for protecting software solutions from possible changing requirements coming from both sides. In the sequel of the paper, we propose an implementation-independent, component-based approach for creating flexible system architecture in an agile way and therefore provide a way of balancing between business needs and technology solutions. Components as design level artifacts, not just implementation code packages as suggested by the UML, can become a central point of a new agile, service-oriented development approach.

INTEGRATING COMPONENTS AND AGILE DEVELOPMENT

Common to all agile methodologies that propose some development practice is that they assume an object-oriented development paradigm. XP creates CRC cards and object diagrams, FDD combines objects and features, Agile Modeling extends the standard UML with additional diagrams, while Extreme Modeling follows standard UML and uses the tools for creating executable and testable models. Interestingly, these methodologies do not include or use any advanced concepts such as components (D'Souza & Wills, 1999) and services (IBM, 2003). In our opinion, exactly components as providers of services representing concepts at a higher level of abstraction than traditional classes/objects can significantly support principles and practices of agile development. Component thinking can provide mechanisms for effective and agile design up-front that can be straightforwardly mapped to software code. We strongly believe that the implementation-independent, service-based component concept can be used as the mechanism for balancing agility and discipline in a software development project.

Service-Based Component Concept

Components have been so far used mainly as implementation artifacts. However, the components are equally useful and important if used as modeling and design artifacts in building the logical architecture of the system. The essence of the component approach is the explicit separation between the outside and the inside of the component. This means that only the question of WHAT is considered (what useful services are provided by the particular building block to the context of its existence), not the HOW (how these services are actually implemented). In the sequel, some important component properties used in architectural design and development will be listed, while more details on a component-oriented design and development approach can be found in Stojanovic and Dahanayake (2003a).

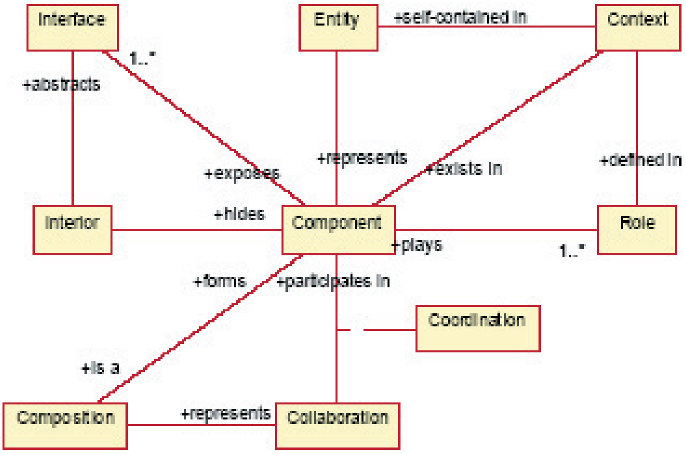
A component fulfils a particular role in the context by providing and requiring services to/from it. A component has a hidden interior and exposed interface. It participates in a composition with other components to form a higher-level behavior. At the same time every component can be represented as a composition of lower-level components. Well-defined behavioral dependencies and coordination of activities between components are of great importance in achieving the common goal. The metamodel of the basic component concepts is shown in Figure 2.

According to the role(s) a component plays in the given context, it exposes corresponding behavior by providing and requiring services to/from its context, or by emitting and receiving events. The services a component provides and requires are the basic part of its contract. Services can be of different types, such as performing computation, providing information, communication with the user, etc. They are fully specified in a contract-based manner using pre-conditions, post-conditions and other types of constraints. A component must handle, use, create or simply be aware of certain information in order to provide its services properly. In order to be used in a different context or to be adaptable to the changes in its context, a component can possess so-called configuration parameters that can adapt the component according to new requirements coming from the outside. A component can possess a set of so-called non-functional parameters that characterize the "quality" of its behavior. Figure 3 shows the component specification concepts.

Component-Orientation in Agile Development

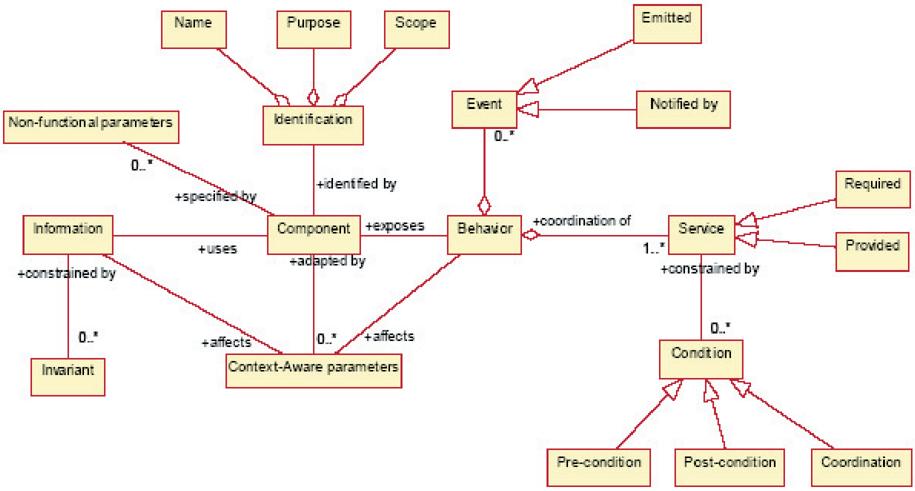
In our opinion, component concepts presented above represent the clear case for agile design and development. They can support the most important principles of AD, such as simplicity, good communication between stakeholders, rapid feedback, and effective adoption of changes. By focusing on components in representing the problem and propos-

Figure 2: Basic component concepts



ing the solution, the effective "divide-and conquer", separation-of-concerns strategy is applied, which makes both the problem and solution simpler and easier to understand and manage. Service-based component concepts are equally well understood by both business and technical people because they are defined at a higher level of abstraction than, e.g., OO objects and classes. In this way, the component-based architecture can provide a common ground and the point of communication and negotiation between all involved stakeholders. Higher-level component-based vocabulary can become a common language between domain experts and software developers.

Figure 3: Component specification concepts



Components are by their nature an excellent way to manage changes. Changes are not harmful for component-based software since they are localized inside the particular component or on the interfaces between components, so they cannot be spread across the system in an uncontrolled manner. By defining higher-level, business-driven component-based architecture that balances business needs and software implementation, developers protect their investments under changing requirements, and at the same time easily map the architecture into software code using e.g., advanced model transformers and code generators. Components can support high quality work, which is one of the main targets in an agile project, since, if used, COTS components or Web Services are usually pre-tested and certified through a number of previous usage cases.

Component and service concepts can add significant value to the simple and easily scalable architectural design in agile development. As stated earlier, agile methodologies do assume certain design activities but they perform them in a different manner than traditional methodologies. In our opinion, component concepts can play a significant role in agile architecture modeling, design and planning game. Since components are defined as providers of business services at a higher level of abstraction and granularity than traditional objects, they can be used as the main building blocks of a simple architectural design understandable for all involved project stakeholders. By its definition a component hides a complexity of its interior so that components can help in more easily making an architecture metaphor and architecture prototypes (spikes) as the main design-level artifacts in an agile project. Components as a mechanism for organizing business requirements in cohesive business service providers, and at the same time a blueprint of future implementation, can provide bi-directional traceability between business needs and software artifacts. That certainly helps in better understanding and communication across the project, and more straightforward iterations and increments. Good business-driven, component-oriented architecture design can reduce the need for refactoring (that can be also a time-consuming task), as well as permanent customer presence, in the case it is not feasible.

The representation of the component in designing architecture can vary in the level of details, which depends on the project nature. The component can be described through its main properties defined briefly above, as well as in Stojanovic and Dahanayake (2003a), using different mechanisms at different levels of formality, such as natural language and sketches on the whiteboard, business vocabulary on some kind of the cards, formal specification language and contract-based theory in a CASE tool, or software code. Hence, the level of details in specifying components can be truly scalable, depending on the project settings. In this way the same component-approach can fit into really agile projects, as well as large, safety-critical projects and teams, depending on particular needs.

Among the main issues in AD are test-driven design and the test-before-implementation principle. Since components are identified and defined based on use cases they are responsible for, as well as information types used by these use cases, it is easy to define test suite for components based on use cases and conditions related to them. These tests can guide an effective implementation of components that fulfill given business needs. These tests represent agile black-box unit tests. By putting components together in a plug-and-play fashion, the whole release can be easily tested as well. Components are well suited for some coding and implementation practices defined in AMs, such as continuous integration, using coding standards and pair programming. The first release as an important artifact in a XP project can be easily constructed using components, where some or most of them have just defined interfaces (so-called dummy interfaces) without real implementation. Components are also well suited for incremental and iterative development and fast, small solution releases. Each new iteration cycle adds more functionality to particular components and refines the interfaces among them. Coding practice can be applied at the level of single components, their sub-components, or a set of collaborative components.

Agility in Component-Based Development

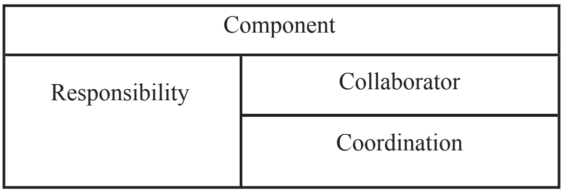
Component-based development can also benefit from certain principles and mechanisms used in agile development processes. The main idea behind code refactoring—changing interior while preserving exposed behavior and semantics—can be effectively applied in designing flexible component-based architecture. We propose here a process called component refactoring, which aims at reallocating and rearranging sub-components of the component being addressed, while preserving its contractual behavior. Component refactoring can be performed at two levels. At the first level, the high-level business components are identified and constructed based on business requirements they fulfill and business services they offer. However, the decision of allocating use cases to particular business service components is not straightforward, and other criteria, such as existing legacy, business rules, and information placement must be taken into account. Furthermore, for every system being developed, a number of so-called changing cases can be defined that can be modified in the future. The process of component refactoring should ensure that the business needs are fulfilled by the well-architected set of business components following the principles of lowest coupling and highest cohesion. At the level of application architecture, the component refactoring practice can support placing lower-level application components into higher-level business components. This is especially important in the case when data or computation redundancy must be avoided, i.e., when an application component is used by several business components and it must be decided what business component is really responsible for it.

For representation components during the development process we can use standard UML diagrams enriched with proper extensions (stereotypes and tagged values). For the purpose of lightweight component modeling that can be easily understood by both business people and software architects, we propose a new variant of well known Class-Responsibility-Collaborator (CRC) cards, called CRCC or CRC^2 cards, which stands for Component-Responsibility-Collaborator-Coordination (Figure 4). In this way the basic properties of each business component (identifier, responsibility, collaborating components and coordination) are specified without going too much into detail, according to AD principles and practice. At this level, the system architect can communicate and negotiate with the business user to see whether this initial business component architecture captures all user requirements. Defined cards can provide fast and easy design of the initial business-driven component-oriented architecture that can be understood by both business and IT sides, and can be used according to XP principles of simple design, architecture metaphor, rapid feedback and extensive user involvement. At the same time, the stack of CRC^2 cards can be used for planning future developments, as well as for delivering tasks (i.e., component development) to particular developers.

Components and Agile Development Limitations

It is obvious that depending on the kind and nature of the system being developed, its domain and environment, as well as involved stakeholders, an agile development process can represent either the right or not proper solution. If short time-to-market and extreme

Figure 4: Component-Responsibility-Collaborator-Coordination card



flexibility of the system under the constantly and rapidly changing requirements are of the highest importance, then that is the case for agile development. If the system being built is rather complex, safety-critical and supposed to be stable over the longer period of time, then the agile development process may not be the right solution (Turk, France & Rumpe, 2002). The following are the limitations of agile methodologies in terms of the types of projects where they cannot potentially provide a full support:

- Limited support for projects with distributed development teams and resources.
- Limited support for outsourcing.
- Limited support for building or using reusable artifacts.
- Limited support for using legacy systems or Commercial-Off-The-Shelf (COTS) components.
- Limited support for projects involving large teams.
- Limited support for the development of large software systems.
- Limited support for the development of safety-critical software systems.

In our opinion, using the component paradigm can help in overcoming or mitigating these limitations of agile methodologies. Using the component way of thinking, the whole problem can be divided into pieces according to cohesive sets of business functionality. These functional units, called business components, can be specified according to the defined component properties, in an informal, semi-formal or formal way, depending on a particular situation. The more formal specification of component interfaces can help in communication between team members and customers when customers are separated from developers, or a development team is distributed over several locations. Components can help in an agile project when a particular task should be outsourced to subcontractors. In that case components related to the task can be specified in a more formal way than in an ordinary agile project, in order to provide a precisely defined subcontracted task. This actually represents the specification of the components that are outsourced. Additional flexibility in a sub-contract specification can be achieved using configuration context-aware parameters of components in order to provide easier adoption of possible changing requirements.

Components are about reusability, so each component that normally encapsulates well-defined business or technical functionality can be reused in a similar context in the future. On the other hand, well-defined component-oriented architecture provides using third-party components such as COTS components or wrapped legacy assets, as long as the interfaces toward the other components are fulfilled. In that case an agile project would be responsible for the rest of the system and its interface to existing third-party solutions. By providing an effective separation of concerns, the component paradigm can help in supporting the agile development that involves a larger team in building large software. Large problems can be broken down into smaller units, and then parts of the team are responsible for developing particular components in an agile manner. System made of components can scale and be extended easily by defining additional components or by extending the scope of existing components. Agile Development Processes produce high-quality software through constant unit and functional testing. Using pre-tested COTS components can further increase the quality of the safety-critical software.

CONCLUSIONS

In the past several years, Extreme Programming (XP) and other Agile Methodologies (AMs) have started to gain considerable interest in the IT community. A number of processes claiming to be "agile" have been proposed. The leading agile methodologists have formed the Agile Alliance and published the Agile Manifesto. AMs are focused on communication among project stakeholders, frequent delivery of software releases and an effective coding practice. The role of modeling, design up-front and documentation is significantly minimized. One of the main assumptions in software development is that the requirements and conditions from the environment are in constant change. The focus of Agile Development is on effective mechanisms to adopt changes through iterative and incremental cycles, small releases, frequent testing, and the constant feedback from the customer. The quality of software solution is maintained through refactoring, pair programming, using coding standards and continuous integration of software releases. Although agile methodologies differ in their characteristics, mechanisms, scope and focus, they share similar concepts, principles and practices that challenge many of the common assumptions in software development and initiatives such as Model-Driven Development. While both Agile Development and Model Driven Development claim to address the challenges of high change rates, short time-to-market, increased return-on-investment and high quality software, their proposed solutions are actually very dissimilar.

This chapter presents the state-of-the-art agile methodologies through their important characteristics, main features and identified shortcomings. Agile methodologies are further analyzed and compared through the set of criteria, from their applicability through the aspects of development practice. Special attention is made for highlighting the nature and kind of the support to modeling and architectural activities found in the selected set of methodologies. Finally, the chapter presents how component concepts used at the level of modeling, architectural design and implementation can effectively support the main principles and practices of agile development. Modeling and specifying components as the main building blocks of simple architecture design at a particular level of details can provide a bridge between traditional model-driven and agile development. Using components can help in overcoming certain limitations of agile methodologies in relation with the type and nature of the project, such as reusability, outsourcing, large teams and building large safety-critical software systems. On the other hand, using agile values, principles and practices in current model-driven, rather heavyweight methodologies, such as RUP (Jacobson et al., 1999) and Catalysis (D'Souza & Wills, 1999) can help in more flexible processes and solutions, as well as shorter time-to-market and products that better fulfill business needs. Integrating certain aspects and principles of agile and model-driven development around the component concept can lead to a new, highly flexible and agile, service-oriented software development approach of the future (Stojanovic & Dahanayake, 2003b). In this way, components can become an effective mechanism for balancing agility and formal plan, depending on the project settings and development team culture.

REFERENCES

Agile Alliance. (2001). *Manifesto for agile software development***.** Available: http://www.agilealliance.org

Ambler, S. (2002). *Agile modeling: Effective practices for eXtreme programming and the Unified Process***.** New York: John Wiley & Sons.

Beck, K. (2000). *Extreme programming explained – Embrace change***.** Reading, MA: Addison-Wesley Longman.

Coad, P., Lefebvre, E., & DeLuca, J. (1999). *Java modeling in color with UML: Enterprise components and process.* Upper Saddle River, NJ: Prentice Hall.

Cockburn, A. (2002). *Agile software development.* Boston, MA: Addison-Wesley.

DSDM. (2003, September 1). *Dynamic systems development modeling consortium*. Available: http://www.dsdm.org

D'Souza, D.F., & Wills, A.C. (1999). *Objects, components, and frameworks with UML: The Catalysis approach.* Boston, MA: Addison-Wesley.

Extreme Modeling. (2003, September 1). Available: http://www.extrememodeling.org

Highsmith, J.A. III (2000). *Adaptive software development: A collaborative approach to managing complex systems*. New York: Dorset House Publishing.

IBM. (2003, September 1). *Web Services*. Available: http://www.ibm/com/webservices

Jacobson, I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process.* Reading, MA: Addison-Wesley.

Jeffries, R., Anderson, A., & Hendrickson, C. (2001). *Extreme programming installed.* Boston, MA: Addison Wesley.

Larman, C. (2001). *Applying UML and patterns.* (2nd Edition). Prentice Hall.

Mellor, S., & Balcer, M. (2002). *Executable UML: a Foundation for model-driven architecture*. Boston, MA: Addison-Wesley.

OMG. (2003, September 1). *Object Management Group, MDA- Model Driven Architecture.* Available: http://www.omg.org/mda/

Palmer, S.R., & Felsing, J.M. (2002). *A practical guide to feature-driven development.* Prentice Hall.

Schwaber, K., & Beedle, M. (2001)**.** *Agile software development with Scrum.* NJ: Prentice Hall.

Sol, H.G. (1983). A feature analysis of information systems design methodologies: Methodological considerations. In T.W. Olle, H.G. Sol and C.J. Tully (Eds.), *Information systems design methodologies: A feature analysis* (pp. 1-8). Amsterdam, The Netherlands: Elsevier.

Stapleton, J. (1997). *DSDM: Dynamic Systems Development Method***.** Harlow, London: Addison Wesley.

Stapleton, J. (ed.) (2003). *DSDM: Business focused development*. DSDM Consortium. (2nd Edition). London: Addison Wesley.

Stojanovic, Z., & Dahanayake, A.N.W. (2003a). A service-based approach to components for effective business-IT alignment. In Joan Peckam (Ed.), *Practicing software engineering in the 21st Century*. Hershey, PA: Idea Group Publishing.

Stojanovic, Z., & Dahanayake, A.N.W. (2003b). Component-oriented agile software development. *Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*, May 26-29, 2003, Genova, Italy.

Turk, D., France, R., & Rumpe, B. (2002). Limitations of agile software processes. *Third International Conference on eXtreme Programming and Agile Processes in Software Engineering,* Sardinia, Italy, (pp. 43-46).

Chapter II

Comparing Metamodels for ER, ORM and UML Data Models

Terry Halpin, Northface University, USA

ABSTRACT

This chapter provides metamodels for some of the main database modeling notations used in industry. Two Entity Relationship (ER) notations (Information Engineering and Barker ER) are examined in detail, as well as Object Role Modeling (ORM) conceptual schema diagrams. The discussion of optionality, cardinality and multiplicity is widened to include Unified Modeling Language (UML) class diagrams. Issues addressed in the metamodel analysis include the normalization impact of non-derived constraints on derived associations, the influence of orthogonality on language transparency, and trade-offs between simplicity and expressibility. To facilitate comparison, the same modeling notation is used to display each metamodel. For this purpose, ORM is used because of its greater expressibility and clarity.

INTRODUCTION

To ensure the correctness and completeness of an information system being developed, requirements analysis should precede its design and implementation. The analysis phase leads to a conceptual schema that specifies the structure of the universe of discourse (application domain). This conceptual structure should be capable of being readily understood and validated by the domain expert, without requiring this subject matter expert to understand technical aspects of the internal structure used to actually implement the application. Once validated, the conceptual schema can be mapped to logical/physical/external schemas using procedures that are partly or fully automatable.

For industrial database work, the traditional approach for high level data modeling is to use a version of Entity Relationship (ER) modeling (Chen, 1976), such as the Information Engineering (IE) approach (Finkelstein, 1998), the Barker version of ER modeling (Barker, 1990), or IDEF1X (Integration Definition 1 extended). Although the original 1993 version of IDEF1X has a standard metamodel (NIST, 1993), we ignore it here since it is actually a hybrid of ER and relational modeling, and its successor, IDEF1X$_{97}$, also known as IDEFobject (IEEE, 1999), has so far been largely ignored by the marketplace.

More recently, Unified Modeling Language (UML) class diagrams (OMG, 2003) and the Object-Role Modeling (ORM) approach (Halpin, 2001a) have also gained popularity for information modeling. Following its adoption by the Object Management Group (OMG), the UML is now the de-facto standard in industry for object-oriented code design. ORM is a fact-oriented approach that can be used as a conceptual front-end to attribute-based approaches such as ER and UML, and is currently being considered by the OMG's Business Rules Special Interest Group as a candidate for business rule modeling at the computation-independent level.

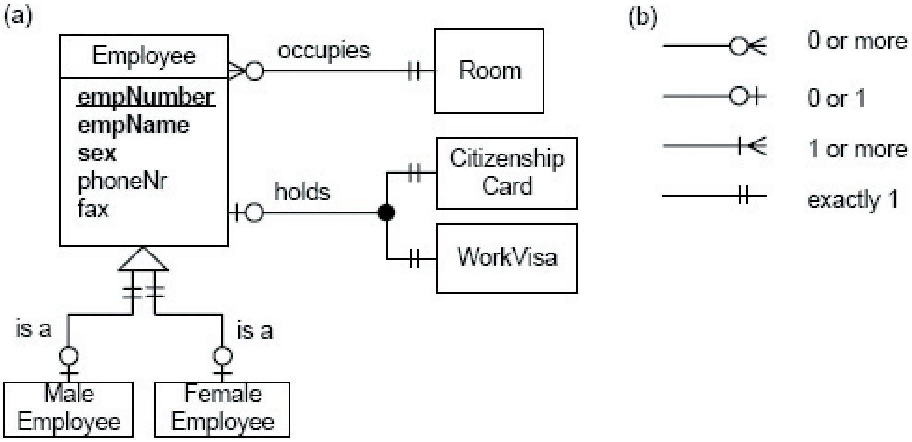
A modeling language can be specified by a *metaschema*, which is a schema that indicates the grammatical structures to which any application schema formulated in the modeling language must conform. Strictly, a *model* is the union of a schema (structure) and a population of instances (e.g., objects or facts that instantiate the information-bearing structures in the schema). A metaschema supplemented by structures to capture specific populations is a *metamodel*. In practice, the term "metamodel" is sometimes loosely used as a synonym for "metaschema". While published metamodels for UML (OMG, 2001, 2003) have been widely debated, and many suggestions have been made to improve UML (e.g., see Siau & Halpin, 2001), it is difficult to find any in-depth analysis of metaschemas for the other approaches. This paper provides new metaschemas for two ER approaches (IE and Barker) as well as ORM to reveal their commonalities and differences, and to address modeling issues such as the use of derived associations and the virtues of orthogonality. UML has been examined previously (e.g., Halpin & Bloesch, 1999; Halpin, 2001b) and is quite complex; hence only an incomplete analysis of its metamodel for data modeling is given here. For a detailed comparative evaluation of all the methods, including IDEF1X, see Halpin (2001a).

The next section of this chapter provides a metaschema and related discussion of the IE notation. The two sections after that metamodel the Barker ER and ORM approaches, respectively. We then evaluate the different approaches to multiplicity in UML, ER and ORM. Some other aspects of the UML metamodel are then discussed. The final section summarizes the main contributions, notes some advantages of an attribute-free modeling approach, and lists references for further reading.

INFORMATION ENGINEERING

The *Information Engineering* approach was originated mainly by Clive Finkelstein, who developed a modeling procedure for the notation and extended IE to Enterprise Engineering (EE). Finkelstein (1998) provides an overview of IE with further details on his website (www.ies.aust.com/~ieinfo/). The IE notation was later adapted by Martin (1993). Although Martin's recent books favor the UML notation, IE is still used far more extensively for database design than UML, which is mostly used for object-oriented code design. Different versions of IE exist, with no single standard. In one form or another, IE has long

Figure 1: (a) A sample, incomplete IE model; (b) The 4 multiplicity patterns



been supported by many data modeling tools, and its simple, intuitive notation has helped it become very popular for database design in industry.

The IE approach depicts *entity types* as named rectangles. *Attributes* are often displayed in a compartment below the entity type name, but are sometimes displayed separately (e.g., bubble charts). Some versions support basic constraints on attributes. For example, an attribute that is part of its entity type's primary identifier might be underlined, and mandatory attributes might be bolded. Although no standard notation exists for these constraints, they are included in our metaschema. The Employee entity type in Figure 1(a) provides a simple example.

Relationships are typically binary only, shown as named lines connecting the entity types. IE usually allows only one reading per association, which must be read left-to-right or top-to-bottom. The line itself corresponds to a binary, logical *predicate*, and the line reading to *predicate text* (e.g., "occupies"). A relationship reading is formed by inserting the entity type names at the start and end of the predicate text (e.g., "Employee occupies Room"). A half-line or line-end corresponds to a *role* in ORM (or association end in UML). In this chapter, we use "role" exclusively to mean "association end". To avoid confusion with other kinds of relationships, we use "binary *association*" for a binary relationship type.

To indicate that a role is *optional*, a circle "O" is placed at the other end of the line, signifying a minimum *multiplicity* (participation frequency) of 0. To indicate that a role is *mandatory*, a stroke "|" is placed at the other end of the line, signifying a minimum multiplicity of 1. A crow's foot is used for a maximum multiplicity of "many". In conjunction with a minimum multiplicity of 0 or 1, a stroke "|" may be used to indicate a maximum multiplicity of 1. So the combination "O|" indicates "*at most one*" and the combination "| |" indicates "*exactly one*".

For example, in Figure 1 the constraints on the association Employee occupies Room specify that each employee occupies exactly one room, and that each Room is occupied by zero or more employees. Some IE notations assume a maximum cardinality of 1 if no crow's foot is used, and hence use just a single "|" for "exactly one". In contrast to our convention, Finkelstein uses the combination "O|" to mean "optional but will become mandatory", which is really a dynamic rather than static constraint—this is excluded from our metaschema.

Some IE versions support an *exclusive-or constraint*, shown as a black dot connecting the alternatives. Figure 1 depicts the situations where each employee holds a citizenship card or a work visa, but not both. Underlying this example, there are two associations: Employee holds CitizenshipCard; Employee holds WorkVisa. The exclusive-or constraint applies to the first roles of these two associations. Although the roles spanned by this constraint are individually optional, their disjunction is collectively mandatory. This is misleadingly depicted by a minimum multiplicity of 1 on the roles at the other end, where the pattern appears as "1 1" or "1 n", although it actually means "0 1" or "0 n" individually. This practice prevents the use of the notation from being adapted to cover simple exclusion constraints.

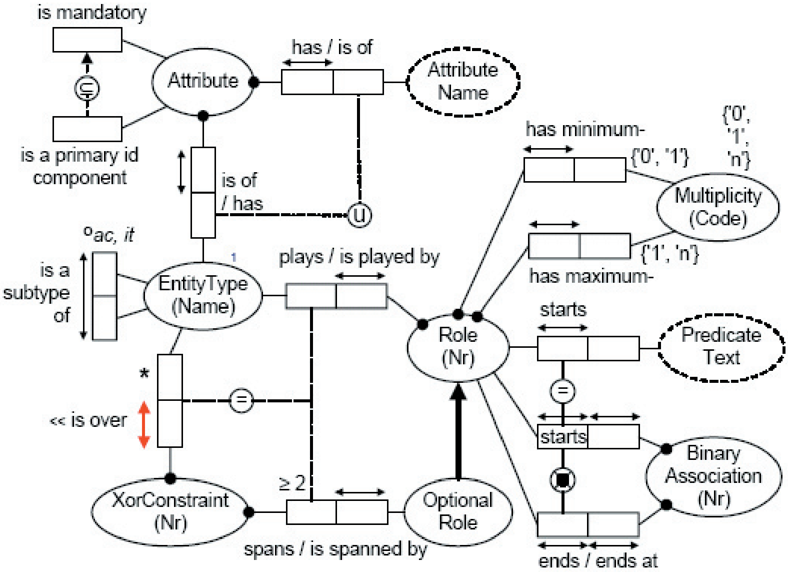
Our metaschema (see later) assumes predicate readings are allowed after the dot. If the predicate reading must be displayed before the dot, the xor constraint can apply only to roles from associations with the same predicate text. In that case, we can't express an xor constraint such as: **each** Employee drives **a** Car **or** catches **a** Bus **but not both**. This restriction does not apply to Barker ER, UML, or ORM. In IE (and Barker ER) the same association role may be spanned by at most one exclusive-or constraint. This restriction does not apply to ORM or to UML.

Subtyping schemes for IE vary. Sometimes Euler diagrams are used, adding a blank compartment for "Other". Sometimes directed acyclic graphs are used, possibly including subtype relationship names and multiplicity constraints (e.g., MaleEmployee and FemaleEmployee in Figure 1). There is no formal support for subtype definitions. Multiple inheritance may or may not be supported, depending on the version.

All our metaschemas use the notation of ORM, a conceptual modeling method that views the world as a set of objects (entities or values) that play roles (parts in relationships, which may be unary, binary or longer). For example, you are now playing the role of being awake (a unary relationship involving just you), and also the role of reading this chapter (a binary relationship between you and this chapter). An entity in ORM corresponds to a nonlexical object (e.g., a country), and a value to a lexical object (e.g., a country code). A role in ORM is a part played in an association, which may be unary, binary or *n*-ary. The main structural difference between ORM and ER or UML is that ORM excludes attributes as a base construct, treating them instead as a derived concept. For example, Person.birthdate is modeled in ORM using the fact type: Person was born on Date. For an overview of ORM see Halpin (1998a; 1998b), and for a detailed treatment see Halpin (2001a). For an in-depth discussion of how ORM is implemented in a Microsoft tool, see Halpin, Evans, Hallock, and MacLean (2003). Many technical discussions of ORM variants are available (e.g., De Troyer & Meersman, 1995; ter Hofstede, 1993; ter Hofstede, Proper, & Weide, 1993).

An ORM metamodel for IE is shown in Figure 2. Entity types are shown as named ellipses, and must have a reference scheme, i.e., a way for humans to refer to instances of that type. Simple reference schemes may be shown in parenthesis (e.g., "(name)"), as an abbreviation of the relevant injective association, e.g., EntityType has EntityTypeName. Value types need no reference scheme, and are shown as named, dashed ellipses. A predicate is shown as an ordered set of one or more role boxes, together with at least one predicate reading. Here we have two unary associations (e.g., Attribute is mandatory) and several binary associations, for which readings in both directions may be shown, separated by a slash "/".

Figure 2: An ORM metaschema for IE



each OptionalRole **is a** Role **that starts a** BinaryAssociation **that ends at a** Role **that** has minimum Multiplicity '0'

[1] In a complete model, **each** EntityType has an Attribute **that** is a primary id component

Note: The derived fact type is defined by its equality constraint but its uniqueness constraint is extra (not derivable).

For each association, or fact type, a sample fact table may be added to help validate the constraints. Each column in a fact table is associated with one role. The arrow-tipped bars are internal *uniqueness constraints*, indicating which roles or role combinations must have unique entries. A black dot on a role connector indicates the role is *mandatory* for its object type. For example, the uniqueness and mandatory constraints on the association Attribute has AttributeName in Figure 1 verbalizes respectively as: **each** Attribute has **at most one** AttributeName; **each** Attribute has **at least one** AttributeName. ORM schemas may be represented in diagrammatic or textual form, and tools such as Microsoft Visio for Enterprise Architects provide automatic transformation between the two representations (Halpin et al., 2003).

The metaschema in Figure 2 assumes a closed world approach for the unary predicates (e.g., if an attribute is not recorded to be mandatory, then it is known to be optional). It also assumes that primary identifier attributes must be mandatory—this is captured by the subset constraint (circled "⊆"), which indicates that the population of the lower role must be a subset of the upper role (i.e., if an attribute is a primary identifier component then it must be mandatory). The external uniqueness constraint (circled "u") indicates that an attribute may be identified by combining its unqualified name with its entity type.

For convenience, roles (association ends) and associations are identified by numbers (whose display is normally suppressed). Roles could also be identified by their position within a standard ordering of an association. Association names are catered for by attaching the predicate text to the role from which the association is read. Equality constraints shown as a circled "=" may also be depicted by a dotted line with arrow-tips at both ends. The fact types Role has minimum- Multiplicity and Role has maximum- Multiplicity use a hyphen to bind the adjective to the object type term for constraint verbalization. So the uniqueness and mandatory constraints verbalize as: **each** Role has **exactly one** minimum Multiplicity; **each** Role has **exactly one** maximum Multiplicity.

In our metaschema, each association role must be annotated by one of the four frequency patterns shown in Figure 1(b). The minimum frequency must be 0 or 1, and the maximum frequency must be 1 or many (denoted here by "n"). These constraints are depicted here as role value constraints {'0', '1'} and {'1', 'n'} beside the relevant roles. Alternatively, this situation may be modeled with object value constraints by using the fact types: Role has MinimumMultiplicity {'0', '1'}; Role has MaximumMultiplicity {'1','n'}. But this alternative makes it awkward to compare minimum and maximum multiplicities, since this must now be done at the lexical level.

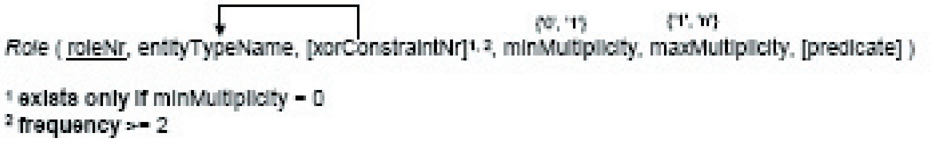
Each xor constraint is identified by a constraint number. The "≥ 2" frequency constraint on XorConstraint spans OptionalRole requires each xor constraint to span at least two roles. The subtype OptionalRole is introduced to ensure that only optional roles may be spanned by xor constraints. A role is optional if it is possible that some instances of its object type's population do not play the role—this is the same as an optional role in ORM. A formal subtype definition is specified textually in FORML (an ORM formal constraint language): this refers to the actual minimum multiplicity (0), not the multiplicity displayed (1). If relevant, further restrictions on the constrained roles (e.g., they must belong to associations with the same predicate text) may be formally specified in an ORM textual language such as FORML or ConQuer (Bloesch & Halpin, 1997).

Subtyping is modeled by the meta-association EntityType is a subtype of EntityType. This allows for multiple inheritance, as well as incomplete sets of subtypes. Single inheritance may be enforced by strengthening the uniqueness constraint to apply to just the first role (so the association becomes n:1). If more than one subtype must be introduced, this can be enforced by adding the frequency constraint "≥ 2" to the second role. The °ac,it annotation declares the subtyping association to be acyclic and intransitive, allowing only direct, proper subtype connections.

The association XorConstraint is over EntityType is derived (as indicated in ORM by an asterisk). By default, ORM predicates are read top-down or left-to-right. A "<<" symbol reverses this reading direction. ORM allows derivation rules to be specified graphically or textually. Here, the derivation rule is expressed graphically by the equality constraint (circled "=") between the derived association and the indicated projection on the XorConstraint and EntityType roles. The derivation rule verbalizes formally as: XorConstraint is over EntityType **if** XorConstraint spans **an** OptionalRole **that is a** Role **that** is played by EntityType. Normally all constraints on a derived association should themselves be derivable. However, in this case, the uniqueness constraint on the derived association is not derivable. This is shown by displaying the constraint in bold (and red for colored displays).

The uniqueness constraint on the derived association asserts: **each** XorConstraint is over **at most one** EntityType (i.e., the roles governed by the constraint must be played by the same entity type). Conceptually, extra constraints on derived associations are enforced by first materializing the association, and then applying the constraint as a base association. Although this practice is unusual in most applications, it is quite common in metamodeling applications to encounter rules that are most conveniently expressed in derived associations.

The use of non-derived uniqueness constraints on derived associations raises normalization concerns, since any uniqueness constraint is equivalent to a functional dependency (FD). For example, applying ORM's Rmap relational mapping algorithm (Halpin, 2001a) to the conceptual schema in Figure 2 leads to this relation scheme for Role:



{'0', '1'} {'1', 'n'}

Role (roleNr, entityTypeName, [xorConstraintNr]^{1,2}, minMultiplicity, maxMultiplicity, [predicate])

[1] exists only if minMultiplicity = 0

[2] frequency >= 2

Relation schemes for Attribute, Subtyping, and BinaryAssociation, and various inter-relation constraints are also created by Rmap, but are omitted here. The primary key is underlined, optional columns are enclosed in square brackets, attribute domain constraints are listed in braces, and the subtyping and frequency constraints are expressed by numbered qualifications.

The normalization issue in question is the functional dependency from the attribute xorConstraintNr to the attribute entityTypeName, depicted by the arrow shown. Since xorConstraintNr is optional, this FD means that each non-null value recorded for xorConstraintNr determines exactly one value for entityTypeName. Since this embedded, partial FD is not implied by the primary key constraint, this might be considered to violate normalization principles. We could enforce the partial FD constraint by materializing a relation scheme XorConstraint(xorConstraintNr, entityTypeName) and setting up a pair-equality constraint between this and the projection Role[xorConstraintNr, entityTypeName]. But in practice it is better to simply enforce the partial FD within the Role table itself. With today's relational DBMSs, this can be easily and efficiently done. For example, the following general form of the constraint can be simplified further for individual inserts or updates (e.g., within an insert/update trigger):

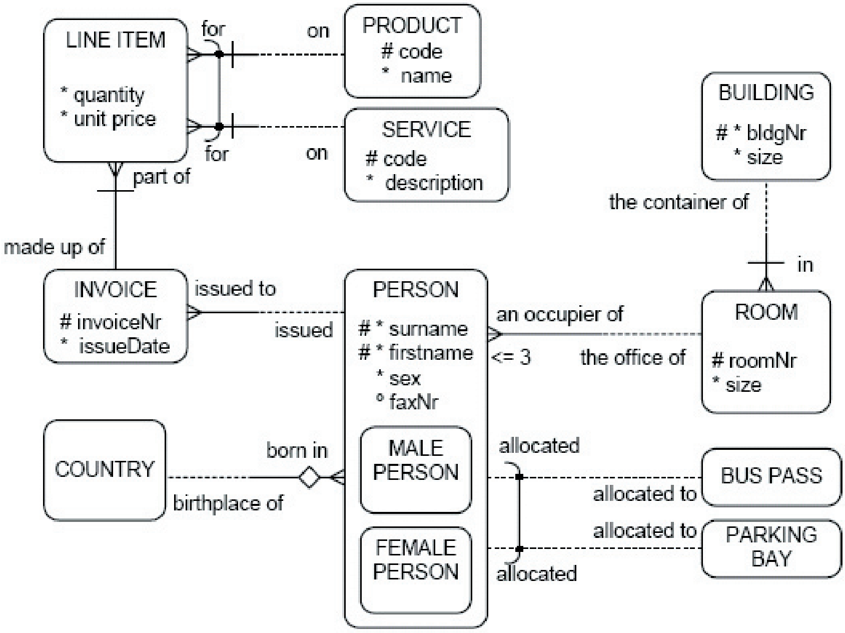
```
check( not exists
    (select xorConstraintNr from Role
    where xorConstraintNr is not null
    group by xorConstraintNr
    having count(distinct entityTypeName*) > 1 ) )
```

Since the FD here is partial (applies only to non-null values within an optional attribute), it is not covered by classical normalization theory. Moreover, no redundancy of base facts is involved, and the derived redundancy for the derived fact type XorConstraint is over EntityType is controlled by the above constraint, so no update anomaly can occur. Since such constraints can also be efficiently implemented, this licenses denormalization arising from non-derived uniqueness constraints on derived fact types.

BARKER ER

The term "Barker ER notation" denotes the notation for Entity Relationship modeling discussed in the classic treatment by Richard Barker (1990). Originating at CACI in the United Kingdom, the notation was later adopted by Oracle Corporation in its CASE design

Figure 3: A sample model in Barker ER notation



tools. Although Oracle now supports UML class diagrams as an alternative to the traditional ER notation, for database applications many modelers still prefer the Barker notation. A representative sample model is shown in Figure 3.

Entity types are shown as named, soft rectangles with their attributes listed inside. A “*” or “o” before an attribute indicates that it is mandatory or optional, respectively. A “#” before an attribute indicates that is the primary identifier for the entity type, or at least part of the primary identifier. All associations are binary, and are denoted by named lines. Forward and inverse predicate readings may be supplied at the line end from which they are to be read.

Each line-half depicts one of the two roles in the association. Like ORM, Barker ER separates the concepts of optionality and cardinality. If the line-half is solid, this usually means the role is mandatory. If the line-half is broken, this always means the role is optional. The cardinality of a role is assumed to be 1 unless a crow’s foot is used (indicating many). In Figure 3, for example, each invoice is issued to exactly one person, and each person is issued zero or more invoices.

The Barker notation uses an exclusive-arc to declare an exclusion constraint over a set of roles played by the same object type. If the roles have solid lines, the roles are also disjunctively mandatory. For example, in Figure 3 each line item is for a product or service but not both (xor), and each person is allocated at most one of the bus pass and parking bay options (possibly neither, so this is simple exclusion).

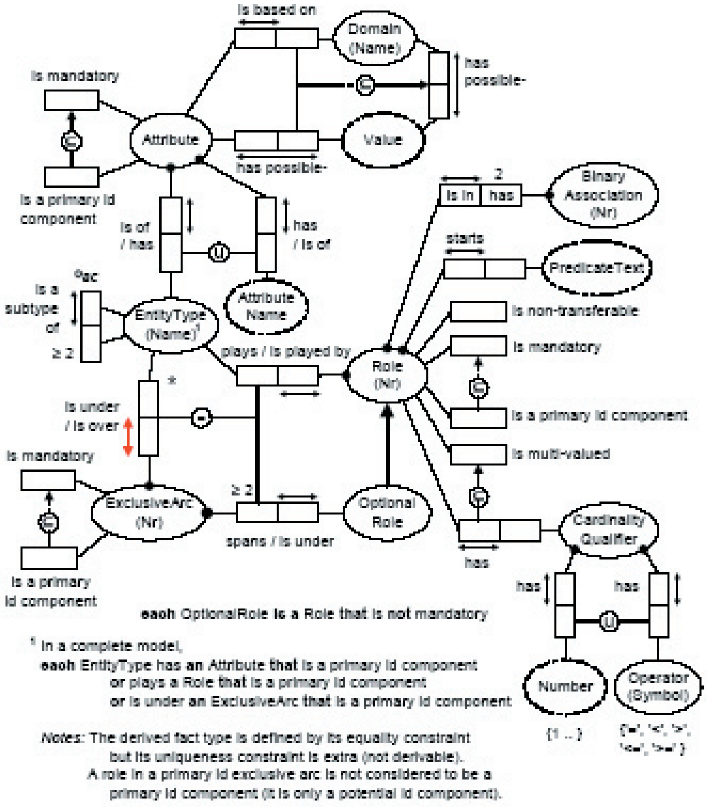
In a complete model, each entity type must have a primary identification scheme involving one or more attributes (marked #) or roles (marked by a stroke “|” across the role line). For example, in Figure 3 a building is identified by its building number, and a room is identified by combining its local room number with the fact that it is in a given building.

Barker ER also supports a restricted form of disjunctive reference, since the roles of a mandatory exclusive arc can be used in an identification scheme. In Figure 3, for example, a line item is identified by combining its invoice with either a product code or a service code.

Subtyping is depicted by Euler diagrams. Only single-inheritance is allowed (each subtype has only one supertype). Moreover, the union of the subtypes must equal the supertype, even if this requires an artificial subtype such as "Other" to ensure this. In Figure 3, MalePerson and FemalePerson form a partition of Person (i.e., they are mutually exclusive, and also they collectively exhaust their supertype).

A dynamic constraint denoted by a diamond marks a role as "non-transferable". For example, the diamond in Figure 3 indicates that once a person is recorded as being born in a given country, their birth-country cannot be changed to another country. Since we usually wish to allow editing of mistaken entries, this constraint has limited practical use except for tasks such as auditing.

Figure 4: An ORM metaschema for Barker ER


is based on
Domain (Name)
has possible-
is mandatory
Attribute
Value
has possible-
is a primary id component
is of / has
has / is of
Binary Association (Nr)
is in
has
2
starts
PredicateText
°ac
is a subtype of
≥ 2
EntityType (Name)¹
Attribute Name
is non-transferable
is mandatory
plays / is played by
Role (Nr)
*
is under / is over
is a primary id component
is multi-valued
is mandatory
ExclusiveArc (Nr)
≥ 2
Optional Role
spans / is under
Cardinality Qualifier
has
is a primary id component
has
has
each OptionalRole is a Role that is not mandatory
¹ In a complete model,
each EntityType has an Attribute that is a primary id component
or plays a Role that is a primary id component
or is under an ExclusiveArc that is a primary id component
Number
Operator (Symbol)
{1 ..}
{'=', '<', '>', '<=', '>='}
Notes: The derived fact type is defined by its equality constraint
but its uniqueness constraint is extra (not derivable).
A role in a primary id exclusive arc is not considered to be a
primary id component (it is only a potential id component).

An ORM metamodel for Barker ER is shown in Figure 4. Entity types are identified by name, and attributes are identified by combining their name with their entity type. Value constraints on domains and attributes are specified as shown. Here the subset constraint (circled "⊆") is applied between role-pairs, the first pair being projected from a join path. This constraint verbalizes thus: **if a** Domain hosts **an** Attribute that has **a** possible Value **then that** Domain has **that** possible Value. ORM allows set-comparison constraints (subset, equality or exclusion) to apply between sequences of compatible roles, where a role sequence can be projected from a join path (the act of moving through an object type performs an object join). For a detailed discussion of join constraints in ORM, see Halpin (2002).

Roles and associations are identified by numbers (whose display is normally suppressed). It is unclear whether the Barker notation allows both roles in the same association to have the same predicate text (e.g., for a symmetric association). If it doesn't allow this, we could also identify a role by combining its predicate text with its association (add the relevant external uniqueness constraint). Roles could also be identified by their position within a standard ordering of an association: this could be expressed as two binaries, as in Figure 2 (Role starts BinaryAssocation, Role ends BinaryAssociation) or by co-referencing Role has Position and Role is in BinaryAssociation.

Subtyping is specified by the acyclic association EntityType is a subtype of EntityType. The frequency constraint (≥ 2) ensures that each supertype has at least two subtypes. The uniqueness constraint makes the association many-to one, so it enforces single inheritance. This uniqueness constraint also implies that subtyping is intransitive, so an intransitivity constraint is omitted.

All unary predicates satisfy the closed world assumption. Optionality of attributes, roles and exclusive arcs is captured using the unary predicate "is mandatory". Non-transferability is modeled with the predicate "is non-transferable".

The basic cardinalities are captured by the unary predicate "is multi-valued": if true, the cardinality is many (displayed as a crow's foot); if false, the cardinality is 1. A cardinality of many may be qualified with a number and operator as shown. Further restrictions that apply to number-operator combinations are omitted here (e.g., "=" requires a number > 1).

Primary identifier components are specified using an "is a primary id component" predicate. The subset constraints between these unaries ensure that only mandatory elements can be components of a primary identifier. Barker ER allows both xor and simple exclusion constraints. An xor constraint is modeled as an ExclusiveArc that is mandatory—its roles, though optional, are depicted by solid lines. A simple exclusion constraint is an ExclusiveArc that is optional (not mandatory)—its roles are depicted by dashed lines. Applying is mandatory and is a primary id component directly to ExclusiveArc ensures that solid lines and identification strokes are distributed uniformly to the spanned roles (these properties cannot apply to just some of the spanned roles). The subtype definition and additional notes at the bottom of Figure 4 are self-explanatory.

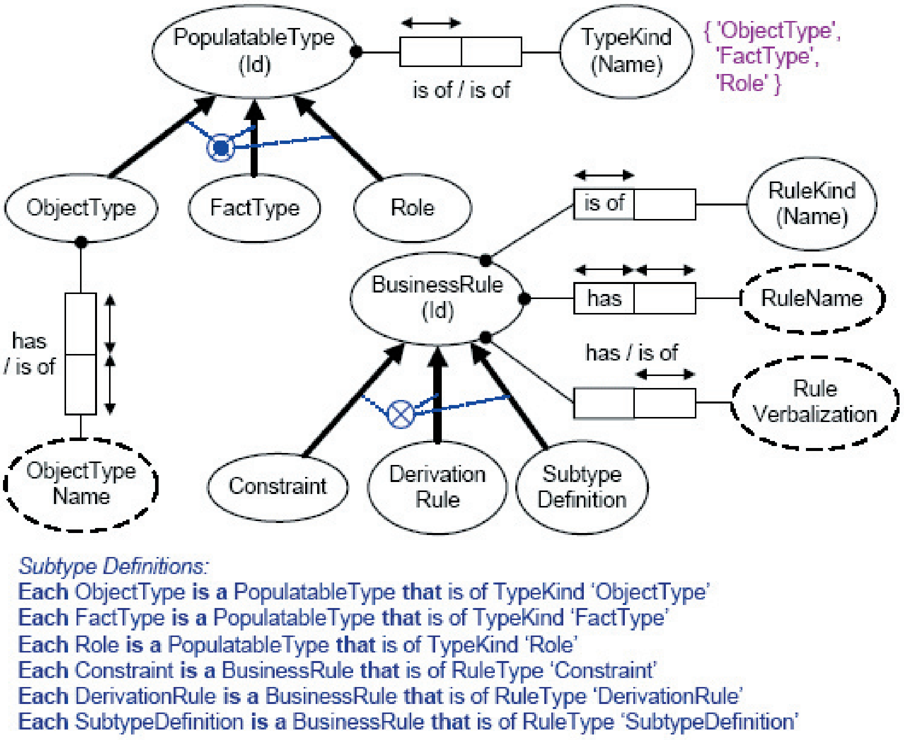
For readers who prefer the Barker ER notation to ORM, Barker (1990, p. H-2) presents a basic ER metamodel for Barker ER. Although mostly correct, it is substantially incomplete (e.g., it ignores various features, constraints and identification schemes), and also contains errors (e.g., it allows alternate keys but not overlapping keys, and it forbids any value from belonging to both a domain and an attribute population).

ORM

Figure 5 shows part of a simplified ORM metaschema for ORM. As for our metaschemas for IE and Barker ER, we have simplified naming schemes by assuming the metaschema can be instantiated by just one model at a time (ignoring reuse of model components across multiple models), and by ignoring namespaces within a model. The populatable types are object types (e.g., Person, Country), fact types (e.g., Person was born in Country), and roles (e.g., being born in a country). The main kinds of business rules are constraints (e.g., **each** Person was born in **at most one** Country), derivation rules (e.g., FactType.arity = **count each** Role **that** is in FactType), and subtype definitions (e.g., **each** MalePerson **is a** Person **who** is of Gender 'M'). In addition to having surrogate identifiers, object types have names, fact types have readings (verbalizations) derived by concatenating object type names with predicate readings (see later), roles may have names (see later), and business rules have names and verbalizations.

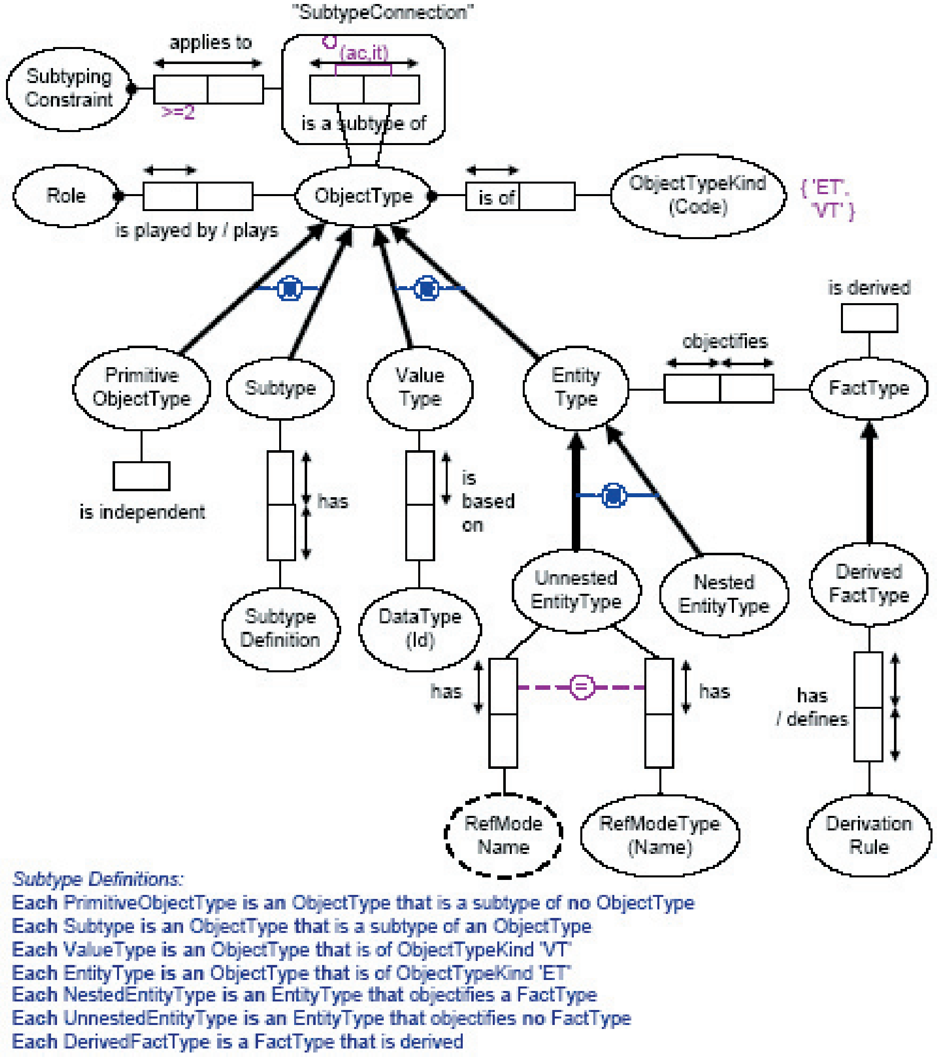
The subtype definitions below the diagram formally define each subtype in terms of roles played by their supertype. If subtypes collectively exhaust their supertype, this may be displayed as a circled dot. If subtypes are mutually exclusive, this may be displayed as a circled "X". These subtyping completeness and exclusion constraints may be overlaid to form a "lifebuoy" or partition symbol, as shown. In ORM, such subtyping constraints are derivable from formal subtype definitions and other constraints. Subtypes and derived fact types should be well defined by rules. To save space, some obvious subtype definitions may be omitted from now on.

Figure 5: ORM metaschema fragment for ORM populatable types and business rules



Subtype Definitions:
Each ObjectType **is a** PopulatableType **that is of** TypeKind 'ObjectType'
Each FactType **is a** PopulatableType **that is of** TypeKind 'FactType'
Each Role **is a** PopulatableType **that is of** TypeKind 'Role'
Each Constraint **is a** BusinessRule **that is of** RuleType 'Constraint'
Each DerivationRule **is a** BusinessRule **that is of** RuleType 'DerivationRule'
Each SubtypeDefinition **is a** BusinessRule **that is of** RuleType 'SubtypeDefinition'

Figure 6: Metaschema fragment for main ORM object types and fact types

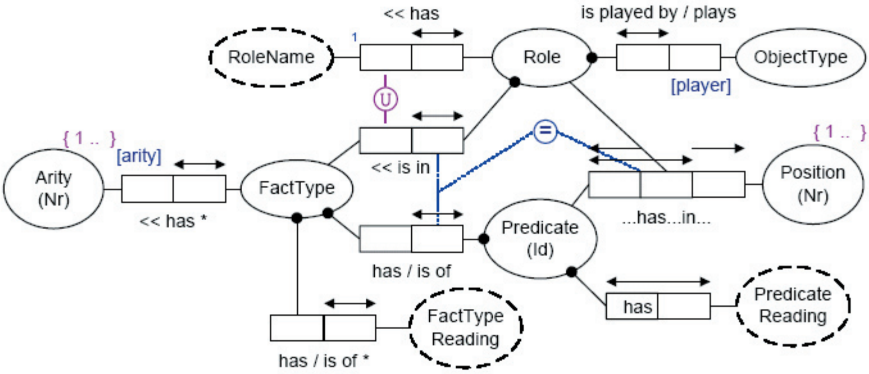


Subtype Definitions:
Each PrimitiveObjectType is an ObjectType that is a subtype of no ObjectType
Each Subtype is an ObjectType that is a subtype of an ObjectType
Each ValueType is an ObjectType that is of ObjectTypeKind 'VT'
Each EntityType is an ObjectType that is of ObjectTypeKind 'ET'
Each NestedEntityType is an EntityType that objectifies a FactType
Each UnnestedEntityType is an EntityType that objectifies no FactType
Each DerivedFactType is a FactType that is derived

Figure 6 shows another fragment of the ORM metschema. An object type either is primitive or is a subtype. A primitive object type is independent if its instances can exist independently of playing any role in a fact type. Subtyping in ORM allows multiple inheritance (e.g., AsianWoman may be a subtype of both AsianPerson and Woman, each of which is a subtype of Person). Subtypehood is acyclic and intransitive.

An object type is also either non-lexical (entity type) or lexical (value type). A nested entity type is an entity type constructed by objectifying a fact type. For example, the fact type ObjectType is a subtype of ObjectType has been objectified as SubtypeConnection. Treating the nested object type as distinct from its source fact type allows a single inheritance implementation of this part of the type hierarchy. Alternatively, a nested entity type

Figure 7: Naming of roles, predicates, and fact types in ORM



[1] For each object type, its far role names (over all its associations not declared symmetric) are distinct.

could be defined to be both an entity type and a fact type, or all fact types could be defined to be object types. If an unnested entity type has a simple reference scheme (e.g., Country has CountryCode), this may be abbreviated by displaying a reference mode (e.g., Code) in parentheses the object type name. Reference modes may be classified into different kinds to control the automatic expansion of a reference mode to its underlying relationship type.

Figure 7 shows one way to metamodel naming of fact types, predicates, and roles in ORM. To save space, derivation rules are omitted. The arity of a fact type is its number of roles, so it is derivable. ORM fact types may be of any arity: unary (e.g., Person smokes), binary (e.g., Person drives Car), ternary (e.g., Person imported Car from Country), and so on. As in logic, a predicate corresponds to an ordered set of roles covering a single fact type; hence each predicate provides one way to traverse the roles of a fact type. Fact types themselves are essentially unordered, but must have at least one predicate defined. Each predicate is identified by a surrogate identifier, not by a name, since different predicates may have the same name (e.g., "has"). Each role is identified by a surrogate identifier, and also has a unique position within its predicate. A role may also be given a name (e.g., "player"). Within a given fact type, the same role name may apply to at most one role (unless the fact type is declared symmetric; e.g., Person is sibling of Person). Globally the same role name may appear in different fact types. Roles in the same fact type are co-roles of one another. A role is a far role of an object type if and only if it has a co-role that is played by that object type. To ensure that role path specifications are unambiguous, we require that for any given object type, the names of its far roles must be distinct unless the fact type has been declared symmetric (textual rule 1).

Each fact type may be regarded as an unordered set of roles, with one or more ways to traverse its roles. For example, given the fact type comprised of the role set {r1, r2, r3}, we might traverse its roles in the order (r1, r2, r3) or the order (r1, r3, r2), etc. Since a traversal corresponds to a permutation (or ordered set) of the roles in the fact type, each fact type with n roles (n > 0) may have up to n! traversals declared by the user. Each such traversal has at least one reading (we allow multiple readings to cater for aliases). The join-equality constraint (circled "=") indicates that the sets of (role, predicate) pairs projected from the two arguments of the constraint must be equal. This ensures that a fact type traversal includes all and only those roles in the fact type.

A fact type reading may be derived by inserting the names of its object types into the placeholders in one of its predicate readings. The non-derived uniqueness constraint on FactTypeReading is of FactType ensures that a fact type reading provides a value-based way to identify a fact type. Let p1 = (r1, r2, r3) denote the predicate underlying the fact type read as Person has Rank in Sport when the roles are traversed in the order (r1, r2, r3). In this case, the predicate reading is "… has … in …". The alternative fact type reading Person in Sport has Rank uses the predicate reading "… in … has …" and traverses the roles in the order (r1, r3, r2).

For modeling, one reading is enough for each fact type. For conceptual queries that navigate via predicate readings, n readings are sufficient (one staring at each role). This metaschema goes well beyond what is sufficient, allowing users complete freedom to express fact types in as many ways as they wish. Although the metaschema for fact type readings appears complex, the user experience is simple and flexible, since users can specify any convenient reading depending on how they want to navigate through n-aries. If a less-user friendly approach is adopted, where only one reading per n-ary is allowed, the metaschema can of course be drastically simplified.

Figure 8: A basic metaschema for ORM constraints

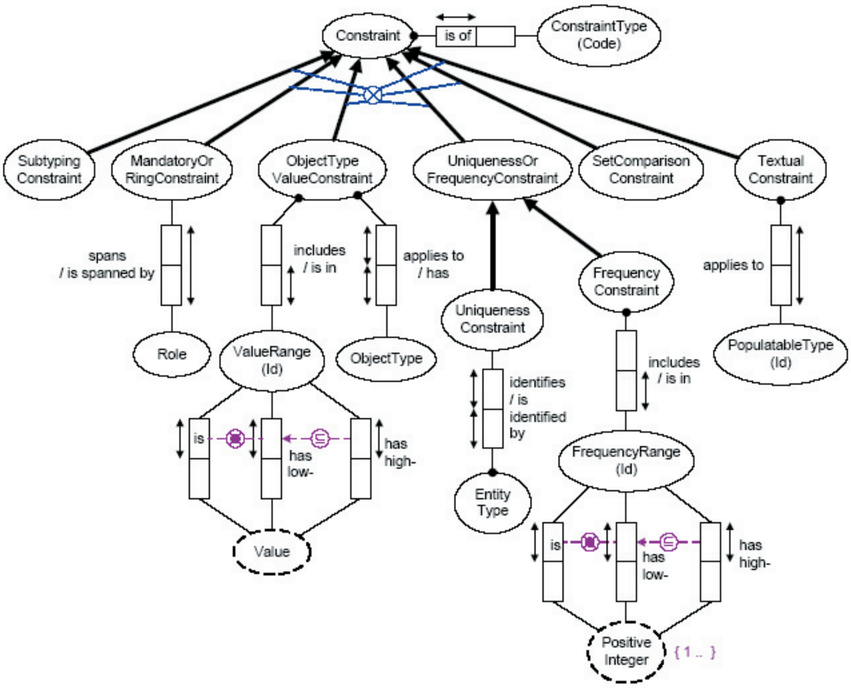


Figure 8 shows a basic metamodel for most of the ORM constraints. Space limitations prevent a treatment of all ORM constraints. For simplicity, subtype definitions as well as several textual constraints are omitted (e.g., each mandatory or ring constraint applies to compatible roles).

The fact type UniquenessConstraint identifies EntityType is required because ORM requires each entity type to have a value-based identification scheme for human communication. In a complete model, each entity type has an identification scheme based on a uniqueness constraint that spans a far role of one of its binary associations. If the uniqueness constraint is external, at least two associations are involved. For these reference associations, the entity type's near roles must be functional (simple uniqueness constraint) and disjunctively mandatory. Hence ORM supports disjunctive reference in its most general form, going well beyond the Barker ER notation in this regard. For a practical example based on botanical naming, see Halpin (2001a).

Industry experience indicates that the additional constraints captured graphically in ORM often occur in practical database application domains. ORM constraints are essentially role-based, and so is the ORM constraint notation. Since each role of a fact type corresponds to a column in a sample fact table, the constraints can easily be understood and validated using sample populations. Moreover, tool support enables the constraints to be automatically verbalized in a variety of natural languages, which also facilitates model validation with the domain expert.

Figure 9 provides further details of constraints that involve role projections. A role projection is a particular *occurrence* of a sequence of roles projected from a path through the conceptual schema, for use in specifying a specific business rule. For pragmatic reasons, metamodels often involve occurrences, rather than relying on extensional uniqueness for identification. For example, two different role projections involved in different constraints may in fact project over the same roles, but will have different surrogate identifiers because they are different occurrences. This allows us to modify one of the occurrences without impacting the other.

Figure 9: A more detailed look at constraints involving role projections

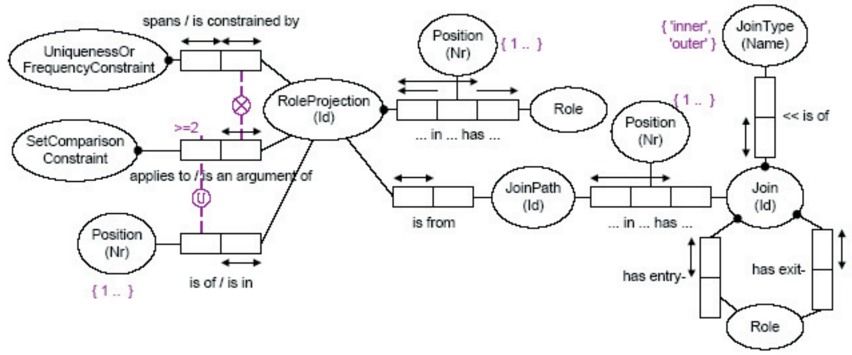
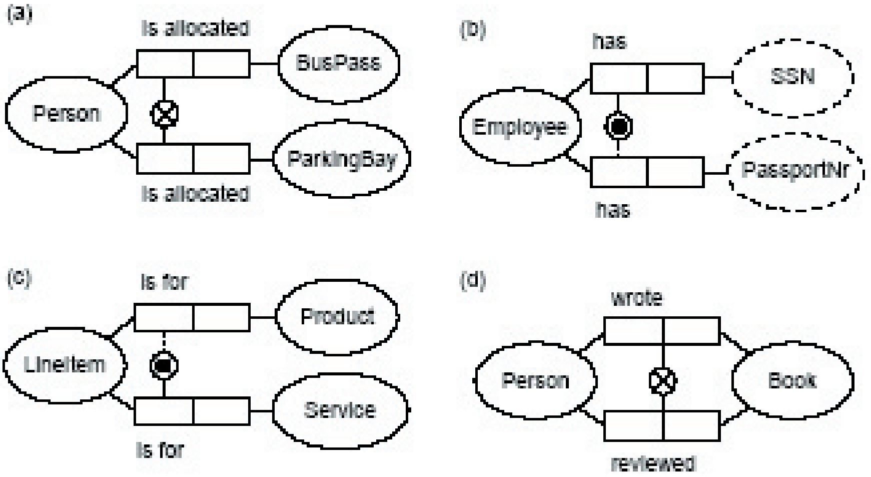


Figure 10: Constraint orthogonality



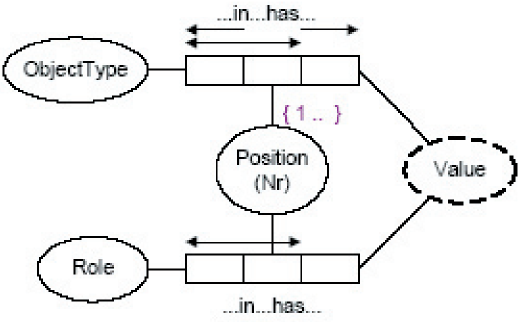
In the metamodel fragment of Figure 9, role occurrences are used for constraints that may potentially involve a conceptual join. A conceptual join occurs when passing through an object type while navigating a conceptual schema. The join requires that the object playing the entry role to the object type is the same object playing the exit role when leaving the object type, and may be an inner or outer join. Join constraints in ORM apply to role projections over paths that may involve one or more conceptual joins, and special care is needed to disambiguate the role paths, as discussed by Halpin (2002).

ORM constraints are orthogonal, both semantically and syntactically, making it easier to master the constraint language. For example, Figure 10(a) shows a simple exclusion constraint (circled "X"), indicating that nobody can be allocated both a bus pass and a parking bay (and they might be allocated neither). Figure 10(b) shows an inclusive-or constraint (circled dot), indicating that each employee must have a social security number or a passport number (or perhaps both). Figure 10(c) shows an exclusive-or constraint, (circled dot superimposed on X) obtained by orthogonally combining an inclusive-or constraint with a simple exclusion constraint, indicating that each line item must be for a product or a service but not both. In each of these three cases, the individual roles are clearly optional.

Contrast this with the Barker ER notation for exclusive arcs in Figure 3, where the optional roles in the xor constraint on line item appear mandatory (solid line) if taken individually. The Barker notation is not context-free, since a solid line for a role means different things in different contexts. This lack of orthogonality makes the notation harder to understand. A similar comment applies to IE. Moreover, neither Barker ER nor IE supports the concept of an inclusive-or constraint at all—in fact, their choice of notation for xor prevents an intuitive extension to their constraint language for this case.

As another example of orthogonality, ORM constraints may be applied wherever they make sense. For example, the notion of mutual exclusion applies between compatible roles, or between compatible role-pairs, etc. As a simple example, Figure 10(d) shows an exclusion constraint between Person-Book pairs (nobody who wrote a book may review that same book). Although the exclusive arc in Barker ER does enable a simple exclusion constraint

Figure 11: Modeling instances in ORM



between roles to be expressed, it does not allow exclusion between sequences of more than one role. Again, the arc notation itself is not intuitively extensible for this case.

Recall that a model is a schema (structure) plus a population (set of instances). Figure 11 adds a metafragment to expand an ORM metaschema to an ORM metamodel. Sample populations may be provided for object types (see top ternary) and roles (see bottom ternary). Fact types are populated by populating all their roles with the same number of values. The position indicates a row number of the instance table.

MULTIPLICITY IN ER, ORM AND UML

Earlier papers (Halpin & Bloesch, 1999; Halpin, 2001b) provided a detailed comparison between the data modeling constructs in ORM and UML, and indicated how the UML metamodel (OMG, 2001; 2003) could be extended to capture some of ORM's additional graphical constraints. Like IE and Barker ER, UML's graphic notation is far less expressive for data modeling than ORM. In addition, the notion of multiplicity or cardinality in UML and these two versions of ER is problematic when it comes to n-ary associations. As background to this claim, let's review the binary association case first. Figure 12 depicts a mandatory, n:1 association in all four notations. The UML multiplicity "*" is short for "0..*" (zero or many) and "1" abbreviates "1..1" (exactly 1).

UML and IE specify minimum and maximum multiplicities at the far end in which the association is read. Barker ER and ORM treat optionality (whether the role is mandatory or optional for each population instance of its object type) as a separate concept. Barker ER places maximum multiplicity on the far role, while ORM uses a uniqueness constraint (or more generally a frequency constraint) on the immediate role(s) to indicate the number of times an instance may occur, if it occurs there at all. ORM constraint notations are designed to assist validation by population, so the uniqueness constraint entails that entries in its fact column are unique, as in Figure 12(e).

Mandatory role constraints have global impact since they impact the object type, whose population may be spread over roles in many predicates. Uniqueness and frequency constraints have only local impact since they constrain the fact type population only. This separation of local and global concerns leads to greater orthogonality and expressibility once ternary or longer associations are used.

Given an n-ary association (n > 2), UML's multiplicity notation cannot express a mandatory role constraint on any association that has between 1 and n–2 mandatory roles,

Figure 12: A mandatory n:1 association in (a) UML, (b) IE, (c) Barker ER, and (d) ORM

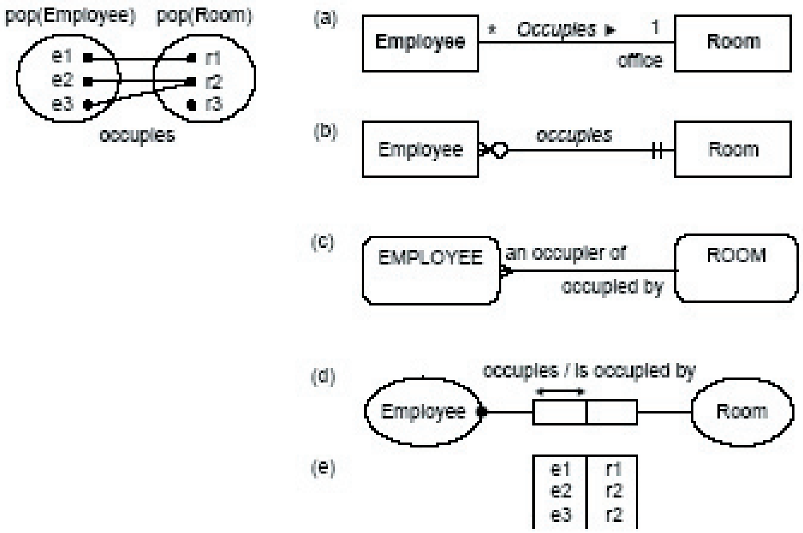
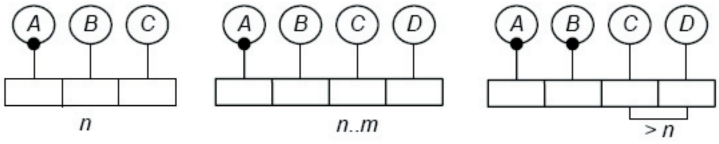
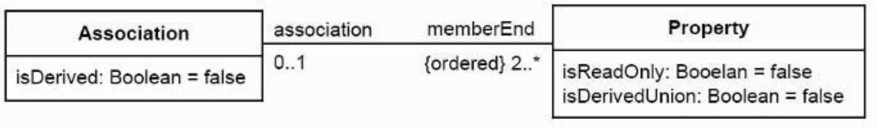


Figure 13: Some mandatory and frequency constraints with no graphic equivalent in IE, Barker ER or UML



nor can it capture a minimum occurrence frequency above 1 for any sequence of fewer than n–1 roles. This is because multiplicity on one role is defined in terms of the other n–1 roles. This is fine for binary associations, where n–1 = 1, but not for ternaries and beyond. For example, none of the mandatory role or frequency constraints expressed in Figure 13 can be graphically expressed in UML (or IE or Barker ER, for that matter). For practical examples of such constraint patterns, see Halpin (2001b). This weakness stems from placing multiplicities on a "far" end of an association rather than directly on the determining roles, and conflating global and local aspects in the same concept. Considerable care is required in choosing constraint primitives if the modeling notation is to scale properly to n-ary associations.

Figure 14: UML associations have two or more association roles

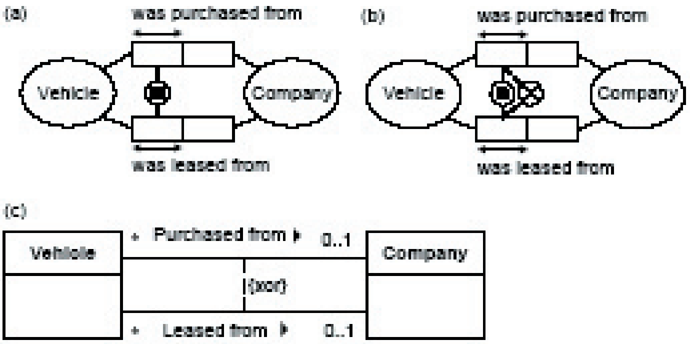

Association
isDerived: Boolean = false
association
0..1
memberEnd
{ordered} 2..*
Property
isReadOnly: Booelan = false
isDerivedUnion: Boolean = false

SOME OTHER ASPECTS OF THE UML METAMODEL

UML class diagrams include many object-oriented implementation features, such as attribute visibility and association navigability, and hence they surpass ER or ORM diagrams for detailed design of object oriented code (e.g., Java or C# programs). However, UML class diagrams currently lack standard notations for value-based identification schemes (e.g., uniqueness constraints on attributes, and external uniqueness constraints between association roles and/or attributes). This omission makes them less suitable for conceptual analysis, because business people do communicate using such identification schemes.

Figure 14 shows a fragment from the metamodel for the recently approved UML 2.0 (OMG, 2003). Roughly, a UML association corresponds to a fact type in ORM, and an association end (here called "memberEnd") corresponds to an ORM role. However, UML associations must have at least two roles, so unary fact types need to be modeled in other ways (e.g., using Booelan attributes or subtypes). This typically leads to formulations that are less natural than unary sentences. For example, the fact instance that may be expressed in ORM as "Person 'Sam Spade' smokes" would typically be rendered awkwardly in UML as "SamSpade: Person.isSmoker = true". So while UML surpasses IE or Barker ER in its ability to model n-ary associations directly, its lack of support for unaries still impedes natural communication. Hopefully this restriction will be removed in some future version of UML.

Figure 15: Exclusive-or constraints in ORM and UML



The ORM constraint notation is unambiguous and orthogonal. UML constraints don't always meet these criteria. As a simple example, consider the exclusive-or constraint in Figure 15(a). This verbalizes in ORM as: **each** Vehicle was purchased from **a** Company **or** leased from **a** Company **but not both**. In ORM, an exclusive-or constraint is an orthogonal combination of an inclusive-or constraint (circled dot) and an exclusion constraint (circled X), and may even be displayed as these two separate constraints if desired, as shown in Figure 15(b). Although UML lacks both an inclusive-or constraint and an exclusion constraint, it does include an exclusive-or constraint as a primitive constraint, using the notation "{xor}". Unfortunately, the UML metamodel defines the xor constraint to apply between associations, not association ends. This leads to ambiguity when two roles are played by the same class. For example, the xor constraint in Figure 15(c) is ambiguous, because formally we have no way of knowing whether it means the constraint verbalized earlier, or the constraint that each Company sold a Vehicle or leased a Vehicle but not both. Such constraints may be disambiguated by adding an OCL expression (Warmer & Kleppe, 1999), but clearly the metamodel should be altered to avoid such ambiguity in the first place.

There are several other aspects of the UML metamodel that need improving to make it more suitable for conceptual analysis. For example, associations should allow multiple readings, and association classes should be able to be named using noun phrases distinct from the verb phrases used for their underlying association. In spite of such problems, UML is clearly superior to both ER and ORM for the detailed design of object-oriented code. Each of the methods discussed in this chapter has its own strengths and weaknesses.

CONCLUSIONS

The ORM metamodels provided for IE, Barker ER and ORM clarify commonalities and differences between the different data modeling notations. The use of non-derived constraints on derived fact types can be convenient, especially for expressing complex rules, so long as the usage is controlled. Exclusive-or constraint notations in IE and Barker ER are unorthogonal. The UML, IE and Barker ER approaches use multiplicity notations that do not scale properly for n-ary associations.

ORM was designed for orthogonality and expressibility from the ground-up. The most debatable aspect of ORM is that it avoids attributes in its base conceptual models, though attribute-based models can be automatically derived from ORM models when desired (Campbell et al., 1996). Combined with its richer constraint language, this tends to make ORM diagrams larger than corresponding models in the other notations. This disadvantage is a price many modelers are willing to pay to see the extra detail and domain connectedness. One advantage of ORM's role-based approach is that its small set of metaconcepts and syntactical elements can specify a wide range of rules in a uniform way. In contrast, attribute-based approaches often lose expressibility if fact types are modeled as attributes instead of associations. For example, xor constraints in IE, Barker ER or UML can be applied only between association roles, not between attributes or between roles and attributes. There is no reason in principle for this restriction, but pragmatically to remove such restrictions would add considerable complexity, requiring additional notations and metarules. The same comment applies to the many additional kinds of constraint supported in ORM. This suggests that the only efficient way to achieve such expressibility without complexity is to adopt an attribute-free approach, as in ORM.

On the other hand, attribute-based approaches lead to more compact diagrams. By using ORM for the initial conceptual analysis and validation with the domain expert, and then transforming the ORM model to an attribute-based model such as an ER model or UML class diagram, modelers can reap the benefits of both attribute-free and attribute-based approaches.

REFERENCES

Barker, R. (1990). *CASE*Method: Entity relationship modelling.* Wokingham: Addison-Wesley.

Bloesch, A., & Halpin, T. (1997). Conceptual queries using ConQuer-II. In D. Embley & R. Goldstein (Eds.), *Proceedings of the 16th International Conference on Conceptual Modeling ER'97* (pp. 113-126). Berlin: Springer.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language user guide*. Reading: Addison-Wesley.

Campbell, L.J, Halpin, T.A., & Proper, H.A. (1996). Conceptual schemas with abstractions: Making flat conceptual schemas more comprehensible. *Data and Knowledge Engineering*, *20*(1), 39-85.

Chen, P.P. (1976). The entity-relationship model—towards a unified view of data. *ACM Transactions on Database Systems*, 1(1), 9-36.

De Troyer, O., & Meersman, R. (1995). A logic framework for a semantics of object oriented data modeling. In M. Papazoglou (Ed.), *Proc. OOER'95: Object-oriented and entity-relationship modeling*, (pp. 238-49). Berlin: Springer-Verlag.

Halpin, T., Evans, K., Hallock, P., & MacLean, B. (2003). *Database modeling with Microsoft Visio for enterprise architects*. San Francisco: Morgan Kaufmann.

Halpin, T. A. (1998a). Object Role Modeling: an overview. [Online]. Available: http://www.orm.net/overview.html

Halpin, T. A. (1998b). ORM/NIAM Object-Role Modeling. In P. Bernus, K. Mertins & G. Schmidt (Eds.), *Handbook on architectures of information systems*, (pp. 81-101). Berlin: Springer-Verlag.

Halpin, T. A. (2001a). *Information modeling and relational databases*. San Francisco: Morgan Kaufmann.

Halpin, T. A. (2001b). Supplementing UML with concepts from ORM. In K. Siau & T. Halpin (Eds.), *Unified Modeling Language: Systems analysis, design, and development issues*, (pp. 167-184). Hershey, PA: Idea Group Publishing.

Halpin, T.A. (2002). Join constraints. In T. Halpin, J. Krogstie & K. Siau (Eds.), *Proc. Seventh CAiSE/IFIP-WG8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, Toronto, Canada (pp. 121-131).

Halpin, T. A., & Bloesch, A. C. (1999). Data modeling in UML and ORM: a comparison. *Journal of Database Management*, 10(4), 4-13.

ter Hofstede, A.H.M. (1993). *Information modeling in data intensive domains*. PhD thesis, University of Nijmegen.

ter Hofstede, A.H.M., Proper, H.A., & Weide, th.P. van der. (1993). Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18(7), 489-523.

IEEE (1999). *IEEE standard for conceptual modeling language syntax and semantics for IDEF1X$_{97}$ (IDEFobject)*. IEEE Std 1320.2-1998. New York: IEEE.

Finkelstein, C. (1998). Information engineering methodology. In P. Bernus, K. Mertins & G. Schmidt (Eds.), *Handbook on Architectures of Information Systems*, (pp. 405-427). Berlin: Springer-Verlag.

Martin, J. (1993). *Principles of object oriented analysis and design*. Englewood Cliffs: Prentice Hall.

NIST. (1993). *Integration definition for information modeling (IDEF1X)*. FIPS Publication 184, National Institute of Standards and Technology. [Online]. Available: http://www.sdct.itl.nist.gov/~ftp/idef1x.trf

OMG. (2001). *OMG Unified Modeling Language Specification*, version 1.4. [Online]. Available: http://www.omg.org/uml

OMG (2003). *UML 2.0 Infrastructure*. [Online]. Available: http://www.omg.org/uml

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language reference manual*. Reading, MA: Addison-Wesley.

Siau, K., & Halpin, T.A. (eds.). *Unified Modeling Language: Systems analysis, design, and development issues*. Hershey, PA: Idea Group Publishing.

Warmer, J., & Kleppe, A. (1999). *The object constraint language: Precise modeling with UML*. Reading, MA: Addison-Wesley.

Chapter III

An Evaluation Framework for Component-Based and Service-Oriented System Development Methodologies

Zoran Stojanovic, Delft University of Technology, The Netherlands

Ajantha Dahanayake, Delft University of Technology, The Netherlands

Henk Sol, Delft University of Technology, The Netherlands

ABSTRACT

Components-Based Development (CBD) and Web Services (WS) nowadays are prominent paradigms for implementing and deploying advanced distributed information systems. They have been proposed as the ways to support effective business/IT alignment and produce high quality and flexible software solutions that fulfill business goals within short time-to-market. However, current achievements in these areas at the level of methodology are much behind the technology ones. CBD methods proposed so far lack a comprehensive support for component and service concepts throughout the development process. By treating components as packages of implementation artifacts during software deployment or as larger-grained business objects during analysis and design, these methods are not well equipped for modeling loosely coupled coarse-grained components that offer business meaningful services organized in a Service-Oriented Architecture (SOA). This chapter presents an evaluation framework that highlights the extent to which a particular method is component-based and *service-oriented. The CBD method sample is selected and evaluated using the framework's concepts and requirements. Based on the evaluation, the method improvements are proposed in order to provide consistent, systematic, and integrated CBD and WS methodology support throughout the lifecycle.*

INTRODUCTION

Modern enterprises are in the flux of rapid and often unpredictable changes in both business and Information Technology (IT). New business demands caused by the enterprise's need to be competitive on the market require an immediate support of the advanced IT solutions. At the same time, new IT opportunities and achievements are constantly emerging and must be rapidly adopted to provide new and more effective ways of conducting business. Therefore, today more than ever it is important to provide an effective business/IT alignment in order to produce high quality and flexible software solutions within short time-to-market, that as close as possible support business goals and match business needs.

During the last years, new development paradigms and models have been proposed to support these aims. First Component-Based Development (CBD) (Brown & Wallnau, 1998), and then Web Services (WS) and Service-Oriented Architecture (SOA) (IBM, 2003; W3C, 2003) have been introduced as the ways to build complex enterprise systems and provide effective enterprise application integration. The CBD platforms and technologies, such as CORBA Components, Sun's Enterprise Java Beans (EJB), and Microsoft's COM+/.NET are now *de facto* standards in web-based systems development. On the other hand, the growing interest in Web Services has resulted in a number of industry standards and initiatives (XML, WSDL, UDDI, SOAP, etc.) (W3C, 2003). What they have in common is that CBD and WS have both been first introduced through new technology standards and infrastructures, and after that corresponding methods, tools and modeling techniques have been proposed. While the technology is a necessary element of any solution, it is not sufficient on its own. Methods, techniques and tools for developing component-oriented applications based on business requirements are equally important (Welke, 1994). Such development methods need to incorporate the concepts of component and service as an integral part of the whole system life cycle, from business to implementation.

While there is an established development methodology practice in the case of CBD, in the field of WS and SOA, current achievements in this respect are much behind the technology ones. The former question of how to make use of object-oriented methods and techniques in practicing CBD is now largely replaced by whether and in what ways CBD methods can be used in developing WS applications. Therefore, of great importance is proposing an approach for architecting the system that consists of collaborating components and services. Such an approach should specify the way of capturing and organizing business requirements within the platform-independent logical system architecture that closely maps business concepts and goals. The approach should further provide mapping of the architecture to the particular technology settings that ensures bi-directional traceability between business concepts and implementation artifacts. This is the main idea behind the current Object Management Group's (OMG) Model Driven Architecture (MDA) (OMG, 2003).

Current object-oriented and component-based development methods do not provide a necessary support for designing and developing component-based and service-oriented business applications. Methods that have evolved from pure object-oriented backgrounds have difficulties in recognizing the fundamental nature of components, considering the componentization aspects at the level of code packaging. By treating components as implementation artifacts during deployment and as larger-grained business objects during analysis and design, these methods are not well equipped for modeling loosely coupled coarse-grained components that offer business meaningful services organized in the service-oriented architecture.

Therefore, the main objective of this chapter is to identify the current methodological shortcomings of the CBD methods and to present a first cut of a methodology framework for designing improved and proper CBD and SOA methods. For this reason, the chapter is organized as follows: first, an account of the current state of the CBD methods and approaches is given by describing and comparing the most prominent and well documented CBD-methods. Based on this analysis, a framework for defining necessary characteristics and requirements for an advanced CBD/WS methodology is defined, and the chosen method sample is evaluated accordingly. Finally, suggestions are made regarding the ways of improving the methodology towards comprehensive component-based and service-oriented systems development support.

THE CURRENT STATE OF CBD METHODS

CBD and WS are evolutionary rather than revolutionary approaches. CBD has evolved from “divide-and-conquer” modularization ideas and concepts in systems development (Gartner Group, 1997; Szyperski, 1998). During the last few years, due to the rapid development of Internet technology and commercial applications, the CBD paradigm has been seen as the main strategic imperative for time-to-market quality solutions (Gartner Group, 1997; Butler Group, 1998). Higher productivity, flexibility, and quality, through reusability, replaceability, efficient maintainability, scalability and parallel work are among the claims and benefits made for CBD (Butler Group, 1998; Allen & Frost, 1998).

From a technical perspective Web Services are essentially extended and enhanced component interface constructs. Using standards for service interoperability, such as XML and SOAP, Web Services can provide location independent business or technical service that can be published, located and invoked across the Web regardless of underlying technology (IBM, 2003). Besides technology, there is a need to architect service-oriented computing systems. Service-Oriented Architecture (SOA) is an approach to distributed computing that considers software resources as services available on the network. A basis of SOA is the concept of a service as a functional representation of a real world business activity meaningful to the end user and encapsulated in a software solution. Using the analogy between the concept of service and business process, SOA provides that loosely coupled service components are orchestrated into business processes that support business goals. Similar initiatives were already proposed in the past, such as CORBA or Microsoft’s DCOM. What is new about SOA is that it relies upon universally accepted standards like XML and SOAP to provide broad interoperability among different vendors’ solutions. And what is more important, the level of abstraction is further raised, so that the main building blocks of SOA are now real world business activities encapsulating in the services that offer business value to the user. Component-based and Web Services technology infrastructures are the ways of implementing the SOA.

CBD Methods and Approaches

The CBD paradigm is seen by many as a further step above the OO paradigm, resulting in many similar concepts, principles and ideas. The similarities of objects and components have become the focus of many discussions and studies (Szyperski, 1998). This has caused the present introduction of course-grained objects as components in object-oriented methods and techniques. On the other hand, the natural first candidate for WS and SOA methodology practice is using CBD methods and techniques. The question is whether current CBD methods provide necessary concepts and mechanisms to support that. Components have been for a long time treated mainly as binary packages of code influenced by the versions 1.x of the standard Unified Modeling Language (UML) (Booch, Rumbaugh & Jacobson, 1999). This suggests handling components at the implementation and deployment phases of a development lifecycle, while still following classical object-oriented modeling, analysis and design. During the last few years, advanced CBD approaches have been proposed that provide more sophisticated support to component concepts and mechanisms. However, the identification and specification of components are still done mainly in an entity-driven fashion, by closely matching the underlying business entities such as Customer, Product, and Order. In this way, components are treated more in the form of business objects than business services. For the purpose of developing modern business-driven service-oriented systems, it is necessary to define coarser-grained business components that potentially encapsulate several business objects and provide real world business services of a measurable and perceivable value to the user. After the original implementation definition of components, a more logical view on components has been introduced in the UML standard 1.4 and the latest version, 1.5. The major revision of the UML (version 2.0), which is scheduled for this year, promises further improvements in representing components as both design-level and implementation-level artifacts.

A sample of well-published and widely used CBD methods has been chosen for analysis and evaluation of the state-of-the-art of CBD methodology practice. These methods are documented in books, on web sites, and in companion papers, in parallel with opportunities for training and consultancy. They have been already used in practical projects and are supported by software development tools. The methods show a clear structure and guidelines for the development lifecycle through a sequence of process steps. The following methods will be presented and analyzed:

- Rational Unified Process (Jacobson, Booch & Rumbaugh, 1999);
- Select Perspective method (Allen & Frost, 1998; Apperly et al., 2003);
- Catalysis approach (D'Souza & Wills, 1999);
- KobrA approach (Atkinson et al., 2002);
- UML Components (Cheesman & Daniels, 2000);
- Business Component Factory (Herzum & Sims, 2000).

Rational Unified Process

Rational Unified Process (RUP) (Jacobson et al., 1999) is a software engineering process developed by Rational Software (now part of IBM). RUP is the direct successor to the Rational Objectory Process (version 4), which resulted from the integration of the Rational Approach and the Objectory process (Jacobson, Christerson, Jonsson & Overgaard, 1992) in 1995. RUP was the first process to use UML from its origin (version 0.8). RUP includes development and management of the process and covers the entire software life cycle. The RUP is very well documented in books and companion papers, and is supported by a web-based knowledge base that provides all team members with guidelines, templates and tool mentors for all development activities. Many training and consultancy opportunities are available. A family of tools, produced by IBM Rational Company, is available to support the process.

The key concept of RUP is the definition of activities (workflow) throughout the development life cycle, such as requirement elicitation, analysis, design, implementation, and testing. Unlike the classical waterfall process, these activities can be overlapped and performed in parallel. Within each of the activities, there are well-defined stages of inception, elaboration, construction, and transition. While they occur in sequence, there may be iterations between them until a project is complete. During the design of the solution, the CBD support is encouraged, but it is rather declarative and implicit. RUP promotes CBD through the use of UML and it is heavily influenced by UML notations and its design approach. UML takes more of an implementation and deployment perspective on components through component and deployment diagrams. Therefore, RUP's view on the component concept is still at the level of physical packaging. This is illustrated by RUP's definition of a component as "a non-trivial, nearly independent, and replaceable part of a system that fulfils a clear function in the context of a well-defined architecture. A component conforms to, and provides the physical realization of a set of interfaces." RUP suggests the use of UML subsystems for modeling components without detailed explanation. It is obvious that RUP is not specifically focused on component-based development. It offers a general framework for object-oriented design and construction that can be used as the basis for other methods. Using the UML as the basic modeling notation provides a great deal of flexibility in system design, but specific support for key component modeling concepts is lacking and limited to the UML notation. In the light of current improvements of the UML towards the new version 2.0, RUP may adapt more complete and consistent CBD mechanisms and principles. One of the main advantages of RUP is that it provides an opportunity for iterative and incremental system development, which is seen as the best development practice.

Select Perspective

The Select Perspective method (Allen & Frost, 1998; Apperly et al., 2003) was created by combining Object Modeling Technique (OMT) (Rumbaugh, Blaha, Premerlani, Eddy & Lorenson, 1991) and Use Case driven Objectory method (Jacobson et al., 1992). After the standardization of UML as an object-oriented modeling language, the method adopted the UML notation. The first version of Select Perspective comprised the activities of business modeling, use case modeling, class modeling, object interaction modeling and state modeling. With the growing interest in CBD, Select Perspective was extended with activities related to different aspects of components—business-oriented component modeling, component modeling of legacy assets, and deployment modeling (Allen & Frost, 1998). The latest version of Select Perspective published recently (Apperly, 2003) provides more comprehensive and sophisticated support for component-based and service-oriented development. The method is well documented in the available books, companion papers, and technical reports. Training and consultancy support are available. A family of component-based tools includes Component Factory, Component Architect, Component Manager, code generations, etc. that effectively support the various aspects of the method.

The Select Perspective uses the standard UML enriched with the extensions to support component modeling. For the purpose of business modeling, it uses the notations of the Computer Science Corporations (CSC) Catalyst methodology (CSC, 1995). This notation and technique help to link the business processes, associated use cases and classes. The method also uses Entity-Relationship Diagrams (ERD) for mapping between the UML class model and the relational data model. Components in Select Perspective are executables offering services through published interfaces. The services can, but not necessarily, be implemented using object technologies. Though based on the UML, the method uses a streamlined set of UML modeling techniques, without introducing new concepts that require UML extensions.

The component concept is seen as the concept of package, defined in UML as "a general purpose mechanism for organizing elements into groups" (Booch et al., 1999). Two basic stereotypes of the package are distinguished: a service package used in business-oriented component modeling and a component package used in component and system implementation. A service package contains classes that have a high level of interdependency, and serve a common purpose by delivering a consistent set of services. A component package represents an executable component, i.e., the actual code. When a service package is placed on a node of the network, it effectively becomes a component package. Special attention is paid to component modeling of the legacy assets, i.e.,on how to use the component principles to efficiently wrap and further integrate legacy systems.

The latest version of Select Perspective includes support for Web Services, as well as for Model-Driven Architecture and Agile Software Development, as promising paradigms in software development. The Select Perspective software development life cycle is a set of workflows that are based on an iterative and incremental development approach. The method defines three basic workflows: Consume, Supply, and Manage. Consume workflow delivers the solution that uses components and services from the component suppliers, then maintains and supports that solution. Supply workflow delivers and maintains components based on the request for services from particular component and service consumers. Manage workflow is concerned with the activities of acquiring, certifying, classifying and locating components to serve the needs of both component consumers and suppliers. Select Perspective provides a comprehensive development lifecycle for component-based solutions that supports business-aligned parallel development in order to reduce time-to-market. The method defines project management features such as iterative working, incremental working and planning, parallel working and monitoring. The method is derived from best practices proven on real projects.

Catalysis

Catalysis (D'Souza & Wills, 1999) is a component-oriented approach with its origins in object-oriented analysis and design. Catalysis began in 1991 as a formalization of OMT (Rumbaugh et al., 1991), and was developed over several years of applying, consulting, and training. It extends second-generation OO-methods such as Fusion (Coleman et al., 1993) and Syntropy (Cook & Daniels, 1994), including support for framework-based development and defining methodical refinements from abstract specification to implementation. Catalysis is well documented in the corresponding book, technical papers, and by a dedicated website (www.catalysis.org). Opportunities for training and consultancy are also provided. Catalysis is effectively supported by the COOL family of tools such as COOL:Gen, COOL:Spex, COOL:Joe, etc., originally developed by Sterling Software. After acquiring Select Software by Computer Associates the tools have been renamed into Advantage Gen, Advantage Joe, etc. (Computer Associates, 2003).

Catalysis is a methodology for modeling and constructing open systems from objects, components and frameworks. Catalysis is mostly a development process, which means that its main purpose is to provide software construction from high-level requirements. Unlike RUP, Catalysis does not cover project management, process measures, tests, and team-task management. Although the details of Catalysis are somewhat complex, the approach is based on a small number of underlying concepts such as types, conformance, collaborations and frameworks, used throughout the approach. Catalysis component development approach encourages a strong separation of component specification from implementation, using an extended form of UML. This allows technology-neutral specifications to be developed and then refined into implementation in a number of different implementation technologies. Although Catalysis covers the complete system lifecycle, "from business to code", the component concept is visible at the implementation level. It defines a component as a "coherent package of software artifacts that can be independently developed and delivered, as well as be composed and extended to build something larger". Higher-level support for the component concept is provided by the concept type, as a stereotype of a class. The type is defined as a representation of some consistent behavior in the domain, while a class is an implementation of the type. External behavior of the type is defined by its interface, which is mapped to class operations. Refinements from abstract to more detailed descriptions of a system are recorded by capturing conformance between types. The interactions among types are modeled as collaborations. This captures a set of actions involving multiple, typed objects playing defined roles with respect to each other. A package is a larger-grained development concept, and acts as the basic unit of a development product that can be separately created, maintained, delivered, updated, assigned to a team, and generally managed as a unit. The Catalysis approach is not a rigorous methodology. It is rather a semi-structured set of design principles, advises and patterns throughout the system development life cycle. Therefore, a systematic "roadmap" of the Catalysis way is lacking. The whole method tends to be vague, with possible difficulties for applying it in practice. However, Catalysis represents an excellent foundation for supporting various CBD concepts, principles and techniques.

KobrA

KobrA is a software development method that uses and supports the component paradigm in all phases of the software life cycle, following the product-line strategy (Atkinson et al., 2002). It has developed as a result of the KobrA project from 1999 to 2001, funded by the German government, and led by Softlab GmbH, Psipenta GmbH, GMD-FIRST and Fraunhofer IESE. The KobrA approach is influenced by other leading software development methods, such as Fusion (Coleman et al., 1993) and Catalysis (D'Souza & Wills, 1999). It is also compatible with the Rational Unified Process (Jacobson et al., 1999) and OPEN (Graham, Henderson-Sellers & Younessi, 1997) process frameworks. The method is documented in the dedicated book (Atkinson et al., 2002), scientific papers, and companion reports. The method is well equipped to support practical software engineering projects and supported by Softlab's specially developed workbench based on the Enabler repository family. This workbench allows organizations utilizing the KobrA method to assemble their own preferred suite of tools to support KobrA development.

Although product-line engineering is fully integrated within KobrA, it is not necessary to develop a product-line when applying KobrA. KobrA extends the usual “binary” view of components by providing a higher-level representation based on a suite of tightly related UML diagrams. The method defines a component by two main parts: specification, which describes the externally visible characteristics of a component, and realization, which describes how a component satisfies the specification in terms of interactions with lower-level sub-components. The central artifact in KobrA is the framework, which represents a collection of KobrA components organized in a tree-structure based on the composition hierarchy. In KobrA, every behavior-rich element of a system is a *Komponent*, i.e., a KobrA component. The method uses qualifiers to distinguish between different kinds of components: instance vs. type and specification vs. realization. KobrA supports the principles of architecture-centric and incremental development. KobrA also includes systematic, rigorous quality assurance techniques, namely inspections, testing and quality modeling. A KobrA component has at the same time properties of a class and a package. On the other hand the role of component interface is not emphasized enough. The composition of components is defined mainly through containment trees, instead of collaboration between component interfaces. The KobrA approach does not offer strict rules about how to identify components. The approach rather treats important business domain concepts as components and follows OO analysis and design on them. The authors of the method are researchers describing a theoretical approach that has not been extensively proven in practice. The authors propose a notation that is not standard UML, but rather a custom notation loosely based on UML. The KobrA method is a broad mix of software engineering guidance for CBD. Much of this guidance is theoretical, without supporting tools or reports of commercial experience. The method is based on a number of software engineering principles (parsimony, encapsulation, and locality) that are often restatements of generally accepted principles for keeping things simple, separating concerns, and minimizing coupling.

UML Components

Cheesman and Daniels (2000) propose a method called UML Components that is strongly influenced by Catalysis, RUP, and Syntropy (Cook & Daniels, 1994). The method focuses on the specification of components using the UML. While the method is published in the book form, there is no information of its application in practice. The method describes how to architect and specify enterprise-scale, component-based systems using the UML. The method gives a detailed explanation of the basic principles of software components and component-based development, in a manner that establishes a precise set of foundational definitions that are essential in practicing the method. The method discusses how core UML diagrams such as the Use Case can be used in the context of components. Although, the method defines the six main workflows (similar to RUP) as Requirements, Specification, Provisioning, Assembly, Test, and Deployment, it primarily focuses on the first two. The specification workflow is the most interesting one from the perspective of CBD, and consists of three main sub-workflows: component identification, component interaction, and component specification. For the purpose of component-based design, the method uses the UML notation enriched with proper extensions, stereotypes and modeling conventions.

The method stops with the activity of Component Specification. It does not offer the ways to translate the component specification into implementation and verify that the implementation complies with the specification. The method proposes a number of extensions that are outside the bounds of standard UML, which makes it difficult to apply one of the existing UML-based modeling tools in practicing the method. The method provides precise and detailed guidance on how to extend and customize the UML for the purpose of component modeling and specification. In essence, UML Components offer a subset of Catalysis concepts together with a much simpler RUP-like process. Components are identified through identifying system interfaces that are related to use cases, and business interfaces that are related to business entity types. Identified components are further specified through the information types that represent the component state, as well as pre-conditions and post-conditions on component operations that specify the component behavior.

The method lacks some of the key ideas of Catalysis, including the nesting of components to arbitrary depths, the recursive application of development concepts, and the use of frameworks to package larger-grained reusable structures. The UML components approach does not take into account potential different levels of component granularity and importance of using the separation of concerns in defining them. Despite some limitations, the UML Components method contains important, practical advices for developers practicing CBD.

Business Component Factory

Herzum and Sims (2000) propose the Business Component Factory (BCF) approach as a way to use components in enterprise system design. Both authors have been active in the OMG's business object development efforts. The approach is split into three parts: i) conceptual framework that covers CBD and component concepts, ii) component factory set-up for putting the factory itself in place and iii) manufacturing component-based software through modeling and design considerations. The authors suggest a classification of components that reflects granularity: language class, distributed component, business component, business component system, and finally, federation of system-level components. The method provides little coverage of commercial implementation platforms such as J2EE, CORBA or COM+/.NET. The focus of the method is on the business components that are defined as important business concepts that are relatively autonomous in the problem space. Business components can be of the following types: entity, process, utility, and auxiliary. These components are more related to business object theory, which is logical since the authors' background is in business objects. By separating entities and behavior, this approach does not provide a uniform view on components. On the other hand the role and importance of service-based interfaces are diminished to some extent.

The method further presents the set-up of the component factory from the viewpoint of the development process, technical architecture, and application architectures, as well as the project management architecture. Finally, the method proposes a number of modeling and design steps and activities by focusing on the functional architecture. There is a lack of precise modeling of the dependency and relationship between components, the importance of which is stressed by the method, but not covered in detail. The approach does not use the standard UML notation, which makes it difficult to relate it to the current UML-based development practice. There is no information about practical experience in using the method. No particular tool support has been proposed. The method is based on practical experience of the authors; it is written by practitioners for practitioners. The central element of the approach is the concept of business component. For the purpose of service-oriented computing, the focus should be moved above that to coarser-grained business components that are by the method called system-level components. Although authors briefly define a standard development process similar to RUP, Business Component Factory represents a general, comprehensive CBD approach rather than a prescriptive method.

Summary

At the time of conducting this research the authors were not aware of other remarkable and complimentary efforts in the area of CBD methods, and they do not claim that this selected list is complete. This is, however, a reasonably representative subset on which to conduct the research. The chosen CBD methods are compared based on the list of evaluation criteria summarized in Table 1. The other CBD approaches and best practices, mainly coming from the industry as proprietary company practices, are not considered for further evaluation because of the following reasons. Although most of these approaches are well supported by appropriate tool-sets and used in some practical projects, they lack structure and tend to be incomprehensible in their presentation of the development process. They combine best of breed OO and CBD concepts, elements and strategies in an ad-hoc man-

Table 1: Variety of CBD support provided by the methods

	Rational Unified Process (RUP)	**Select Perspective**	**Catalysis**	**KobrA**	**UML Components**	**Business Component Factory**
Availability	Book, website, consultancy, training	Book, website, consultancy, training	Book, website, consultancy, training	Book, papers	Book, papers, consultancy	Book, papers, consultancy
Background	Industry	Industry	Academic & Industry	Academic & Industry	Theoretical & Practical	Theoretical & Practical
Type of methodology	Development + management	Development + management	Development	Development + management	Development	Development
Usage of methodology	Regularly used in industry	Regularly used in industry	Catalysis-based methods used	Used by KobrA consortium	Potentially used in industry	Potentially used in industry
Process form	Workflows, guidelines, templates	Phases, guidelines	Rough guidelines, patterns	Phases, activities, guidelines	Workflows, activities	Phases, guidelines, patterns
Tool support	Rational product family (Rational Rose, etc.)	Select Component Factory (Select Architect, Component Manager)	COOL tools (COOL: Spex, COOL:Gen, etc.), now Advantage tool family	Enabler Workbench and Repository	No specific tool; UML-based tools can be used	No specific tool; UML-based tools can be used
Modeling techniques	UML	BPM Catalyst, UML, ERD	UML	UML-based	UML (with extensions)	UML-based
View on components	Logical + physical	Logical + physical	Logical + physical	Logical + physical	Logical + physical	Logical + physical

Table 1: Variety of CBD support provided by the methods (continued)

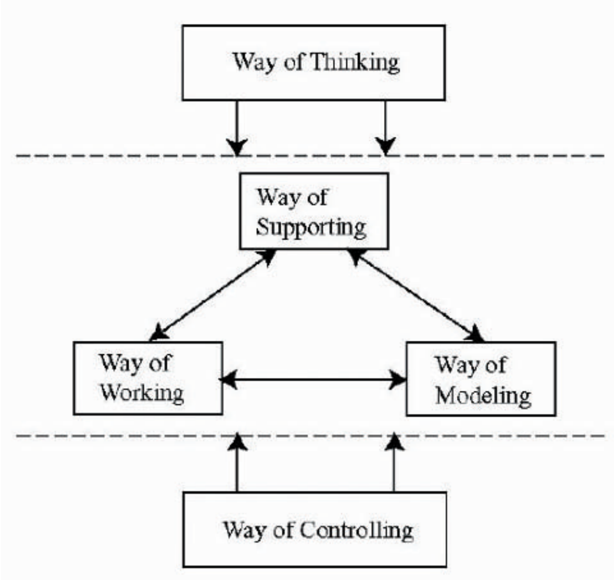
Component representation	UML subsystem	Service package, UML subsystem	Stereotype type	Stereotype of the UML class	Stereotype of the UML class	Not specific
Component implementation	UML Component and Deployment diagram	Component package, Deployment diagram	Package, Software components	Realization component	Not specific	Software components
Defined design patterns	No	Yes	Yes	No	No	Yes
Component repository	No	Yes	No	Yes	No	No
Reusability	Software components	Components, patterns	Components, patterns, frameworks	Design-level and software components	Design-level and software components	Design-level and software components, patterns
Incremental & Iterative	Yes	Yes	Yes	Yes	Yes	Yes

ner, often combining and using concepts from the selected methods. Their support for the component concepts varies greatly in nature and extent. Therefore, these approaches were not selected as representatives for the purpose of CBD method evaluation.

CBD METHODOLOGY EVALUATION FRAMEWORK

Most visible high level categorization of the evaluation approaches for information systems engineering methods fall into two main categories, namely the evaluation of the whole methodology and the evaluation of the modeling approaches within a methodology. A complete and a comprehensive review of the latter can be found in Siau and Rossi (1998). In this study, a methodology as a whole entity is the subject of evaluation. Many such evaluation approaches exist (see e.g., Hong et al., 1993; Blank & Krijger, 1983). All these approaches present a framework with interest areas of concerns and recipes for conducting the evaluation process. In this research, the evaluation of the whole methodology is considered in terms of making a judgement as to whether a methodology truly supports CBD and SOA. Kumar and Welke's (1992) evaluation of shortcomings of methodologies introduces the concept of methodology engineering and argues that the contingency factors given within the analytical framework of Sol (1988) provide a promising way to identify the content of a methodology. This fairly standard framework for method evaluation proposed by Sol (1988) was chosen because of its generic character, which makes it suitable for adapting to CBD and SOA issues. The experiences of many methodology evaluation researchers

Figure 1: A framework for understanding information system development



such as Hofstede and Verhoef (1996), Dahanayake (1997) and of researchers within CBD methodology evaluation community such as Boertien et al. (2001), and Stojanovic et al. (2001) also contributed to this decision. For issues relevant to the methodology engineering perspective, Dahanayake (1997), Kumar and Welke (1992), Rossi (1998), Tolvanen (1998), and Hofstede and Verhoef (1996) were used to complement the generic framework and to identify the appropriate requirements for CBD Methodology engineering.

Framework Foundation

Sol's analytical framework pays explicit attention to all important aspects of a process, and defines a set of contingency factors that characterizes the information systems development process: a way of thinking, way of modeling, way of working, way of controlling and way of supporting (Figure 1).

- *Way of thinking:* visualizes the essential philosophy of an information system development method regarding the information system's functionality and its role in the environment.
- *Way of modeling:* a way to structure problems by distinguishing between types of models required for problem specification and solution finding.
- *Way of controlling:* includes a set of directives and guidelines for managing the information systems development process, management of time, means, and quality aspects.
- *Way of working:* is seen as a way to structure problems by distinguishing between types of tasks to be performed for systems development process.
- *Way of supporting:* represents the tools that are used to support information systems development process.

To achieve the most from CBD, the nature and structure of the whole development process have to be aligned with the five contingency factors of the analytical framework. This in turn means that entirely new development methods and tools that are aligned with CBD and SOA principles are required. Therefore, the framework given above was used to identify the required method and characteristics for each of the contingency perspectives to provide a consistent and comprehensive CBD methodology. The requirements were determined by studying the CBD literature and visionary papers such as Welke (1994), Gartner Group (1997), Butler Group (1998) and from methodology engineering approaches such as Kumar and Welke (1992), Dahanayake (1997), Rossi (1998), and Tolvanen (1998). We then looked at the trends in CBD methodologies and finally categorized the requirements according to the contingencies of the analytical framework. We then presented the first cut of the evaluation framework with its requirements to a small number of practitioners involved in CBD application development and incorporated their feedback to provide an improved framework. The number of experts and the level of expertise at the time of the study was limited due to the pace of adoption of CBD technology into the local (the Netherlands) systems and (Dutch) software industry. The analytical framework and its contingency factors leading to the CBD methodology requirements evaluation framework is as follows:

Way of Thinking

The underlying philosophy of a CBD method should focus on:

- *Components and services as the main focus of a development process.*
  - The concepts of component and service should be the focus and the main elements of a method consistently used throughout the system development lifecycle.
- *Clear, consistent, and technology-independent component and service concepts.*
  - By defining a component as an encapsulated concept with specific roles and behavior in the domain, and with hidden interior and exposed services through interfaces, it can be easily understood by both business and IT worlds. The component concept should enable business domain experts to model business processes and requirements at a higher level, in a domain-specific, but implementation-independent way. On the other hand, application developers retain control over how these component models are turned into complete applications using advanced component-based technology.
- *Semi-formal and/or formal definition of component and service concepts.*
  - The semantics of the basic component and service concepts should be clearly and precisely defined using the semi-formal way (by defining metamodels using e.g., the UML and MOF) and/or formal notation (by using Object Constraint Language (OCL) grammar, mathematical set-theory expressions, or other formal specification techniques).
- *Enriched contract-based interface construct.*
  - The interface of a service-based component must be extended beyond simple operations' signatures to represent a real business contract between the provider and consumer of the service. Complete and precise, implementation-independent service specification including configuration and quality-of-service parameters provides effective mechanisms for service discovery and usage.

- *Focus on behavior-driven rather than data-driven service and component concepts.*
  - Services and components should not correspond to a single business object such as Customer or Order; they should manage information across the set of objects in providing required business value-added functionality.
- *Defining different scope and granularity levels of components and services.*
  - It is essential to define different scopes and granularity levels of services fulfilling different roles in business/technical system architecture through recursive composition and choreography. This means that each service can be realized through lower-level services and at the same time is a part of a higher-level service.

Way of Modeling

The underlying way of modeling of a CBD method should focus on:

- *Appropriate modeling notation for component and service concepts.*
  - The method should provide proper textual and/or graphical notations for component and service representation (human-understandable, machine-readable, or graphical notation) that is uniquely understandable by all actors in the development process.
- *Defining models at different levels of abstraction.*
  - Techniques and mechanisms for defining component-based and service-oriented computational independent models (CIM), platform independent models (PIM) and platform specific models (PSM) of the system being developed are necessary elements for achieving truly model-driven system development using components and services (OMG, 2003).
- *Modeling from various viewpoints using the concepts of component and service.*
  - System should be modeled from different viewpoints in order to reflect different concerns in the development process, such as enterprise, information, computational, engineering and technology (ODP, 1996; Stojanovic, Dahanayake & Sol, 2000). The concepts of component and service should be integrating factors across the viewpoints.
- *Focus on collaboration, interaction and coordination of components and services.*
  - Components and services should support particular steps of a business process and should be chained and coordinated in a way to create a business process flow. The modeling focus should be on representing service interaction, nesting, coordination and mutual dependencies, rather than on component internal realization.
- *Rigorous component specification.*
  - A precise, formal or semi-formal notation should be available to describe component specification. This should be sufficient for a rigorous analysis of the specification against a user's needs. Precise component specification should be precise and complete in order to provide easy, straightforward and effective component implementation. It should also represent the main information support for browsing a COTS catalogue or a Web Service registry.
- *Reusability of modeling artifacts.*
  - Single component or the whole patterns of component interactions can be explicitly modeled, stored in the repository and subsequently reused in further system designs. Thus, the models beside software code can represent valuable reusable artifacts.

Way of Working

The underlying way of working of a CBD method should focus on:

- *Full component/service lifecycle.*
  - The full component lifecycle should be provided including the activities of business component modeling, component architecture design and specification, acquisition of components, component discovering and identification, modification, binding, wrapping, assembling, testing, execution and maintenance.
- *Traceable component and service concepts.*
  - Make a component concept traceable and consistent throughout the system development life cycle, i.e., each phase in the component life cycle should transfer concepts to the corresponding development process phase.
- *Business-driven identification of components and services.*
  - Services and components must be identified and defined in a business-driven way as larger-grained, loosely coupled system units that can communicate synchronously, as well as asynchronously. They should correspond to real business activities and add a measurable business value to their consumers. In this way, business requirements and needs are seamlessly mapped to first-cut component-based, service-oriented system architecture.
- *Integration of different views and viewpoints.*
  - The method should provide techniques for integrating multiple views and perspectives on the component, e.g., specification vs. implementation components, business vs. technical components, and entity vs. process components in the context of different phases in the development process.
- *Providing model transformations and code generation.*
  - The method should provide effective ways for transformation of Computational Independent Model (CIM) into Platform Independent Model (PIM) and further into Platform Specific Model (PSM) according to the chosen technology platform, as well as software code generation for that platform from PSM (or directly from PIM) (OMG, 2003). The defined models should be kept in synchronization with the generated code, according to the principles of round-trip engineering.
- *Iterative and incremental development practice.*
  - CBD naturally supports iterative and incremental development, by breaking the complex problem down into smaller parts, and defining possible phases, increments and opportunities for parallel work inside a development process. Furthermore, a rigorous, repeatable refinement process through all presented component-based lifecycle phases should be provided to arrive at a software solution that meets original business requirements using an easily traceable pathway.

Way of Controlling

The underlying way of controlling of a CBD method should focus on:

- *Support for the measurement of non-functional process parameters.*
  - The method should define quantitative and qualitative measures based on non-functional parameters and associating proper control points in the lifecycle phases. CBD targets a market-driven application assembly model, where non-functional issues,

such as quality, flexibility, security, scalability, availability, etc. are associated with the methodology to measure the success of the development approach.

- *Proper process management approach.*
  - Process management within the CBD process should define control points in each process phase truly relating to a component concept. The management process has a broader meaning, and in parallel with the software development process, it schedules work, provides guidance for a team's activities, plans deliveries of artifacts, allocates resources, monitors and measures a project's progress, and directs the tasks of individual developers and the team as a whole.

Way of Supporting

The underlying way of support of a CBD method should focus on:

- *Effective tool support.*
  - CBD development method must be well supported throughout the system life cycle. A family of tools must be provided and systematically integrated covering particular aspects or parts of the development process. Tool support must be sufficiently flexible and tailorable to adapt to eventual changes in the particular method, and even to provide integrated support for the whole spectrum of CBD-dedicated development methods.

EVALUATION OF CBD METHODS

The CBD methodology requirements presented above were set against the documented materials of the selected methods discussed above and assessed as to the extent of their support for truly component-based and service-oriented system development. Further, in some instances the freely available demonstration and companion tools were used to identify the availability of CBD requirements. We first checked to see if the listed requirements were available, and if so, the extent to which they were available was evaluated by assigning a number from 1 to 5. The results of this assessment were sent to an expert panel familiar with CBD methods and based on their comments we adjusted our evaluation to arrive at a final value for each CBD requirement.

The Evaluation Process

The evaluation of CBD methods was carried out in association with an expert panel set up by a review group of experts in systems development. These experts were selected from large and medium-size systems development organizations. Alongside with the experts from the industry a group of academic researchers were chosen from the Masters level students involved in studying CBD methods. The industry experts belonged to functions such as project managers, systems architects, application designers, and delivery mangers. They were chosen for their organization's experience in OO application development and their trend in initiating component-based and service-oriented architecture design. The selected experts were from systems development branches of international banks, insurance companies, and large software houses. They were selected on the basis of being able to identify with one or more of the following:

1. They use a CBD approach.
2. They are in favor of using CBD and knowledgeable in CBD methods.
3. They are training their staff on CBD.
4. Their knowledge of CBD is based on a certain CBD method.
5. They have their own CBD approach developed on the basis of their OO methodology.

The first cut of the evaluation framework was presented to the industry expert panel and refined based on their comments. The resulting framework has been used to evaluate the selected six CBD methods.

The academic group was given the assignment to study two CBD methods from the six selected methods, and based on their knowledge to assign a number between one and five for each requirement in the evaluation framework. The academic group of 42 students was divided into six groups of seven members, while the industry expert panel consisted of 16 members. The evaluation was first conducted by the authors of this paper and compared with the evaluation of the academic group ratings. The majority of the ratings of the authors and the academic groups were the same but there were some differences in few cases, with maximum of two points of difference. In such cases the average was taken. Finally the evaluation framework was presented to the industry expert panel and asked for their evaluation. They were asked to evaluate only the methods they are familiar with. Their evaluations were similar in most cases with some exceptions—as the difference was not greater than two points the average was taken. A summary of the evaluation of CBD methods RUP, Select Perspective, Catalysis, KobrA, UML Components and Business Component Factory based on the requirements evaluation framework proposed above is presented in Table 2.

Findings

Summarizing our findings, we see that the idea of CBD and SOA is not yet fully integrated in the investigated methods. Components and services do not yet become the real focus of the methods. The concepts of component and service are not properly and clearly defined and specified yet. Components are often at the level of packaging of software code, or old-fashioned business objects. However, during the last years there have been positive signs in this direction together with the emerging of the new version of the UML 2.0 that treats components at both a logical and implementation level. Semantics and characteristics of components and services are mainly defined informally using prose text. More recent CBD methods propose an extended version of the interface concept beyond simple signatures of operations. They specify pre-conditions and post-conditions on operations, as well as information type model of the interface, but they still lack, among other things, the coordination aspects of operations, configuration mechanisms, and non-functional parameters. The importance of defining different scope and granularity levels of components, as well as their recursive composition, has been truly recognized only in Business Component Factory approach.

Regarding the way of modeling, the investigated methods are based on the current version of the UML, and based on that define proper extensions to represent necessary component and service concepts they utilize. Modeling from different viewpoints is an important mechanism in Business Component Factory. Rigorous component specification is to some extent provided in Catalysis and UML Components, while model reusability is

Table 2: Evaluation framework and the evaluation results of the CBD methods

Way of thinking	RUP	Select Perspective	Catalysis	KobrA	UML Components	Business Component Factory
Components and services as the main focus of a development process	2	4	3	4	4	4
Clear, consistent, and technology-independent component and service concepts	1	4	3	3	4	4
Semi-formal and/or formal definition of component and service concepts	2	3	3	4	3	3
Enriched contract-based interface construct	1	4	3	3	4	3
Focus on behavior-driven rather than data-driven service and component concepts	2	3	2	2	3	3
Defining different scope and granularity levels of components and services	1	3	2	2	2	4

Way of modeling	RUP	Select Perspective	Catalysis	KobrA	UML Components	Business Component Factory
Appropriate modeling notation for component and service concepts	3	3	3	2	4	2
Defining models at different levels of abstraction	3	3	3	3	3	4
Modeling from various viewpoints using the concepts of component and service	2	2	2	2	2	4
Focus on collaboration, interaction and coordination of components and services	1	3	3	2	2	3
Rigorous component specification	2	4	4	2	4	3
Reusability of modeling artifacts	2	4	4	3	2	2

Table 2: Evaluation framework and the evaluation results of the CBD methods (continued)

Way of working	**RUP**	**Select Perspective**	**Catalysis**	**KobrA**	**UML Components**	**Business Component Factory**
Full component/ service lifecycle	1	3	2	3	4	4
Traceable component and service concepts	2	3	3	3	4	4
Business-driven identification of components and services	1	3	2	3	4	4
Integration of different views and viewpoints	2	4	4	2	2	4
Providing model-transformations and code generation	3	3	2	2	3	3
Iterative and incremental development practice	4	4	4	4	4	4

Way of controlling	**RUP**	**Select Perspective**	**Catalysis**	**KobrA**	**UML Components**	**Business Component Factory**
Support for the measurement of non-functional process parameters	2	3	2	4	2	3
Proper process management approach	3	4	2	2	2	3

Way of supporting	**RUP**	**Select Perspective**	**Catalysis**	**KobrA**	**UML Components**	**Business Component Factory**
Effective tool support	4	5	4	4	2	2

***Explanation of the marks**: **1**-no match, **2**-poor match, **3**-matching to some extent, **4**-good match, and **5**-full match*

an important aspect of Select Perspective. The collaboration and choreography between components and services are not well specified in the methods. Although the collaboration concept is a first-class citizen in Catalysis, that is still at a lower level of abstraction than needed for the purpose of defining truly service-orientated architecture.

Regarding the way of working, the recent methods provide more a complete component and service life cycle, as well as traceable component concepts from business to technology.

Business-driven and behavior-driven identification of components and services are not yet fully supported by the methods. Modeling from different viewpoints becomes an important mechanism in managing system complexity by separating the concerns. The techniques for the transformations of models that are at different levels of abstraction and their further mapping to software code are not yet fully supported by the methods. All of the methods provide an iterative and incremental development practice that becomes *de facto* standard in software system development.

The way of controlling of the investigated methods should be further improved in the spirit of CBD and WS. The ways of measurement of non-functional process parameters and a proper process management approach must be defined in an improved way. Although most of the investigated methods are accompanied with effective tools to support them, a more sophisticated and comprehensive suite of tools is needed to support the variety of aspects of component-based and service-oriented development process, defined well by the evaluation framework's ways of thinking, working, modeling and controlling.

CBD Method Improvements

Current CBD methods and approaches do not define the concepts of component and service in a precise and implementation-independent way. Instead of making components the focal point of the complete development process in order to gain the huge benefits of the component way of thinking, the methods handle components at the implementation and deployment phases, or just as another form of old-fashioned business objects. Methods that have evolved from pure object-oriented backgrounds inherit difficulties in recognizing the fundamental nature of components, considering the componentization as a way of code packaging. A more formal and systematic approach to component-based and service-oriented development is needed covering the whole system life cycle with the component concepts and principles integrated into each of the phases. Integration between the phases, such as business, information, application and technology issues must be provided. This can be done using general well-grounded component theory as the means to bridge the different perspectives and viewpoints. A common CBD "language" used throughout the life cycle for the integration of different principles, concepts and perspectives, and a smooth transition among them must be ensured. In the framework for effective CBD methodology support presented above, we have proposed guidelines towards a systematic and integrated approach to component-based development. It has the potential to provide comprehensive, theoretical and practical methodological support for the CBD and WS paradigms. The framework can be seen as a first step in arriving at truly component-based and service-oriented systems development methodology engineering.

By following the requirements defined in the framework through the ways of thinking, modeling, working, controlling and supporting, we can create a method that can be fully applied in the new SOA and WS world. The most important elements of the next-generation CBD method are:

- Standard definition of component and service concepts in consistent, contract-based, and implementation-independent way.
- Specification of different component scope and granularity levels that are mutually related by composition and collaboration relationships and used throughout the system design and development.

- Appropriate standard modeling notation for representing component and service concepts—Different but isomorphic notation types can be proposed, such as textual (human understandable), graphical (e.g., UML) or machine-readable (e.g., XML-based grammar), that can serve the needs of different actors in the development process.
- The mechanisms for defining the system models at different levels of abstraction (from business to code) as well as the rules for performing transformations between them.
- Traceability of the component and service concepts from business requirements to software code and at the same time the integration of different viewpoints on the system being developed using these concepts.
- Model transformations, code generation and model reusability through an iterative and incremental development practice.
- A proper process management approach focused on components and their collaborations as the main artifacts of the development process, together with measuring quality parameters of the process.
- High-quality, effective component-based and service-oriented tools that should provide the necessary support for all the elements of the CBD/WS method.

CONCLUSIONS

A framework for effective CBD methodology support is introduced as a first step towards arriving at truly component-oriented systems development methodology. The framework is based on the five aspects of the system development process, namely the way of thinking, modeling, working, controlling and supporting, each of them capturing truly component-oriented requirements. A methodology sample was evaluated using the concepts and requirements of the evaluation framework. The framework can be used in practice to evaluate the true nature of available CBD methodologies and to elicit requirements. It can also be used to guide the development process to focus on CBD and SOA principles and concepts consistently throughout the development phases. At the same time, the confusion between the OO and the CBD way of systems engineering can be eliminated by referring to the framework.

Current CBD methods and approaches, such as Rational Unified Process, Select Perspective, Catalysis and so forth, do not include full support for the component and, specially, service concept. They propose handling components mainly at the implementation and deployment phase, instead of throughout the complete system life cycle. The methods are significantly influenced by their OO origins, while trying to introduce the CBD concepts using standard UML concepts and notation. The research presented here is an early call for researchers to re-think the fundamentals behind the whole research area of CBD. CBD is not another way of using old methodology structures for getting OO software technology to produce functionality. It is a paradigm shift and an opportunity to tighten the loose ends left dangling from the OO era. CBD should be considered from a market-based production platform that will bring the whole demand-supply chain in line with future developments, able to deliver time-to-market units of functionality. This opens a number of new opportunities for researchers as well as for practitioners. These new opportunities will include taking the challenge to determine what are truly CBD and SOA methodologies, techniques, and tools, and how to develop further CBD-based market models. Researchers undertaking these tasks have to consider the unlimited process power, and the "anywhere-anytime" information retrieval capacity that has become available with the use of plug-and-play functional components.

The research in CBD methods evaluations conducted so far has its limitations. One of the main limitations of this research is that it covers a few well-documented and established CBD methodologies. These methods have been evaluated using an average expert panel and the authors' knowledge and experience gathered through available literature. There were no interviews conducted to access the experiences of those who have used these methodologies in practice. As the CBD and WS fields are quite new, experienced practitioners are rare. The framework was based on a limited number of expert opinions and feedback was generated via questionnaires to find the appropriateness of the evaluation criteria. The approach can be improved by analyzing the evaluation criteria via interviews with expert users of the methods. Evaluation of the methods and the methodology framework used for this purpose led us to propose possible improvements in the CBD and WS methodology practice.

REFERENCES

Allen, P., & Frost, S. (1998). *Component-Based Development for enterprise systems: Applying the select perspective*. Cambridge University Press.

Apperly, H. et al. (2003). *Service- and Component-Based Development: Using the select perspective and UML*. Addison-Wesley.

Atkinson, C., Bayer, J., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wust, J., & Zettel, J. (2001). *Component-based product line engineering with UML*. Addison-Wesley.

Blank, J., & Krijger, M.J. (1983). *Evaluation of methods and techniques for the analysis, design and implementation of information systems.* Academic Service.

Boertien, N., Steen, M.W.A., & Jonkers, H. (2001). Evaluation of Component-Based Development methods. *Proceedings of the 6th CAISE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, Interlaken, Switzerland.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language user guide*. Addisson-Wesley.

Brown, A.W., & Wallnau, K.C. (1998). The current state of component-based software engineering. *IEEE Software*, September/October.

Butler Group. (1998). *Component-Based Development*. Management Guide, Researched by D. Sprott and L. Wilkes. Available: http://www.butlergroup.com (September, 2001).

Cheesman, J., & Daniels, J. (2000). *UML Components: A simple process for specifying Component-Based Software.* Addison-Wesley.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., Gilchrist, H., Hayes, F., & Jeremaes, P. (1993). *Object-oriented development: The fusion method*. Prentice Hall.

Computer Associates. (2003, September 1). Information available: http://www.ca.com/products/

Cook, S., & Daniels, J. (1994). *Designing object systems: Object-oriented modelling with syntropy*. Englewood Cliffs, NJ: Prentice Hall.

CSC. (1995). *Catalyst methodology.* (Internal Document). Computer Sciences Corporation.

Dahanayake, A.N.W. (1997). *An environment to support flexible information systems modeling.* (Dissertation), Delft University of Technology, The Netherlands.

DSDM Consortium. (2000). *Dynamic Systems Development method*. DSDM Consortium, http://www.dsdm.org/ (September 2001).

D'Souza, D.F., & Wills, A.C. (1999). *Objects, components, and frameworks with UML: The Catalysis approach.* Addison-Wesley.

Gartner Group. (1997). *Componentware: Categorization and cataloging*. Applications Development and Management Strategies Research Note, by K. Loureiro and M. Blechar. Available: http://www.gartnergroup.com (September 2001).

Graham, I., Henderson-Sellers, B., & Younessi, H. (1997). *OPEN process specification*. Addison-Wesley.

Herzum, P., & Sims, O. (2000). *Business component factory: A comprehensive overview of business component development for the enterprise*. John Wiley & Sons.

Hofstede, ter A., & Verhoef, T. (1996). Feasibility of method engineering. *Information Systems Journal*, *6*, 41-68.

Hong, S., Goor, van den P., & Brinkkemper, S. (1993). A formal approach to the comparison of object-oriented analysis and design methodologies. *Proceedings of the 26th Hawaii International Conference on System Sciences*, Vol IV.

IBM Web Services. (2003, September 1). Available: http://www.ibm/com/webservices

Jacobson I., Booch, G., & Rumbaugh, J. (1999). *The unified software development process.* Reading, MA: Addison-Wesley.

Jacobson, I., Christerson, M., Jonsson, P., & Overgaard, G. (1992). *Object-oriented software engineering – a use case-driven approach*. Reading, MA: Addison-Wesley.

Jacobson, I., Griss, M., & Jonsson, P. (1997). *Software reuse – Architecture, process and organisation for business success*. ACM Press, Addison-Wesley Longman.

Kumar, K., & Welke, R.J. (1992). Methodology engineering: a proposal for situation-specific methodology construction. In W.W. Cotterman & J.A. Senn (Eds.), *Challenges and strategies for research in systems development* (pp. 257-269). John Wiley & Sons.

ODP. (1996). *International Standard Organisation (ISO), information technology - Open distributed processing - Reference model: Overview, foundations, architecture and architecture semantics*. ISO/IEC JTC1/SC07, 10746-1/4, ITU-T Recommendations X.901/904.

OMG Object Management Group. (2003, September 1). *MDA - Model Driven Architecture*. Available: http://www.omg.org/mda/

Rossi, M. (1998). *Advanced computer support for method engineering- Implementation of CAME environment in MetaEdit+.* (Dissertation). University of Jyvaskyla.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorenson, W. (1991). *Object-oriented modeling and design*. Prentice Hall.

Siau, K., & Rossi, M. (1998). Evaluating information modeling methods - a review. In D. Dolk (Ed.), *31st Hawaii International conference on System Sciences (HICSS-31), Vol. V,* (pp. 314-322).

Siegel, J. (2000). *CORBA 3: Fundamentals and programming*. OMG Press, John Wiley & Sons.

Sol, H.G. (1988). Information system development: A problem solving approach. *Proceedings of 1988 INTEC Symposium Systems Analysis and Design: A research strategy*, Atlanta, Georgia.

Stojanovic, Z., Dahanayake, A., & Sol, H. (2001). A methodology framework for component-based systems development support. *Proceedings of the 6th CAISE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, Interlaken, Switzerland, (pp. XIX-1 - XIX-14).

Stojanovic, Z., Dahanayake, A.N.W., & Sol, H.G. (2000). *Integrated component-based framework for effective and flexible telematics application development*. Technical report. ISBN: 90-76412-13-8. Delft University of Technology.

Szyperski, C. (1998). *Component software: Beyond object-oriented programming*. ACM Press, Addison-Wesley.

Tolvanen, J-P. (1998). *Incremental method engineering with modeling tools.* (Dissertation). University of Jyvaskyla.

W3C World-Wide-Web Consortium. (2003, September 1). *XML, SOAP, WSDL*. Available: http://www.w3c.org/

Welke, R.J. (1994). The shifting software development paradigm. *Data Base,* 25(4), 9-16.

SECTION II:

DATABASE DESIGN AND DEVELOPMENT: ISSUES AND SOLUTIONS

Chapter IV

Improving the Understandability of Dynamic Semantics:
An Enhanced Metamodel for UML State Machines

Eladio Domínquez, Universidad de Zaragoza, Spain

Angel Luis Rubio, Universidad de La Rioja, Spain

María Antonia Zapata, Universidad de Zaragoza, Spain

ABSTRACT

A clear understanding of the dynamic semantics of languages involved in the representation of behavior is essential for a large and varied audience such as final users of these languages, CASE tool builders or method engineers. This chapter introduces a proposal aimed at achieving such an understanding by suggesting a different metamodeling approach. This approach is based on a two layer architecture which puts forward the explicit distinction between the generic behavior represented in a dynamic model (Base Layer) and the behavior represented in relation to a particular situation (Snapshot Layer). Using this architecture as a starting point, a metamodel of UML State Machines is proposed, which consists basically of two UML class diagrams (one diagram for each layer of the architecture) and two maps. These maps represent, respectively, the determination of the initial status and the process performed by a run to completion step as defined in the UML semantics.

INTRODUCTION

The statechart technique is a visual formalism defined as an enhancement of finite-state machines, originally developed by D. Harel (Harel, 1987) to specify complex reactive systems. Much literature has been written on this topic in recent years and in particular, a large number of variants of the technique have been proposed (Beek, 1994). More recently, the success of the statechart formalism has received a major boost since an object-oriented adaptation of the technique, namely State Machines[1], has been adopted as part of the Unified Modeling Language (OMG, 2003; Rumbaugh, Jacobson & Booch, 1999).

There are many works which define a complete formal semantics of Harel's Statecharts (see, for example, Ehrig, Geisler, Klar & Padberg, 1997; Harel, Pnueli, Schmidt & Sherman, 1987; Harel & Politi, 1998; Hong, Kim, Cha & Kwon, 1995; Hooman, Ramesh & Roever, 1992; Beek, 1994; Maggiolo-Schettini, 2003). However, a known shortcoming of UML State Machines is that in the UML specification document (OMG, 2003), although the syntax and static semantics of State Machines are precisely stated, the dynamic semantics is not rigorously defined (Engels, Haussmann, Heckel & Sauer, 2000; Latella, Majzik & Massink, 1999; Lilius & Paltor, 1999). Undoubtedly, a precise specification of the behavior of State Machines is essential for a large and varied audience. For instance, the final users of the language (such as system analysts and designers) need at least an overall but accurate idea of how a state machine behaves. Secondly, CASE tool builders interested in supporting State Machines greatly benefit from having an unambiguous specification of the language. Finally, method engineers would use a precise specification of State Machines to analyze issues such as language adaptability, comparison with other behavioral approaches, transformation, and so on. This complex situation has resulted in the definition of a precise dynamic semantics of State Machines being the subject of recent intensive research (Borger, Cavarra & Riccobene, 2000; Engels et al., 2000; Jin, Esser & Janneck, 2002; Latella et al., 1999; Lilius & Paltor, 1999; Mann & Klar, 1998; Reggio, 2002; Reggio, Knapp, Rumpe, Selic & Wieringa, 2000; Varro, 2002).

The problem is that the majority of approaches that try to establish a precise dynamic semantics of State Machines make use of formal notations such as Rewrite Rules (Kwon, 2000; Lilius & Paltor, 1999), Hierarchical Automata (Latella et al., 1999), Abstract State Machines (Borger et al., 2000) or Object Z (Mann & Klar, 1998). However, like other authors (Engels et al., 2000; Reggio, 2002), we think that these approaches have the drawback of being difficult to read and understand, and therefore they are not wholly suitable since dynamic semantics must be precisely established but in such a way that the understandability and readability of the specification is facilitated.

Without neglecting the need for formal notations when issues such as verification or model checking have to be dealt with, we propose to adopt a metamodeling approach which is a widely accepted way of improving the properties of understandability and readability (Hofstede & Verhoef, 1997; Verhoef, 1993). This proposal is based on a two-layer architecture we outlined in Domínguez, Rubio and Zapata (2000a, 2000b). This architecture makes explicit the distinction between the generic behavior represented in a dynamic model (Base Layer) and the behavior represented in relation to a particular situation (Snapshot Layer). In addition, the concept of movement from a current situation to another is captured by using the notion of mapping. Using this architecture as a starting point, a metamodel of UML State Machines is proposed, which consists basically of two UML class diagrams (one diagram for each layer of the architecture) and two maps. These maps represent, respectively, the determination of the initial status and the process performed by a run-to-completion step as defined in the UML semantics.

The chapter is organized as follows. In the following section we compare our proposal with other related works. This is followed with a section detailing our view of dealing with the behavior of dynamic systems, presenting the architecture in which we base our approach. Next, we show the metamodel of UML State Machines we propose. And, finally, conclusions and plans for future work are presented.

RELATED WORK

There are several relevant aspects that must be outlined with regard to the comparison of the approach we propose in the present chapter with other works in the literature. First of all, it must be noted that our proposal provides a complete formalization of State Machines, whereas, Kwon (2000) and other works (Engels et al., 2000; Gnesi, Latella & Massink, 1999; Latella et al., 1999) consider only some basic constructs, rendering their proposals incomplete. Among these, it is worth comparing our approach with Engels et al. (2000), since in this work a metamodeling approach is also proposed for representing the dynamic semantics of State Machines. These authors propose to extend the UML State Machines metamodel (OMG, 2003) with state information that can be viewed as providing the metamodel with information of the Snapshot Layer of our proposed architecture. Apart from this shared issue, the remaining aspects are quite different mainly because they adopt a different way of metamodeling the run-to-completion step: they use collaboration diagrams and we use the notion of map. Reggio (2002) also proposes a metamodeling approach which can be interpreted according to the architecture we present. On the one hand, in Reggio (2002), Labelled Transition Systems are used as the semantic domain of state machines, by means of which aspects related with the Snapshot Layer are captured. On the other hand, they propose the use of Labelled Transition Diagrams in order to represent the change of state within the statechart, instead of mappings as in our proposal.

The notion of map or transformation has been claimed by several authors (see, e.g., Domínguez & Zapata, 2000; Domínguez, Zapata & Escario, 2000; Marttiin, Harmsen & Rossi, 1996; Saeki, 2002; Hofstede & Verhoef, 1997; Verhoef, 1993) to be a necessary artifact for solving similar problems within the field of method engineering, problems such as method interoperability or method adaptation. This necessity has been recently recognized by the UML community with the advent of the Model Driven Architecture, MDA (Miller & Mukerji, 2003). Within this architecture, models are leveraged to be primary artifacts during software development, and so are transformations. MDA is based on several OMG standards, such as UML or MOF — the Meta Object Facility (OMG, 2002a), considered as the meta-metamodel of the UML metamodel (OMG, 2003, pp. 2-6). In particular, the ongoing process of development of the new UML 2.0 and MOF 2.0 embodies the Request For Proposals of the MOF Query/Views/Transformations (QVT) (OMG, 2002b), also known as 'Unified Transformation Language'. Other artifacts have been proposed in the literature for specifying the sequence of steps of a procedure. For example, the value of process modeling for representing in a rigorous and explicit way this type of functional aspect is proved in Song and Osterweil (1994). An objective, in-depth analysis of which artifact is the most suitable for representing the procedures for calculating the initial status and the next status in the context of representation of behavior remains an ongoing project.

Finally, unlike other approaches cited, our work has a broader scope, since the use of a metamodeling perspective brings us to a level independent of the specific case of State Machines. This perspective has the advantage of making our approach applicable to other representation techniques for dynamic systems, such as Petri Nets (Peterson, 1981) or the different variants of the statechart technique. Furthermore, our proposed architecture aims to be also applicable within the method engineering context (Brinkkemper, 1996; Brinkkemper, Lyytinen & Welke, 1996; Dietzsch, 2002; Hofstede & Verhoef, 1997). Method engineering, as the engineering discipline for designing, constructing and adapting methods, techniques and tools, has to deal, in particular, with techniques intended to represent some kind of behavior. Therefore, a method-independent way of representing the behavior of models seems to be a valuable artifact for method engineers. However, as is claimed in Saeki (2000), most of the existing metamodeling techniques used with this goal (see, e.g., Brinkkemper, Saeki & Harmsen, 1999; Kelly, Lyytinen & Rossi, 1996; Saeki, 1995) focus their efforts on representing the structural artifacts provided by the methods, leaving out essential aspects of the behavior. In this context, the architecture suggested in this chapter can help to broaden the metamodel definition provided by any metamodeling technique, taking into account behavioral aspects.

AN ARCHITECTURE FOR BEHAVIOR

The UML gathers several sub-languages that have been designed with the main aim of representing some kind of system behavior, such as *State Machine Diagrams* or *Sequence Diagrams.* More specifically, five (out of nine) types of UML diagram are involved in the representation of behavior, and State Machines in particular is the third most complex type of UML diagram (after the much more complex core type *Class Diagrams* and the slightly more complex *Component Diagrams*), as is shown in the complexity study in Siau and Cao (2001).

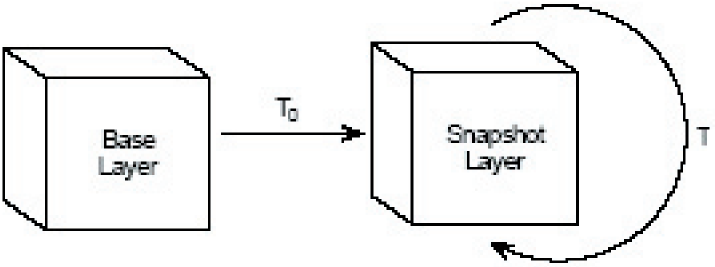
The purpose of representing behavior is common to other modeling-related fields. For instance, within the modeling of reactive systems, Palanque et al. (Palanque, Bastide, Dourte & Sibertin-Blance, 1993) classify three approaches to represent this kind of system, namely state-based, event-based and Petri Nets-based modeling approaches, and several techniques (or variants of existing techniques) have been developed following each paradigm. This diversity of languages and techniques suggests that it would be very valuable to have an infrastructure for representing behavior aspects in a language-independent way.

In order to provide a sound support for the representation of the behavior features of different languages, we describe in this chapter a full version of the architecture outlined in Domínguez et al. (2000a). This architecture has been designed to serve as an abstract framework, and it is not linked to any particular technique or formalism. The architecture suggested consists of two layers, namely the *Base Layer* and the *Snapshot Layer,* and two maps, denoted T_0 and T (see Figure l).

Overview of the Architecture

The Base Layer captures those aspects that appear to be independent from any particular situation, and the Snapshot Layer gathers the aspects characteristic of any particular situation. If we consider the dynamics of a system as if it were a film, the Snapshot Layer describes each frame, that is to say, each one of the potential situations in which the system

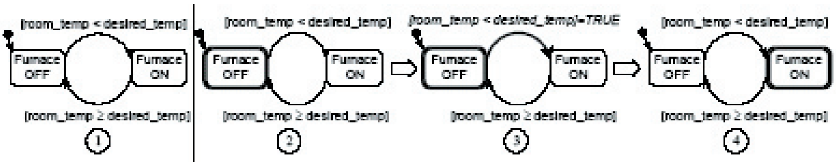
Figure 1: A representation of the architecture



can lie. The Base Layer describes those elements that are perceived as independent from any particular situation; to some extent, this layer represents those base elements that we would see permanently in each frame, provided that we were able to see the whole film at one glance. To clarify the meaning of these aspects, we now review a fragment of the sample dynamic model shown in Rumbaugh, Blaha, Premerlani, Eddy and Lorensen (1991), which represents a programmable thermostat. In this example, the comparison between the actual room temperature and the programmed (desired) temperature is identified permanently and independently of which of these temperatures is higher. Therefore, the *comparison* is a base (status-independent) feature. At any given moment, one of the temperatures will be higher, but this situation will vary as the system develops: the *status of the comparison* is a purely dynamic feature.

The differences between the two layers that we have shown with respect to the model of a system can also be identified at higher levels of abstraction. A detailed discussion about modeling and metamodeling levels goes beyond the scope of this chapter; for such a discussion, see for instance Smolander, Lyytinen, Tahvanainen and Marttiin (1991) or Hofstede and Verhoef (1997). Henceforth, we will use the terminology of a particular approach to metamodeling, that of the Four Layer Metamodeling Architecture (OMG, 2003, pp. 2-5). In this Architecture, the Metamodel Level "defines the language for specifying a model." At this level, the differences between the base (status-independent) features and the snapshot (status-dependent) features are revealed in the modeling artifacts that each specific language, technique or method provides to represent one or other feature. For example, in Rumbaugh et al. (1991), the Statecharts formalism is used to create a model of the thermostat. Under our perspective, Statecharts concepts such as *state, transition, condition* and *variable* belong to the Base Layer, and so the 'standard' statechart (1) of Figure 2 would become an instance belonging to this aspect (in particular, a condition is used to model the comparison between the temperatures). On the other hand, concepts such as *active state, compound transition, enabled compound transition, true condition,* etc., are related to the Snapshot Layer. Therefore, diagrams (2), (3) and (4) of Figure 2, which represent several consecutive situations of the thermostat by means of a widened notion of statechart, are related to the Snapshot Layer (and thus, the value of the condition models the status of the comparison). It must be noticed that the standard statechart notations do not offer graphical representations for the concepts we have gathered in the Snapshot Layer, such as *active state* or *enabled transition.* In spite of this, several authors and tool developers have represented in a visual manner some of these purely dynamic aspects of the behavior of a statechart, and they have chosen graphical representations for such dynamic aspects. For instance, several

Figure 2: Statecharts diagrams for thermostat furnace relay


[room_temp < desired_temp]
Furnace OFF
Furnace ON
[room_temp ≥ desired_temp]
1
[room_temp < desired_temp]
Furnace OFF
Furnace ON
[room_temp ≥ desired_temp]
2
[room_temp < desired_temp]=TRUE
Furnace OFF
Furnace ON
[room_temp ≥ desired_temp]
3
[room_temp < desired_temp]
Furnace OFF
Furnace ON
[room_temp ≥ desired_temp]
4

illustrative examples in Harel and Naamad (1996) have been depicted using a shaded box symbol to represent active states, and the Rhapsody tool by I-Logix uses a thicker line to represent enabled transitions during the animation of statechart models. We have made use of a similar approach for the specification of the diagrams (2, 3, 4) of Figure 2. It is necessary to note that we are not stating that this kind of notation must be used by the analyst during modeling, but, in order to specify in detail the behavior of statecharts, a clear distinction has to be made between the basic view and the snapshot view notions. It seems clear that the support of graphical examples and specifications can be of great help in order to clarify this subtle distinction.

The other fundamental elements of the architecture are the maps traced from the Base Layer to the Snapshot Layer (map T_0) and from the Snapshot Layer to itself (map T). On the one hand, map T_0 starts from the information available at the Base Layer, and determines one status that is fixed as the representation of the initial status of the system. In the case of the thermostat example we have mentioned, map T_0 will specify the passage from Diagram 1 to Diagram 2 as an initial status. In particular, the setting-up of the state 'Furnace OFF' as the initial active state (since, on sight of the model in Diagram 1, this is the default situation) must be embedded in map T_0. On the other hand, map T, starting from a current status, determines the next status the system will reach. In fact, map T reflects the behavior of the system, enabling the representation of an execution trace of this behavior by means of consecutive applications of the map. With regard to the thermostat, map T will specify the passage from a current status (Diagram 2) to the next status (Diagram 4) (as we show in the next subsection, Diagram 3 represents an intermediate situation). For instance, map T embodies the dynamic principle stating that if 'Furnace OFF' state is active and 'desired-temp' is higher than 'room-temp' then 'Furnace ON' state must become active (and of course 'Furnace OFF' inactive). Up to this point, the explanation of the maps of the architecture lies in the Model Level of the Metamodeling Architecture. At the Metamodel Level, maps T_0 and T formalize respectively the processes of 'fixing the initial status' and 'moving from the current status to another', which are common to every model at the Model Level. These processes will be different according to the modeling language being described. For example, as will be shown in the chapter, in the specific case of UML State Machines, map T specifies the *run-to-completion step.*

Refinement of the Architecture

The proposed architecture can be refined, adapting to our perspective the dimension of *granularity* as stated for instance in Rolland, Souveyet and Moreno (1995), where it is proposed that "a single process modeling formalism should accommodate a wide range of model granularity in a homogeneous fashion". In general, the highest levels of complexity of representation of behavior are reached when dealing with the concepts we have gathered in the maps of the architecture. In other words, the more sophisticated the behavior to be modeled, the more complex and hard it is to specify maps T_0 and especially T. Because of this, we propose a kind of refinement that facilitates the specification of both maps, and that allows the expression of the desired degree of detail. For instance, map T could be refined by dividing it into several maps, specifying a chain of *Intermediate Snapshot Stages* starting from the Snapshot Layer (see Figure 3). These stages represent intermediate situations between two consecutive statuses, as they are necessary to detail the usually hard passage from a current status to another. For instance, Diagram 3 of Figure 2 represents one of these intermediate situations for the particular thermostat example. Our perspective has been inspired by the literature: for instance, in the original first Statecharts semantics (Harel et al., 1987), the concept of *micro-step* was introduced to alleviate the difficulties in defining the noticeably more complex concept of *step.* With regard to how many intermediate stages should be specified, and therefore the number of maps, this decision is left to the discretion of the analyst, who has to take into account that the greater the number of stages, the greater the degree of detail. Whatever the number, the composite $T_{n+1} \circ T_n \circ ... \circ T_2 \circ T_1$ must give map T as a result.

System Development Issues

The proposed architecture provides an interpretation of behavioral features that can help in system development at different levels of abstraction, according to the different views that several kinds of users (software engineers, tool developers, method engineers, etc.) have on the subject. For instance, a system analyst or designer would use this approach to behavior to get a more accurate interpretation of the system being modeled. In particular, this modeling task would be facilitated if the language chosen by the software engineer to model the behavioral features was precisely described following the guidelines of the architecture.

In turn, such behavioral modeling languages can be supported by CASE tools, which have been recognized as being of great value in software development. The consideration of the proposed architecture can provide guidance for the analysis of existing CASE tools and the development of new ones, according to the developers' purposes. Focusing again on the statecharts formalism, and as has been previously stated, a standard statechart model

Figure 3: Refinement of map T of the architecture

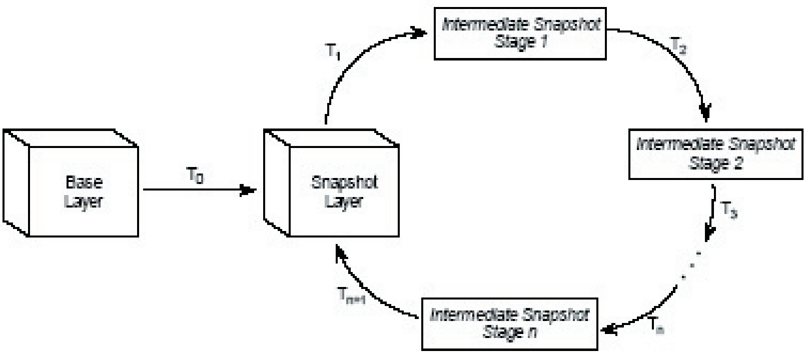
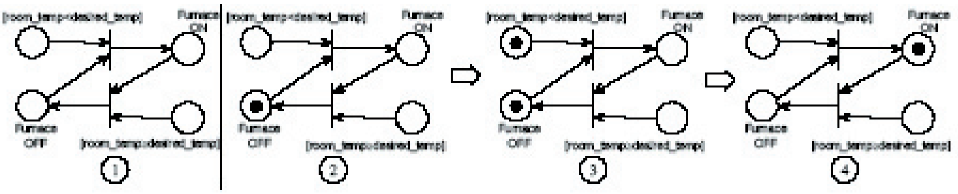

Base Layer
T0
Snapshot Layer
T1
Intermediate Snapshot Stage 1
T2
Intermediate Snapshot Stage 2
T3
Tn
Intermediate Snapshot Stage n
Tn+1

Figure 4: Petri Nets diagrams for thermostat furnace relay



corresponds to the Base Layer of the architecture. Therefore, a statecharts drawing tool can concentrate its efforts on this layer. However, a CASE shell developer interested in providing the tool with some simulation or animation features must be aware that the simulated execution of a statechart model involves the combined application of the concepts related to the T_0 map, the Snapshot Layer and the T map, applied to the particular case of the statechart formalism. Some up-to-date, commercial CASE tools supporting UML State Machines are based on similar approaches since they have been provided with some animation features (for example, the above-mentioned Rhapsody tool provides animated views of the modeled application, and in particular, allows the observation of the behavior of a state machine using a color scheme to differentiate between, for instance, active or inactive states). However, it must be stressed that this kind of approach is one part of particular tools, and does not belong to the standard State Machines definition. In any case, a system analyst or designer would benefit from the use of a CASE tool that follows the architecture, since such a tool would provide, as a sub-product, a pattern to the modeling of behavior. In particular, a software engineer should not be burdened with a detailed specification of the maps of the architecture, but he/she must be aware of their availability in the tool.

Furthermore, it is necessary to stress that the issues we are analyzing are not exclusive to State Machines, although this language is our main example. For instance, with regard to Petri Nets (Peterson, 1981), within our architecture, a non-marked Petri Net belongs to the base view, whereas a Petri Net together with a particular marking belongs to the snapshot view. The process of firing transitions and re-marking of places is equivalent to the change of status in State Machines. Figure 4 illustrates this situation by means of a petri net model of the thermostat, representing the base layer (1), two consecutive statuses of the snapshot layer (2 and 4) and an intermediate situation (3). It is worth noting that we are not stating here that the statechart model in Figure 2 and the petri net model in Figure 4 are (or are not) equivalent (see Palanque et al., 1993, for a related discussion). What this example shows is that the differences between the base and the snapshot aspects can be analyzed in a language-independent way, and it is therefore of great value to be provided with a framework that allows the analysis of the behavior of models of techniques in a general way. In particular, as we have explained in the 'Related Work' section, the proposed architecture can be a valuable artifact for method engineers in order to design, construct or adapt a method to be used within a particular project.

UML STATE MACHINES METAMODEL

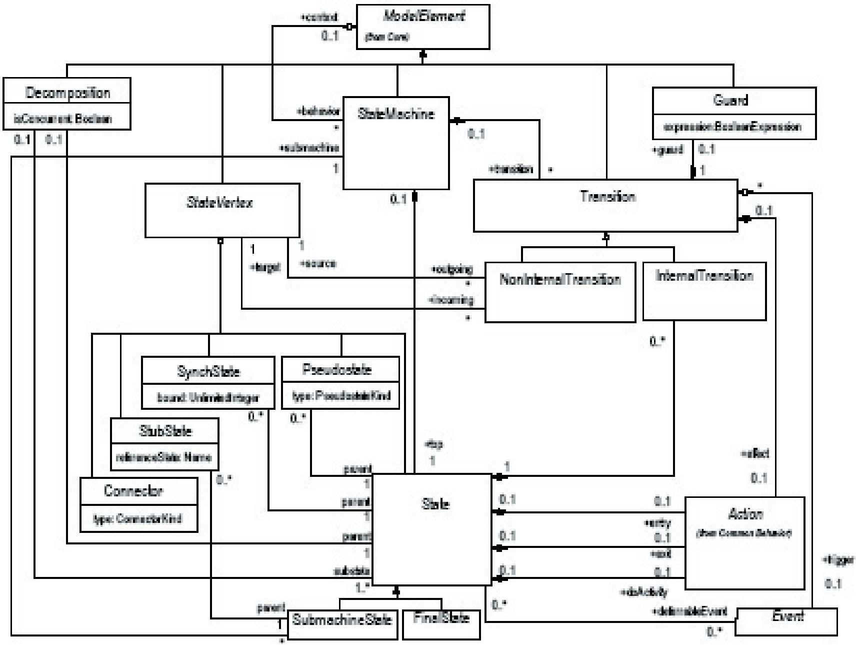
In the next subsections we define a metamodel which captures the syntax, static semantics and dynamic semantics of UML State Machines, bringing into play the architecture we have described in the previous section.

Base Layer

The Base Layer corresponds to the syntax and static semantics of State Machines since they capture those aspects that appear to be independent from any particular situation. In order to represent these features in a similar way as is proposed in the UML Specification, we propose to use a UML class diagram that we have called *Base Diagram* (see Figure 5), and a set of Object Constraint Language (OCL) expressions. We have used the class diagram proposed in the UML Specification (OMG, 2003, pp. 2-141) as a starting point and we have modified it in several ways, with the main aim of making explicit some restrictions that State Machines must hold and taking into account the Statecharts metamodel we proposed in Domínguez et al. (2000a; 2000b) using the Noesis metamodeling technique (Domínguez, Zapata & Rubio, 1997).

One of the main differences between the class diagram of Figure 5 and the class diagram proposed in OMG (2003) is that we have interpreted the fact of being a simple or a composite state not as an intrinsic property of the state but as a characteristic derived from

Figure 5: Base Diagram



the fact of whether the state is decomposed into substates or not. This interpretation has led us to include in the Base Diagram a new class, Decomposition, which represents the existence of a relationship between a state and a set of substates. The decomposition can be of two kinds: concurrent or not concurrent. This new proposal solves a latent problem related to simple and composite states. During the state machine construction process, a state which initially was considered as simple can now be considered to be compound. According to the version proposed in OMG (2003), this situation would entail an instance of the SimpleState class becoming an instance of the CompositeState class, which leads to the problem known as *object reclassification anomaly* (Chu & Zhang, 1997; Drossopoulou, Damiani, Dezani-Ciancaglini & Giannini, 2002). This problem does not arise in our proposed version since a simple state becomes a composite state by adding an instance of decomposition (without it being necessary to reclassify any object). Another possible solution for avoiding the object reclassification problem is based on the consideration of only one class State, in such a way that the instances (objects) of that class could be either simple or composite. This solution would entail the introduction of a recursive relationship between the class State and itself, in order to represent the hierarchy of states inside a state machine. However, as is claimed in Lee (1999), in the context of the entity-relationship model, "the semantics of recursive relationships are quite difficult to grasp". We have adopted the solution discussed above that includes a Decomposition class, since we think this solution can improve understanding of the metamodel.

The inclusion of the class Decomposition leads to the class State having only two subclasses: FinalState and SubmachineState. Two restrictions are stated in the Well-Formedness Rules in OMG (2003) with regard to the SubmachineState class: (1) only stub states are allowed as substates of a submachine state and (2) submachine states are never concurrent. In our proposal, we have substituted the first restriction including an association between the StubState and SubmachineState classes (representing the relation of substate) and a restriction which states that 'submachine states must not take part in any decomposition as parent'. From the restriction we consider it follows that a submachine state is never concurrent, so no other restrictions have to be stated.

Another modification we propose is related to the classification of the different types of state vertices. We have considered it convenient to differentiate between, on the one hand, the initial, deep history and shallow history state vertices (which remain considered as Pseudostates) and, on the other hand, the join, fork and junction vertices (which are considered as Connectors). The reason for this lies in the fact that initial, deep history and shallow history state vertices have to be associated with a state (which will be their parent), whereas join, fork and junction vertices are not associated with any state, and are strongly related with the definition of *compound transitions.* Obviously they have to be represented inside the graphic representation of a state but it does not matter what the state is. In this sense we share the views of several authors of formalizations of Statecharts, as they do not associate any parent to these connectors (Harel & Naamad, 1996). It must be assumed that the inclusion of new classes increases the complexity of the metamodel. In this case, the distinction between Pseudostates and Connectors has a conceptual nature, and therefore we have considered it essential in order to get a better understanding of the State Machines language.

The last change we propose is that we have specialized the Transition class into two subclasses according to whether a transition is an internal transition associated to a state or not. Our aim is to emphasize the fact that the source and target of an internal transition are always the same and can only be a state (it can not be any other type of StateVertex).

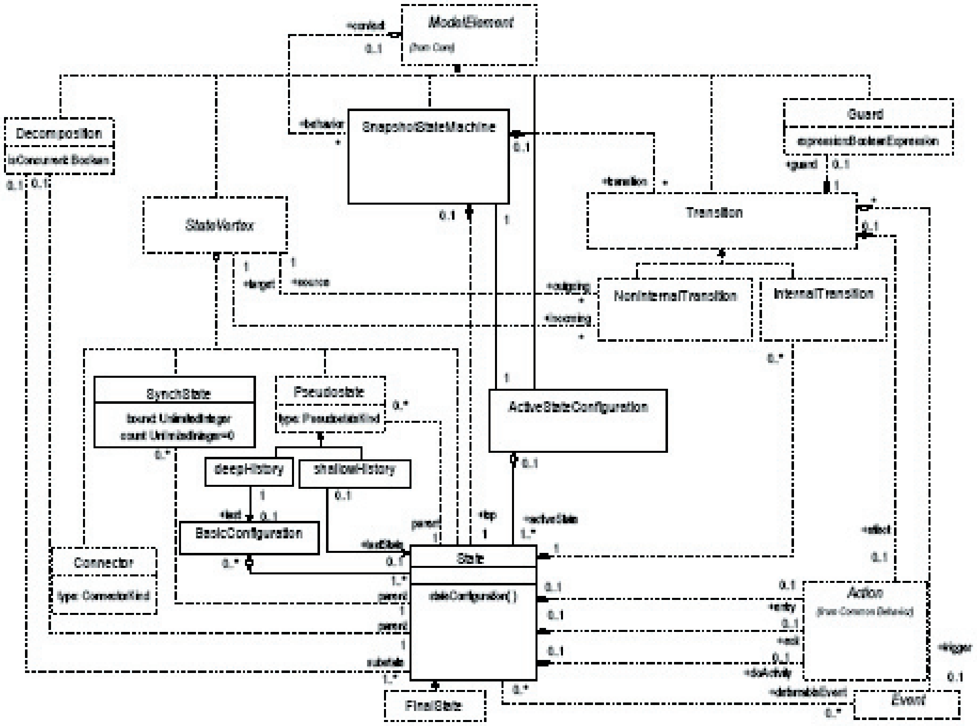
The static semantics of the new proposed UML class diagram is defined by means of OCL expressions. Basically they are the same rules proposed in OMG (2003), but with some additional ones. We do not include them here because they do not represent any relevant contribution.

Snapshot Layer

The dynamic (execution) semantics of State Machines is related to the Snapshot Layer since this layer has to capture those aspects that are related to the status of a state machine at a given moment. In order to specify the dynamic semantics, we take into account that the actual behavior of a statechart "consists of a series of detailed snapshots (...). The first in the sequence is the initial status, and each subsequent one is obtained from its predecessor by executing a step" (Harel & Naamad, 1996). This excerpt is taken from the semantics definition source of a particular CASE tool (STATEMATE, (Harel & Politi, 1998)), but it can be considered as a widely accepted elemental description of statecharts behavior. On the other hand, in the UML Specification the dynamic semantics is described in English prose, and in these natural language explanations several concepts have to be introduced, as for example 'active state configuration'.

In the same way that the base concepts are represented by means of a UML class diagram, we propose to adopt an analogous approach with the snapshot concepts, by means of the construction of another class diagram that we have called *Snapshot Diagram* (Figure 6). It

Figure 6: Snapshot Diagram



must be noticed that, in order to help the reader to easily detect the differences between both class diagrams, we have used discontinuous lines for depicting the classes and relationships of the Snapshot Diagram that remain unchanged with regard to the Base Diagram[2].

As an initial remark regarding the Snapshot Diagram and the notion of submachine state, let us note that in OMG (2003, pp. 2-158), it is said that "a submachine state is a convenience that does not introduce any additional dynamic semantics" and that "it is semantically equivalent to a composite state." For this reason, from now on, we are going to suppose that the base state machine does not contain any submachine state (if this is not the case, an equivalent state machine should be constructed previously). As a consequence of this, we have not included in the Snapshot Diagram either the SubmachineState or the StubState Classes.

Thus, the Snapshot Diagram belongs to the Snapshot Layer and captures, together with the status-independent concepts, those concepts necessary to determine the status of a state machine at a given moment. In particular, each one of the 'snapshots' of the sequence described in the previous quotation would correspond to a model of the Snapshot Diagram we propose. In other words, a 'snapshot' could be described by means of an UML Object Diagram, which is an instance of the Snapshot Diagram. The way in which a snapshot is obtained from its predecessor (i.e., the representation of a step) will be analyzed in the following subsection.

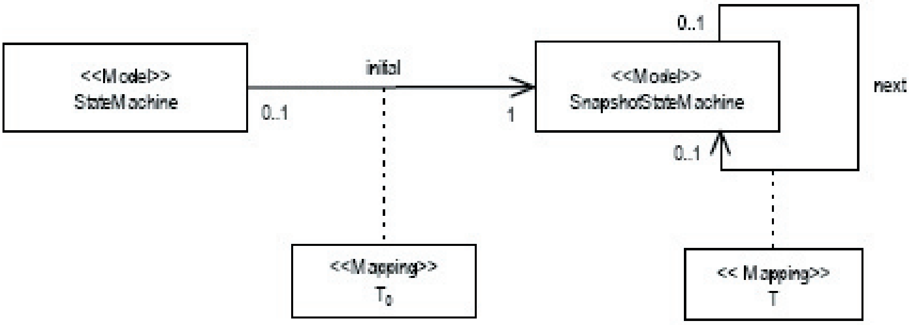
It can be observed in the Snapshot Diagram that the ActiveStateConfiguration class is added, which represents the states which are active at a given moment. Furthermore, it is necessary to capture the information relative to the history of the behavior of the state machine. This has been done by incorporating information about the last active substate and the basic configuration relative to a state. Finally, the SynchState class has a new attribute associated to it which represents, for each object of the class, the difference between the number of times its incoming and outgoing transitions are fired.

T_0 and T maps

Once the Base and Snapshot Diagrams have been determined, the metamodel has to be completed, following the proposed architecture, with two maps. On the one hand, the procedure which must be followed in order to calculate the initial status has to be determined. This procedure (represented in the metamodel by means of a map T_0) calculates a snapshot state machine (which has to be a model of the Snapshot Diagram) from a base state machine (which has to be a model of the Base Diagram). On the other hand, it is necessary to define the procedure which has to be followed in order to calculate, from a snapshot state machine representing the current status, that which represents the next status. This second procedure (represented in the metamodel by means of a map T) captures the meaning of a run-to-completion step, and it allows the construction of a sequence of models of the Snapshot Diagram, representing an execution trace. Figure 7 can help to clarify the relationships between the several kinds of models and mappings included in the metamodel. This figure has been inspired by the philosophy of MDA (Miller & Mukerji, 2003), and in particular by the MDA Metamodel Description, in which mappings (as well as models) are represented by means of classes.

We propose to make use of a notion of map between class diagrams in order to formalize the procedures represented by T_0 and T. Since a precise and exhaustive definition of the notion of map goes beyond the scope of this chapter, we will consider a very general defini-

Figure 7: Relationships between models and mappings



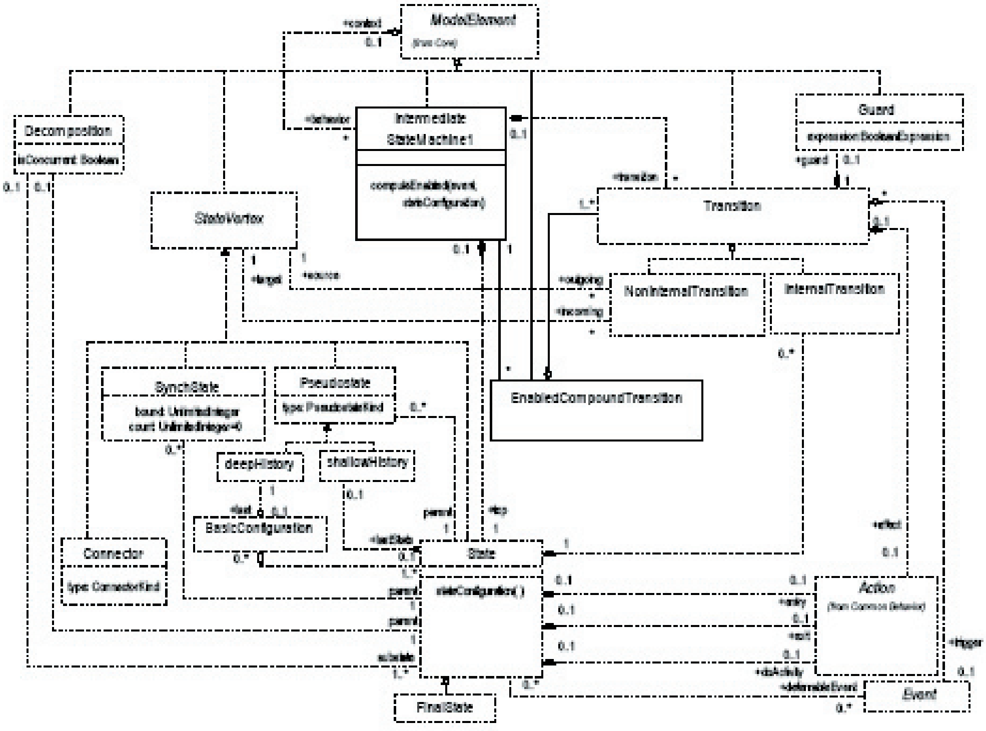
tion of the concept. Given two class diagrams *C* and *C'* at the Metamodel Level, we define a map from *C* to *C'* as a method which allows a model of *C'* to be determined starting from a model of *C*. In our particular case, the maps T_0 and *T* that we have informally described above have to be defined. More specifically, T_0 has to be a map from the Base Diagram to the Snapshot Diagram and *T* has to be a map from the Snapshot Diagram to itself.

The specification of each map contains, firstly, its signature, which specifies the name of the map, its parameters and the specification of the result that is obtained after the application of the map. One of the parameters determines the model that will be modified, and the rest determine the specific information that each map needs to know in order to accomplish its effect. In addition, a schematic algorithm of the map is given. We are going to comment on those aspects we consider important in relation to T_0 and *T* maps.

The setting-up of the initial status of a state machine is modeled in our metamodel by means of the application of map T_0 (see Table 1), that starting from a basic state machine *M* leads to a snapshot state machine M^o_D. Map T_0 does not include in M^o_D any information about the history of the state machine since it represents the first status. As for the initial active state configuration, this is obtained calculating the default state configuration of the top state, by means of the stateConfiguration operation of the State class (see Figure 6 and Table 1). Map T_0 should be specified with a higher degree of detail, probably using a refinement of the map, but we have not analyzed it here for space reasons (note that in OMG (2003) this matter is not studied in detail either).

The run-to-completion step is modeled through the map *T* that results in a snapshot state machine M'_D starting from another M_D. An accurate description of a run-to-completion step is a very complex task, which is likely to be unapproachable all at once. We have been inspired by the algorithmic description of step in Harel and Naamad (1996), where three subtasks (in turn non-trivial) are specified to define a step. This idea has led us to consider a two-stage refinement of the map *T*. These stages are represented in the metamodel by means of two additional class diagrams (Figures 8 and 9), in which we have used the same graphical notation previously explained (so that the elements of each class diagram that remain unchanged with respect to the previous one are depicted using discontinuous lines). As a consequence of the specification of both class diagrams, three intermediate maps, T_i,

Figure 8: First Intermediate Snapshot Diagram



i = 1, 2, 3 are also specified (see Table 1). These maps do not correspond exactly with the above-mentioned three subtasks in Harel and Naamad (1996), since we are dealing with UML State Machines and Harel and Naamad (1996) deal with a different version of Statecharts. More specifically, map T_1 calculates the enabled compound transitions (ECTs) taking into account the current event instance, the active state configuration, the distinct connectors (in order to compute the compound transitions), the values of variables and guards, and so on. This is done by means of the computeEnabled operation of the IntermediateStateMachine1 class (see Figure 8 and Table 1). A step is calculated by means of T_2 starting from the set of ECTs, solving the possible conflicts between transitions in this set and taking into account the firing priorities. The computeStep operation of the IntermediateStateMachine2 class is used in this case (Figure 8 and Table 1). Finally, the step is executed by means of map T_3, which causes the execution of some actions and gives rise to a new snapshot state machine, representing the new current status. The execute operation of class Step performs this task (Figure 9 and Table 1). In turn, any of these intermediate maps could also be refined, with new stages and maps being added until the desired degree of detail is reached. Furthermore, each one of the operations used in these maps must be implemented by means of a method, but we do not include them here for space reasons. In this respect, it is important to mention the detailed algorithm shown in Lilius and Paltor (1999), which specifies the run-to-completion step.

Figure 9: Second Intermediate Snapshot Diagram

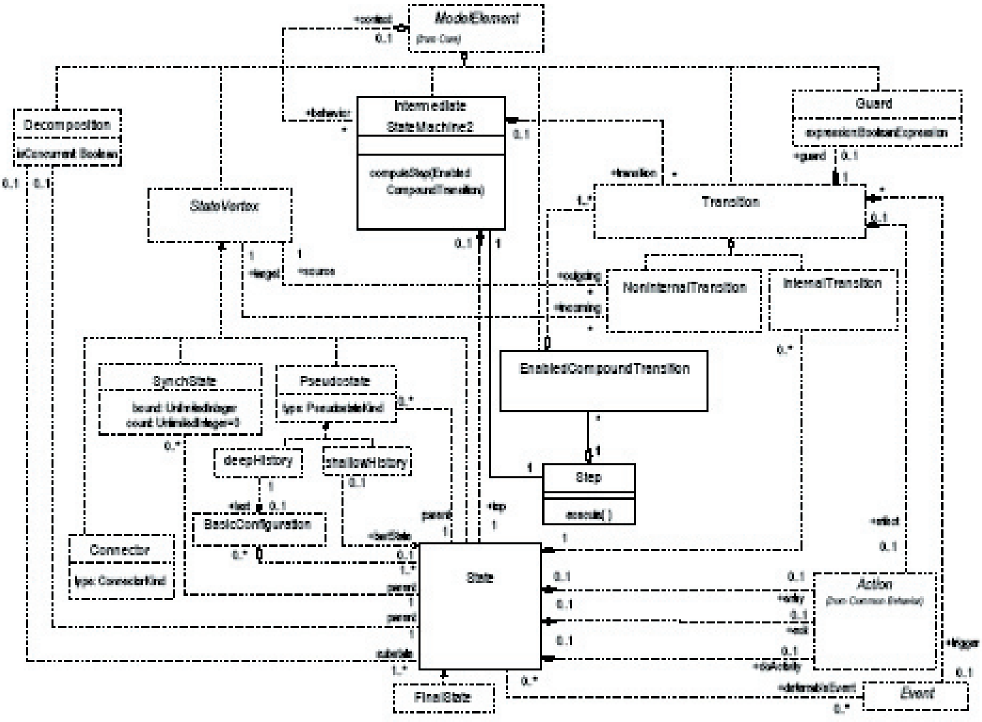


Table 1: Maps

$T_0(M : \mathit{StateMachine}) \Rightarrow M^0_D$: *SnapshotStateMachine* create M^0_D as copy of M with M^0_D do activeStateConfiguration.activeState←top.stateConfiguration( )
$T_1(M_D : \mathit{SnapshotStateMachine, currentEvent : Event}) \Rightarrow M_1$: *IntermediateStateMachine1* create M_1 as copy of M_D with M_1 do enabled.CompoundTransition←computeEnabled(currentEvent, M_D.activeStateConfiguration)
$T_2(M_1 : \mathit{IntermediateStateMachine1}) \Rightarrow M_2$: *IntermediateStateMachine2* create M_2 as copy of M_1 with M_2 do step←computeStep(M_1.enabledCompoundTransition)
$T_3(M_2 : \mathit{IntermediateStateMachine2}) \Rightarrow M'_D$: *SnapshotStateMachine* create M'_D as copy of M_2 with M'_D do M_2.step.execute( )

CONCLUSIONS

In this chapter we have suggested a two-layer architecture which helps to represent behavior features. This architecture has been used as a basis for defining a metamodel of UML State Machines. This metamodel, which is an extension of the metamodel proposed in OMG (2003), basically consists of two class diagrams (Base Diagram and Snapshot Diagram) and two maps. The main contributions of our proposal, with respect to other approaches, are the distinction between Base and Snapshot Diagrams and the representation of the run-to-completion step by means of a map.

From the vast literature on the subject it can be seen that the analysis of behavioral aspects adds great complexity to modeling and metamodeling tasks. This level of complexity is likely to be the reason why a widely accepted (neither formal nor metamodeling or other) approach to behavior and behavioral languages has not yet been found. The perspective of the present chapter aims to make a significant contribution in this sense, although it has its limitations. In particular, the high number of models (and metamodels) that have to be created using this approach could lead to efficiency issues. We think that future development of automated tool support (CASE and MetaCASE tools) can be of great help to solve such issues.

We have restricted our attention to an analysis of the dynamic semantics of a single state machine. However, as pointed out in Jürjens (2002), in order to provide complete executable UML specifications, message-passing between different diagrams must also be formalized. For this reason, in future work, we will investigate how message-passing can be captured within our approach. Furthermore, there is no general agreement on the meaning of inheritance when considering the dynamic behavior of objects (Basten & Aalst, 2001), so that aspects related with the refinement of UML State Machines (subtyping, inheritance and general refinement) remain for future work. Finally, the improvement of the specification language used for the transformations also remains an ongoing project. In this respect, the Action Semantics proposed for UML (OMG, 2003) and QVT (OMG, 2002b) need to be analyzed.

ACKNOWLEDGMENTS

We gratefully acknowledge the helpful discussions and comments of our colleague Dr. Julio Rubio from Universidad de La Rioja, and the fruitful comments of reviewers on a previous version of this work.

This work has been partially supported by DGES, projects TIC2000-1368-C03-01, TIC2002-01626, and by Ibercaja-University of Zaragoza, project IB 2002-TEC-03.

ENDNOTES

[1] We will use the term 'State Machines' whenever we want to refer to the UML version of the statechart technique; in other cases we will use the term 'Statecharts'.

[2] There are several reasons why we have chosen to use this presentation option. On the one hand, since a notion of comparison between class diagrams does not exist in the UML, there is no UML standard notation that can fully satisfy our requirements. On the other hand, we can not use italics or bold fonts because they are used in UML with

other purposes. Our preferred solution would be use a gray color for the classes and relationships that remain unchanged from one diagram to another, but this solution can be problematic from a technical, printing point of view. Taking all these facts into account, we chose to use discontinuous lines in the notation.

REFERENCES

Basten, T., & Aalst, W. M. P. van der. (2001). Inheritance of behaviour. *The Journal of Logic and Algebraic Programming,* 2, 47-149.

Beek, M. von der. (1994). A comparison of statechart variants. In L. de Roever & J. Vytopil (Eds.), *Formal techniques in real time and fault tolerant systems* (Vol. 863 of LNCS, pp. 128-148). Springer.

Borger, E., Cavarra, A., & Riccobene, E. (2000). Modeling the dynamics of UML state machines. In Y. Gurevich, P. W. Kutter, M. Odersky, & L. Thiele (Eds.), *Abstract state machines 2000* (Vol. 1912 of LNCS, pp. 223-241). Springer.

Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information and Software Technology*, 38, 275-280.

Brinkkemper, S., Lyytinen, K., & Welke, R. J. (1996). *Method engineering. Principles of method construction and tool support.* Champan & Hall.

Brinkkemper, S., Saeki, M., & Harmsen, A. F. (1999). Meta-modelling based assembly techniques for situational method engineering. *Information Systems*, 24(3), 209-228.

Chu, W., & Zhang, G. (1997). Associations and roles in object oriented modeling. In D. Embley & R. Go1dstein (Eds.), *Conceptual modeling ER'97* (Vol. 1331 of LNCS, pp. 257-270). Springer.

Dietzsch, A. (2002). Adapting the UML to business modelling's needs – Experiences in situational method engineering. In J. M. Jézéquel, H. Hußmann & S. Cook (Eds.), *UML 2002 – the Unified Modeling Language* (Vol. 2460 of LNCS, pp. 73-83). Springer.

Domínguez, E., & Zapata, M. A. (2000). Mappings and interoperability: A meta-modelling approach. In T. Yakhno (Ed.), *Advances in information systems ADVIS 2000* (Vol. 1909 of LNCS, pp. 352-362). Springer.

Domínguez, E., Rubio, A. L., & Zapata, M. A. (2000a). Meta-modelling of dynamic aspects: the NOESIS approach. In J. Bézivin & J. Ernst (Eds.), *Proceedings of the ECOOP 2000 International Workshop on Model Engineering* (pp. 28-35). Cannes, France.

Domínguez, E., Rubio, A. L., & Zapata, M. A. (2000b). A way of dealing with behaviour of state machines. In G. Reggio, A. Knapp, B. Rumpe, B. Selic & R. Wieringa (Eds.), *Proceedings of the UML 2000 Workshop Dynamic Behaviour in UML Models: Semantic questions* (pp. 32-37). Institut für Informatik. Ludwig-Maximilians-Universität München.

Domínguez, E., Zapata, M. A., & Escario, I. (2000). Meta-model transformations as a support for method adaptation. In B. Sanchez, N. Nada, A. Rashid, T. Arndt & M. Sanchez (Eds.), *World multiconference on systemics, cybernetics and informatics SCI 2000* (Vol. II Information Systems Development, pp. 44-49).

Domínguez, E., Zapata, M. A., & Rubio, J. (1997). A conceptual approach to meta-modelling. In A. Olivé & J. Pastor (Eds.), *Advanced information systems engineering, CAISE'97* (Vol. 1250 of LNCS, pp. 319-332). Springer.

Drossopoulou, S., Damiani, F., Dezani-Ciancaglini, M., & Giannini, P. (2002). More dynamic object reclassification: FickleII. *ACM Transactions Programming Languages and Systems,* 24(2), 153-191.

Ehrig, H., Geisler, R., Klar, M., & Padberg, J. (1997). Horizontal and vertical structuring techniques for statecharts. In A. W. Mazurkiewicz & J. Winkowski (Eds.), *CONCUR'97 concurrency theory* (Vol. 1243 of LNCS, pp. 181-195). Springer.

Engels, C., Haussmann, J. H., Heckel, R., & Sauer, S. (2000). Dynamic meta-modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. In A. Evans, S. Kent & B. Selic (Eds.), *UML 2000 the Unified Modeling Language* (Vol. 1939 of LNCS, pp. 323-337). Springer.

Gnesi, S., Latella, D., & Massink, M. (1999). Model checking UML statechart diagrams using JACK. In *Proceedings of the IEEE International symposium on high assurance systems engineering.*

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8, 231-274.

Harel, D., & Naamad, A. (1996). The STATEMATE semantics of statecharts. *ACM Transactions on Software Engineering and Methodology,* 5(4), 293-333.

Harel, D., & Politi, M. (1998). *Modeling reactive systems with statecharts: The STATEMATE approach*. McGraw Hill.

Harel, D., Pnueli, A., Schmidt, J. P., & Sherman, R. (1987). On the formal semantics of statecharts. In *Proceedings of the 2nd IEEE Symposium on logic in computer science* (pp. 54-64). IEEE Press.

Hofstede, A. H. M. ter, & Verhoef, T. F. (1997). On the feasibility of situational method engineering. *Information Systems*, 22(6/7), 401-422.

Hong, H. S., Kim, J. H., Cha, S. D., & Kwon, Y. R. (1995). Static semantics and priority schemes for statecharts. In *9th Annual International Computer Software and Applications Conference COMPSAC'95* (pp. 114-120). IEEE Press.

Hooman, J., Ramesh, S., & Roever, W. de. (1992). A compositional axiomatization of statecharts. *Theoretical Computer Science*, 101, 289-335.

Jin, Y., Esser, R., & Janneck, J.W. (2002). Describing the syntax and semantics of UML statecharts in a heterogeneous modelling environment. In M. Hegarty, B. Meyer & N. H. Narayanan (Eds.), *Diagrammatic Representation and Inference, Second International Conference, Diagrams 2002* (Vol. 2317 of LNAI, pp. 320-334). Springer.

Jürjens, J. (2002). A UML statecharts semantics with message-passing. *Proceedings of the 2002 ACM Symposium on Applied Computing* (pp. 1009-1013). ACM.

Kelly, S., Lyytinen, K., & Rossi, M. (1996). MetaEdit+: a fully configurable multi-user and multi-tool CASE and CAME environment. In P. Constantopoulos, J. Mylopoulos & Y. Vassiliou (Eds.), *Advanced Information Systems Engineering, CAISE'96* (Vol. 1080 of LNCS, pp. 1-21). Springer.

Kwon, G. (2000). Rewrite rules and operational semantics for model checking UML statecharts. In A. Evans, S. Kent & B. Selic (Eds.), *UML 2000 the Unified Modeling Language* (Vol. 1939 of LNCS, pp. 528-540). Springer.

Latella, D., Majzik, L, & Massink, M. (1999). Towards a formal operational semantics of UML statechart diagrams. In *3rd International Conference on Formal Methods for Open Object Oriented Distributed Systems* (pp. 331-344). Kluwer Academic Publishers.

Lee, H. K. (1999). Semantics of recursive relationships in entity-relationship model. *Information and Software Technology*, 41(13), 877-886.

Lilius, J., & Paltor, I. P. (1999). Formalising UML state machines for model checking. In R. France & B. Rumpe (Eds.), *UML'99 the Unified Modeling Language* (Vol. 1723 of LNCS, pp. 430-445). Springer.

Maggiolo-Schettini, A., Peron, A., & Tini, S. (2003). A comparison of statecharts step semantics. *Theoretical Computer Science,* 290(1), 465-498.

Mann, S., & Klar, M. (1998). A metamodel for object oriented statecharts. In The Second Workshop on Rigorous Object Oriented Methods, ROOM 2.

Marttiin, P., Harmsen, F., & Rossi, M. (1996). A functional framework for evaluating method engineering environments: the case of Maestro II/Decamerone and MetaEdit+. In S. Brinkkemper, K. Lyytinen & R. J. Welke (Eds.), *Method engineering. Principles of method construction and tool suppport* (pp. 63-86). Champan & Hall.

Miller, J., & Mukerji, J. (2003). MDA Guide version 1.0 omg/03-05-01. Available: http://www.omg.org

OMG. (2002a, April). MOF specification version 1.4 formal/2002-04-03. Available: http://www.omg.org

OMG. (2002b, April). MOF 2.0 Query/Views/Transformations RFP ad/2002-04-10. Available: http://www.omg.org

OMG. (2003, March). UML specification version 1.5 formal/2003-03-01. Available: http://www.omg.org

Palanque, P., Bastide, R., Dourte, L., & Sibertin Blance, C. (1993). Design of user driven interfaces using petri nets and objects. In C. Rolland, F. Bodart & C. Cauret (Eds.), *Advanced Information Systems Engineering, CAISE'93* (Vol. 685 of LNCS, pp. 569-585). Springer.

Peterson, J. L. (1981). *Petri net theory and the modelling of systems*. Prentice Hall.

Reggio, G. (2002). Metamodelling behavioural aspects: The case of UML State Machines. In H. Ehrig, B. J. Kramer & A. Erta (Eds.), *Proceedings of Integrated Design and Process Technology, IDPT-2002.* Society for Design and Process Science.

Reggio, G., Knapp, A., Rumpe, B., Selic, B., & Wieringa, R. (Eds.). (2000). *Proceedings of the UML 2000 Workshop Dynamic Behaviour in UML models: Semantic Questions.* Institut für Informatik. Ludwig-Maximilians-Universität München.

Rolland, C., Souveyet, C., & Moreno, M. (1995). An approach for defining ways of working. *Information Systems,* 20(4), 337-359.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., & Lorensen, W. (1991). *Object oriented modeling and design.* Prentice Hall.

Rumbaugh, J., Jacobson, I., & Booch, G. (1999). *The Unified Modeling Language reference manual.* Addison Wesley.

Saeki, M. (1995). Object-oriented meta modelling. In M. P. Papazoglou (Ed.), *Proceedings of the OOER'95, 14th International Object Oriented and Entity Relationship Modelling Conference* (Vol. 1021 of LNCS, pp. 250-259). Springer.

Saeki, M. (2000). Towards formal semantics of meta-models. In J. Bézivin & J. Ernst (Eds.), *Proceedings of the ECOOP 2000 International Workshop on Model Engineering* (pp. 2-9). Cannes, France.

Saeki, M. (2002). Role of model transformation in method engineering. In A. B. Pidduck, J. Mylopoulos, C. C. Woo & M. T. Ozsu (Eds.), *Advanced information systems engineering, CAISE 2002* (Vol. 2348 of LNCS, pp. 626-642). Springer.

Siau, K., & Cao, Q. (2001). Unified Modeling Language (UML): A complexity analysis. *Journal of Database Management,* 12(1), 26-34.

Smolander, K., Lyytinen, K., Tahvanamen, V. P., & Marttiin, P. (1991). Metaedit: A flexible graphical environment for methodology modelling. In R. Andersen, J. Bubenko Jr. & A. Solvberg (Eds.), *Advanced information systems engineering, CAISE'91* (Vol. 498 of LNCS, p. 168-193). Springer.

Song, X., & Osterweil, L. J. (1994). Experience with an approach to comparing software design methodologies. *IEEE Transactions on Software Engineering*, 20, 364-384.

Varro, D. (2002). A formal semantics of UML Statecharts by Model Transition Systems. In A. Corradini, H. Ehrig, H.-J. Kreowski & G. Rozenberg (Eds.), *Graph Transformations: First International Conference, ICGT 2002* (Vol. 2505 of LNCS, pp. 378-392). Springer.

Verhoef, T. F. (1993). *Effective information modelling support.* Unpublished doctoral dissertation, Delft University of Technology, Delft, The Netherlands.

Chapter V

Metrics for Workflow Design: How an Information Processing View on Business Processes Helps to Make Good Designs

Hajo A. Reijers, Technische Universiteit Eindhoven, The Netherlands

ABSTRACT

On the way to process automation, an important issue is the definition of the various activities or work tasks within the respective business process. Design decisions on this issue considerably affect business performance. Several guidelines known in the area of workflow management exist, but do not give the inexperienced workflow designer much to hold on to. This chapter introduces a cohesion metric that can be used for the identification of weakly cohesive activities in a workflow design. Also, a heuristic is presented that is based on this cohesion metric to decide between various workflow design alternatives. A theoretical and an empirical evaluation are included in this chapter, both positively supporting the soundness of the metric. The inspiration for the introduced notion is derived from similar cohesion metrics in software engineering.

INTRODUCTION

Workflow management projects typically start with the design of a business process. This usually results in a model of the process as a network of related activities. After the design phase, a formal version of the model can be used to configure a workflow management system. One of the functions of such a system is that it can allocate activity instances to the workers in that process at run-time (Jablonski & Bussler, 1996; Van der Aalst & Van Hee, 2002). However central the activity concept may be within such a setting, it is the author's experience in various workflow projects (De Crom & Reijers, 2001; Reijers, 2003) that the knowledge of identifying activities within a business process is limited and can result in ill-defined activities.

The results of ill-defined activities on the operational performance of a process may be substantial. One may think of activities that are needlessly small. This increases the number of hand-offs between activities, with a corresponding increase of errors (Seidmann & Sundararajan, 1997). Activities that are too large may cause inflexibility within a business process, since its underlying operations must be performed regardless of their merits under the circumstances (Van der Aalst, 2000).

The aim of this chapter is to provide some tangible guidance for activity definition in the form of a heuristic, by making the intuitively appealing notion of a 'logical unit of work' operational. The application area is the design of workflow processes. The heuristic we propose is based upon a cohesion metric for activities, as inspired by similar notions in software engineering. In this way, insights from computer science are transferred to the business area, which in this case seems to be a successful undertaking.

The structure of this chapter is as follows. First, we will introduce some basic concepts and give a short overview of existing activity definition heuristics in the workflow management field. Next, we will present the cohesion notion we mentioned earlier, as well as a heuristic for its use. After its introduction and the presentation of some examples, we will subject the cohesion notion to both a theoretical and empirical evaluation. Some concluding remarks and directions for further research form the final part of this chapter.

ACTIVITY DESIGN IN WORKFLOW MANAGEMENT

Terminology

An activity is a specification of a part of work to be accomplished. We use 'workflow process' as a synonym for a specific type of business process. A business process itself is a conceptual way of organizing work and resources by distinguishing a set of related activities. Workflow processes are usually found in administrative contexts (e.g., banking, insurance, government, etc.). They are particularly suitable to be supported by workflow management systems.

Each single activity that is distinguished within a workflow process may be divided into a number of operations. Operations are used to identify small parts of work in a way that is still useful within the business context. In general, it is also possible to distinguish an activity without mentioning the operations it comprises (non-determinism).

We interpret the matter of activity definition as the formulation of a goal and/or the assignment of operations to an activity within the context of a single workflow process. Part of the work in defining activities involves an evaluation of its properties, such as its size, its workability, its performance, etc. Although a broader view on an activity definition may also include matters such as the development of work instructions, various views and abstraction levels, supporting information systems, interfaces, etc., these are outside the scope of this chapter.

In choosing the respective terminology, we aspired consistency with the standards of the Workflow Management Coalition (Fischer, 2001).

Workflow Management

When taking the perspective of an organization, it immediately becomes clear that there are many factors that determine—or at least influence—how activities should be defined. One can think of regulations and considerations that emerge from human resource management, ergonomics, quality management, social sciences, accountancy, and various other fields.

One specific view is that of workflow management (WfM), which produces relatively pragmatic directions on activity definition, aimed at process automation. Sharp and McDermott (2001), for example, allow on each of the three to five hierarchic levels of a workflow process a number of five to seven activities. In other words, no matter what the operations are, they should be fitted in somewhere.

Van der Aalst and Van Hee (2002) note that to prevent problems in supporting processes by WfM systems it is necessary to only regard as an activity a logical unit of work (LUW). This means that the so-called ACID properties known from transaction processing apply: Atomicity, Consistency, Isolation, and Durability (Harder & Reuter, 1983). The authors also state their decomposition criterion for distinguishing a LUW: There must exist unity of time, place, and operation.

The unity of time, place, and operation criterion often does not act as an imperative, but rather as a first step for activity decomposition, according to Van der Aalst and Van Hee (2002). They give the following additional decomposition criteria:

- The recognizability of an activity by the members of the organization that must perform it is important, with a clear function and objective.
- Sensible interim states: All resulting interim states in the process caused by activity decomposition should be sensible.
- Acceptable "commit work" for each of the process activities: Violations of the ACID properties should be acceptable, especially with respect to possible rollback and the split up of tasks.

Van der Aalst and Van Hee (2002) admit that in practice it is not easy to address the ACID properties, because of the properties and limitations of current workflow systems. A partial solution to this problem is given by Grefen et al. (2001), who distinguish high-level (long-living) and low-level (relatively short-living) processes. The latter are subprocesses of the former, but both have different requirements. The low-level requires strict execution of the ACID properties. The high-level needs relaxation of the atomicity and isolation requirements. Their WIDE model for WfMS's supports this view (Grefen et al., 1999).

In conclusion, in the WfM field heuristics and rules of thumb are used to identify operations that more or less naturally belong together. Characteristically, decisions on splitting up or combining activities are context-sensitive. All the given rules provide a considerable degree of design freedom.

A COHESION METRIC FOR WORKFLOW ACTIVITIES

Introduction

In software engineering, manipulations (declarations, assignments, invocations, etc.) that are strongly related are preferably grouped together within the same module or class (Stevens et al., 1974). There are clear implications for the maintainability and re-usability of programs by using this approach and there is also considerable empirical evidence that the resulting computer programs contain fewer errors (Card et al., 1986; Selby & Basili, 1991).

Workflow processes are in some sense similar to computer programs, as they primarily involve *information processing steps*: checks, decisions, computations, copying, etc. Also, in workflow processes the administration of information is often vital (customer data, order information, etc.). Furthermore, workflow processes consist of activities, where computer programs are divided into modules or classes. And where modules contain statements and classes methods, activities consist of operations. For more information on the characterization of workflow processes, we refer the reader to Reijers (2003).

Because of the advantages of cohesion metrics in software design and the rough similarities between programs and workflow processes, it seems both fruitful and plausible to pursue something similar to a software cohesion metric for the sake of activity definition in workflow processes.

Activity Cohesion Metric

Prior to the formulation of the cohesion metric and the discussion of some examples, we introduce its supporting notions. For the sake of clarity, we do not consider an entire workflow process. Instead we directly zoom in on a part of the process for which it is unclear how to define activities, a so-called *operations structure*. We assume as given a sense of *operations* that take place within this part. Operations in our view can be seen as information functions, which take as *inputs* zero or more pieces of information and produce one new piece of information, its *output*. Consider, for example, a workflow that handles insurance claims. The decision whether or not a claim is acceptable to be processed could be given form in the following operation: Only when the claimant has not issued a claim earlier in the same year (input 1) and the damage is covered according to the policy (input 2), then the claim would be acceptable (output). Inputs 1 and 2 and the mentioned output can be seen as information elements being inspected and manipulated by executing operations.

The above view on the elementary operations is also used in the workflow design methodology Product-based Workflow Design (Reijers, 2003; Reijers et al., 2003), which is tried and tested in several cases (De Crom & Reijers, 2001; Reijers, 2003).

The formal definition of an operations structure is now as follows.

Definition 1. (Operations Structure). An operations structure is a tuple (D, O) with:

- D: the set of information elements that are being processed,
- $O = \{(p, cs) \in D \times \Pi(D)\}$ is a set of operations on the information elements, such that there are no 'dangling' information elements and no value of an information element depends on itself:

$R = \{(p, c) \in D \times D \mid \exists (p, cs) \in O : c \in cs\}$ is connected and acyclic.

Using the example introduced already, examples of data elements are 'claimant has not issued a claim earlier in the same year' (i1), 'damage is covered according to the policy' (i2), and 'claim is acceptable' (i3). An example of an operation is (i3, {i1, i2}). As said, the issue we consider is how to find a set of activities that partition the total set of operations. For a proper division, we introduce the notion of a *valid activity* and a *valid activity ordering*.

Definition 2. (Valid activity). Given an operations structure (D, O), any subset $t \subseteq O$ is a *valid activity* on the operations structure, or simply an *activity*.

Definition 3. (Valid activity ordering). Given an operations structure (D, O), the tuple (T, F) is a valid activity ordering on that operations structure if:

- T is a set of valid activities, $T \subseteq \Pi(O)$, such that:
  1. $\forall o \in O : (\exists t \in T : o \in t\}$.
- F is a partial ordering on F, $F \subseteq T \times T$, such that for each $t \in T$:
  2. $\forall t, u \in T : ((\exists (p, cs) \in t, (q, ds) \in u : q \in cs) \Rightarrow (u, t) \in F')$.

Within this definition it is expressed by 1 that all operations from the operation structure should appear at least once in one activity. This seems a reasonable requirement if we assume that in an earlier stage it has been decided that all remaining operations are essential to be performed. Condition 2 of Definition 3 enforces that when one operation depends on the output of another, then the respective tasks they are part of are ordered such that they respect this dependency. In other words, all information that is produced by an operation can only be consumed by a later activity.

The definition of our cohesion metric, then, depends on two important parts: the relation cohesion and the information cohesion. The relation cohesion gives a measure on how much the different operations within one activity are related. We will first give the formal definitions, after which we explain these notions with some examples.

Definition 4. (Activity relation cohesion). For a valid activity t on an operation structure (D, O), its relation cohesion $\lambda(t)$ is defined as follows:

$$\lambda(t) = \begin{cases} \dfrac{\sum_{(p,cs) \in t} \left| \{(q, ds) \in t \setminus \{(p, cs)\} \mid (\{p\} \cup cs) \cap (\{q\} \cup ds) \neq \varnothing\} \right|}{|t| \cdot |t - 1|}, \text{ for } |t| > 1 \\ 0, \text{ for } |t| \leq 1. \end{cases}$$

To compute the activity relation cohesion, for each operation it should be determined with how many other operations it overlaps, i.e., it shares an input or output. The *average* overlap per operation over all operations within an activity is then divided by the maximal overlap, i.e., the number of operations minus 1, to get a relative measure between 0 and 1.

The other component of our cohesion metric, the activity information cohesion, focuses on the sharing of information elements.

Definition 5. (Activity information cohesion). For a valid activity *t* on an operation structure (*D*, *O*), its information cohesion μ(*t*) is defined as follows:

$$\mu(t) = \begin{cases} \dfrac{\left|\left\{d \in D \mid \exists (p,cs),(q,ds) \in t : d \in \left(\{p\} \cup cs\right) \cap \left(\{q\} \cup ds\right) \wedge (p,cs) = (q,ds)\right\}\right|}{\left|\left\{d \in D \mid \exists (p,cs) \in t : d \in \left(\{p\} \cup cs\right)\right\}\right|}, \text{for } |t| > 0 \\ 0, \text{ for } |t| = 0 \end{cases}$$

The activity information cohesion focuses on all information elements that are used either as input or output by any operation within the respective activity. It determines how many information elements are used more than once in proportion to all the information elements used, which is a relative measure between 0 and 1.

The total cohesion of an activity is now given as the product of both the relation and information cohesion. An activity has to score high on both cohesion metrics to say it is cohesive in total. Clearly, an extreme score on one coefficient may outweigh a mediocre result on the other, which is seen as satisfactory.

Definition 6. (Activity cohesion). For a valid activity *t* on an operation structure (*D*, *O*), its (general) cohesion *c*(*t*) is defined as follows:

$$c(t) = \lambda(t) \cdot \mu(t)$$

In the following subsection, we will incorporate the cohesion metric in an overall strategy for its application and explore two examples where we apply the introduced cohesion metrics.

Application of the Cohesion Metric

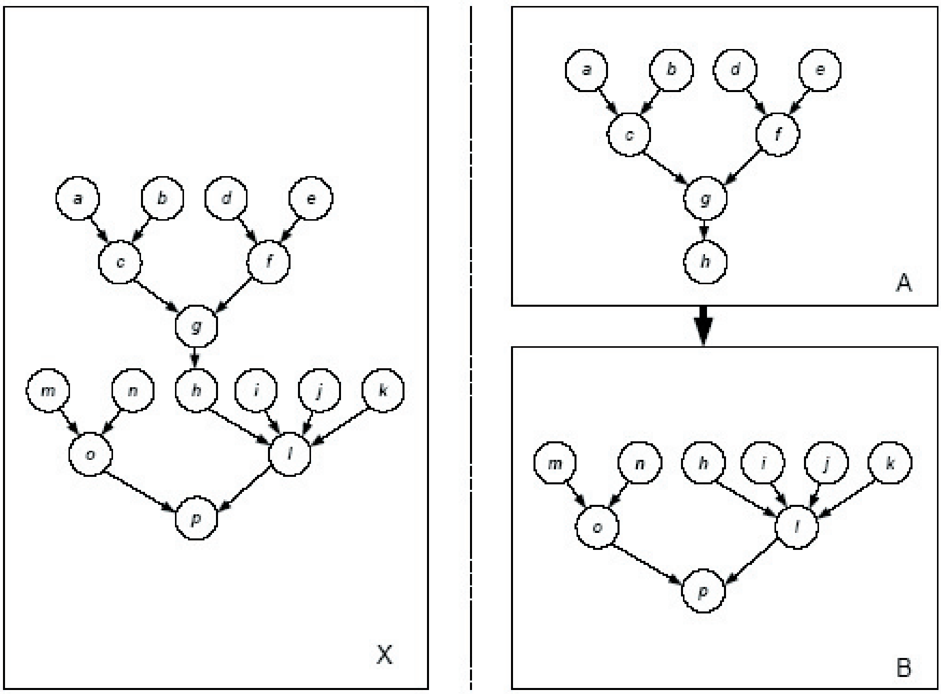
Let us assume that there is an activity X, which is relatively incohesive in an overall workflow design on the basis of the presented cohesion metric. It is subsequently considered to be split up into validly ordered activities A and B. An evaluation that could take place on the basis of the same activity cohesion is then as follows:

1. Determine the cohesion of A and B (the cohesion of X is already known).
2. If both cohesion coefficients of A and B are higher than that of X, then the division into A and B is preferable.
3. If the cohesion coefficient of X is higher than both cohesion coefficients of A and B, then the larger activity X is preferable.
4. In all other cases, the heuristic is indecisive.

Note that our heuristic *does not describe how the candidates A and B can be determined*. Obviously, in small enough cases it is feasible to generate a great number of partitions, but it grows exponentially in the number of operations that are considered. Also note that a similar approach could be taken when activity X is considered to be integrated with another activity Y, resulting in activity C.

Consider the example in Figure 1. We totally abstract at this place from the content of the information elements and operations within the operations structure. The example is

Figure 1



based on the operations structure (D_1, O_1) with $D_1 = \{a, b, \ldots, p\}$ and $O_1 = \{(c, \{a, b\}), (f, \{d, e\}), (g, \{c, f\}), (h, \{g\}), (p, \{m, n\}), (l, \{h, i, j, k\}), (o, \{m, n\}), (p, \{o, l\})$. It represents two alternatives. Alternative X consists of one activity that comprises all these operations. The other alternative consists of a valid ordering of activities A and B. Note that an arrow leading from one information element to another signifies that the former is needed as an input for the other in some operation. Also note that the ordering of activities A and B is valid, in accordance with Definition 3.

If we consider operation $(c, \{a, b\})$ we see that it has a relation with just one other operation, namely $(g, \{c, f\})$. After all, output c of the former operation is an input of the latter. On average for activity X, each of its operations has a relation with 12/7 other relations, being the quotient of the summed number of relations over all operations and the total number of relations. The maximum number of pair-wise relations that any element within a set of 7 could have equals 6. Therefore, the relation cohesion of activity X equals 12/(7*6) = 2/7.

Furthermore, there are 6 information elements—c, f, g, h, o, and p—that are shared among several operations within X. The rest of the 16 elements are in use by exactly one operation at a time. Therefore, the information cohesion of activity X equals 6/16. The total cohesion of activity X is the product of its relation and information cohesion: 2/7 * 6/16 = 12/112. Similarly, we can compute the cohesion coefficients of activities A and B. The results of this exercise are given in Table 1.

If we apply our heuristic to this example, then the division of the operations structure into A and B is preferable over the single activity X. This appeals to the intuition that activity X is divided into two parts that are only related to each other through operation (h, {g}).

Table 1: Cohesion Coefficients for Figure 1

	X	**A**	**B**
relation cohesion	2/7	5/12	2/3
information cohesion	6/16	3/8	2/9
total cohesion	12/112 (≈ 0.1)	15/96 (≈ 0.2)	4/27 (≈ 0.2)

Somebody who is to perform this task may easily wonder what the processing of *a*, *b*, *c*, *d*, *e*, *f* has to do with the processing of *m*, *n*, *i*, *j*, *k*. Obviously, we cannot say anything at this point about, e.g., the semantical resemblances between the various operations.

To appreciate the heuristic's opposite discrimination consider the example as given in Figure 2, based on the operations structure (D_2, O_2) with $D_2 = \{a, b, ..., i\}$ and $O_2 = \{(a, \{g, h\}), (d, \{i\}), (c, \{a, b\}), (e, \{b, d\}), (f, \{c, e\})\}$. It again represents two alternatives in the fashion of the first example. The various cohesion coefficients are given in Table 2.

Perhaps the reader may be under the impression that the relation cohesion and information cohesion coefficients are highly related. However, it is possible to think of various activities with very different distributions of the respective metric. It is, for example, possible to think of an activity with one type of cohesion that equals 1, while the other almost equals 0. Because of limitations of space, actual examples are not given here.

Figure 2

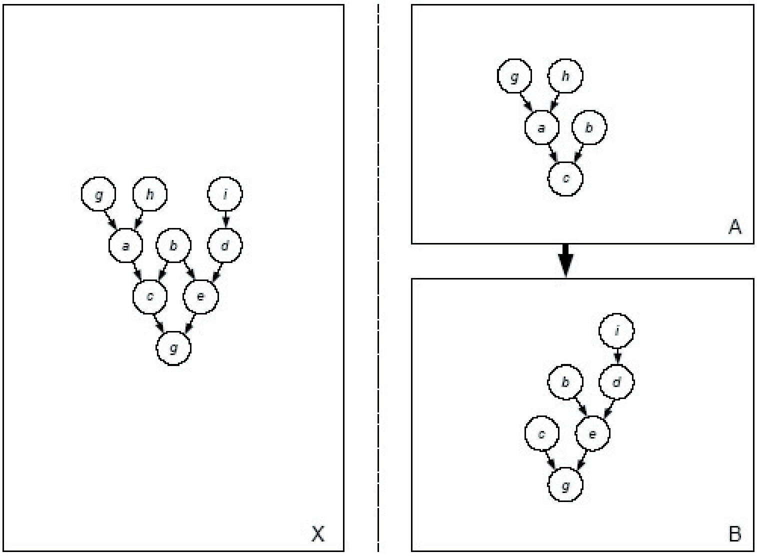


Table 2: Cohesion Coefficients for Figure 2

	X	A	B
relation cohesion	1/2	1	2/3
information cohesion	5/9	1/5	1/3
total cohesion	5/18 (≈ 0.3)	1/5 (= 0.2)	2/9 (≈ 0.2)

EVALUATION

Theoretical Perspective

One could wonder if the introduced cohesion metric has a sound theoretical basis. For the sake of comparison: In computing science many metrics exist to express the quality of some piece of software, but many of these metrics have been subject to wide criticism because of the lack of such a theoretical basis (e.g., Vessey & Weber, 1984).

To the author's knowledge, design quality criteria are lacking so far in the area of workflow design, aside from *correctness* notions (e.g., Aalst, 1998). Therefore, we consider a set of principles as defined by Chidamber and Kemerer (1994), which in turn have been derived from Weyuker (1988). These principles specifically aim at providing support when specifying cohesion metrics in objected oriented software design. Although the differences between this area and that of process design are clear, we refer to it in lack of more specific support.

The principles of Chidamber and Kemerer (1994) are defined on classes, which in Object-Oriented Design (OOD) can be seen as abstractions of the problem space. They are the following:

1. *Noncoarseness*: Given a class P and a metric μ, another class Q can always be found such that $\mu(P) \neq \mu(Q)$. This implies that not every class can have the same value for a metric; otherwise it has lost its value as a measurement.
2. *Nonuniqueness (Notion of Equivalence)*: There can exist distinct classes P and Q such that $\mu(P) = \mu(Q)$. This implies that two classes can have the same metric value, i.e., the two classes are equally complex.
3. *Design Details are Important*: Given two class designs, P and Q, which provide the same functionality, this does not imply that $\mu(P) = \mu(Q)$. The specifics of the class must influence the metric value. The intuition behind the property is that even though two class designs perform the same function, the details of the design matter in determining the metric for the class.
4. *Monotonicity*: For all classes P and Q, the following must hold $\mu(P) \leq \mu(P+Q)$ and $\mu(Q) \leq \mu(P+Q)$ where $P+Q$ implies combination of P and Q. This implies that the metric for the combination of two classes can never be less than the metric for either of the component classes.
5. *Nonequivalence of Interaction*: $\exists P, \exists Q, \exists R$, such that $\mu(P) = \mu(Q)$ does not imply that $\mu(P+R) = \mu(Q+R)$. This suggests that interaction between P and R can be different from interaction between Q and R, resulting in different complexity values for $P + R$ and $Q + R$.

6. *Interaction increases Complexity*: $\exists P$, $\exists Q$, such that $\mu(P) + \mu(Q) < \mu(P+Q)$. The principle behind this property is that when two classes are combined, the interaction between classes can increase the complexity metric value.

We will now consider each of these requirements on our cohesion metric, interchanging the class notion for that of an activity. For this evaluation, we will assume that in general an operations structure will contain a non-trivial number of operations which are not totally unrelated (i.e., there are at least two operations that share at least one information element).

Noncoarseness

With respect to the first property, noncoarseness, suppose that there is an activity t defined on a certain operations structure. If $c(t)$ equals zero, it is always possible to add two related operations such that $c(t)$ increases. If $c(t)$ is unequal to zero, it is possible to take operations away from it until its cohesion equals zero. Therefore, noncoarseness is guaranteed on theoretical grounds. More practically, taking away or adding an operation will *mostly* affect both the relation cohesion and the information cohesion of that activity.

Nonuniqueness

If we consider the second property, nonuniqueness, it is immediately clear that each activity with only one operation—regardless which one—has a cohesion of 0. So on theoretical grounds this property is satisfied. Practically, it will often be possible to define two activities with the same cohesion if an operations structure is sufficiently large. For instance, it will then be possible to find two sets (activities) of two operations each, such that both sets involve an equal number of information elements of which only one information element is shared among the two operations it consists of. Both these activities will exactly have the same cohesion.

Design Details are Important

An evaluation of the third property, design details are important, requires us to evaluate the notion of functionality within the context of process design. We assume that two activities are functionally the same when they share the same outputs, i.e., they contain operations with equal outputs (which are *not* used as inputs by other operations within these activities). In practical operations structures it will often be the case that there are two operations with the same output, but with different inputs, e.g., determining somebody's suitability for a job by an interview or by means of a psychological test. The specific choice for one of the alternatives to include in an activity will very likely be of influence on its cohesion, satisfying the property in question.

Monotonicity

The fourth property of monotonicity is in general *not* satisfied by our notion of cohesion. There is a very good explanation for this: The explicit intention of the cohesion metric is to decide whether it is wise to combine activities or not. If cohesion would have been a monotonic property, combining activities *always results* in a higher cohesion. This would have made the criterion worthless for our purpose. The original thought behind this monotonicity property may be inspired by such simple complexity metrics as 'number of lines in code' and 'number of methods in a class'. It is interesting to note that this property is neither satisfied by the suite of cohesion metrics that Chidamber and Kemerer (1994) themselves propose.

Nonequivalence of Interaction

It is easy to see that the fifth property, nonequivalence of interaction, will be satisfied in most practical cases. We only have to consider two different activities with equal cohesion and an operation that is related to one or more activities within the first, but to none of the activities in the other activity. In adding the operation to both activities, the cohesion may respectively increase in the first case and will *always* decrease in the second.

Interaction Increases Complexity

Finally, the sixth property, interaction increases complexity, can be satisfied by choosing any two operations from an operations structure that share an input or output. Two separate activities with only one of these operations will have a cohesion that equals zero; one activity that includes both operations will have a positive cohesion, so that the property is satisfied. Although it will be not very common that $\mu(P) + \mu(Q) < \mu(P+Q)$ in a practical case, our criterion is specifically intended to identify cases where $\mu(P) < \mu(P+Q)$ and $\mu(Q) < \mu(P+Q)$. After all, in this situation we would prefer to combine P and Q (see example 2).

Discussion

On the basis of the above evaluation, the cohesion metric we presented could be said to satisfy all relevant theoretical requirements of Chidamber and Kemerer (1994). Obviously, there are many reservations to be made, for example the absence of more context-specific theoretical requirements. As this is really a best effort, it is interesting to take a look at an empirical evaluation of the heuristic as well.

EMPIRICAL PERSPECTIVE

Web-Based Survey

The approach we followed for the empirical evaluation is to test the heuristic by applying it to a set of design dilemmas and compare its outcomes to the judgment of human experts. For this purpose, we used a digital web-based survey, which contained ten design dilemmas in the same spirit as the examples of Figures 1 and 2. A respondent must choose for each of the dilemmas on a three-point Likert scale whether he or she:

- prefers to combine the operations in one large activity,
- has no preference for combining or splitting up these activities, or
- supports the split-up of the same operations in the two *given* activities.

The respondent is instructed to follow his intuition whether the operations as depicted seem to "belong together" or not. The only thing that is explained to the respondent is the meaning of the used symbols in each figure and the context of workflow design. A screenshot of one of the presented dilemmas in the web-based survey is given in Figure 3.

Fifteen Dutch workflow designers working for management consulting companies, one large bank, and one utilities provider, were asked to cooperate in this survey. Fourteen of them actually responded before the deadline. The average number of workflow design projects they participated in ranged from two to 25, with an average of 10.

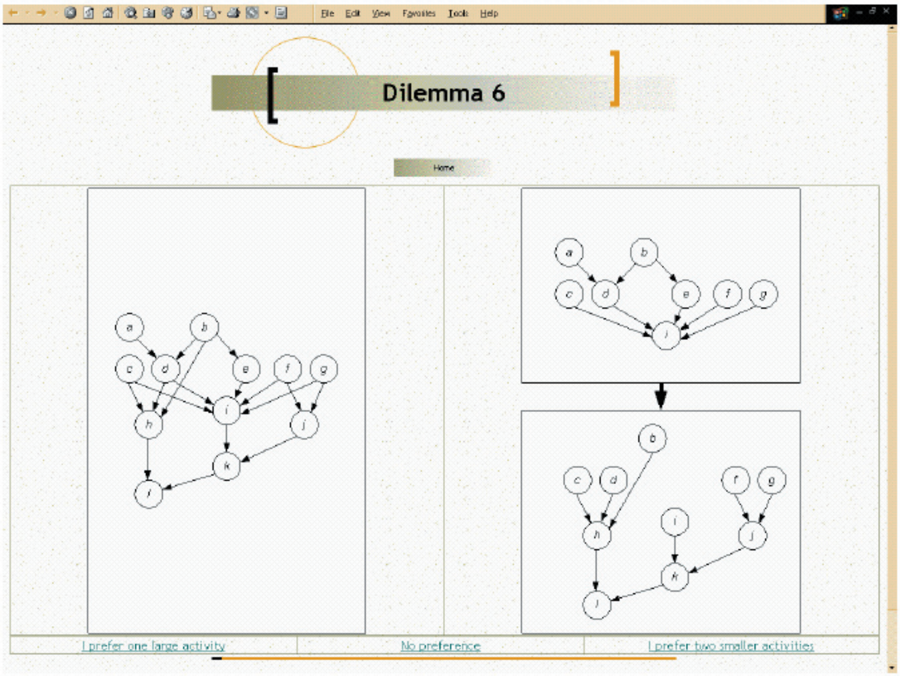
For the analysis of the results we used statistical methods for *rater agreement*, as often used in medical settings to compare experts' opinions on the same data. For more background on rater agreement statistics, see Uebersax (1992, 2001). We computed the Pearson correlation between the respondents' average score and the heuristic outcome. In Table 3, we have represented the outcomes of the web survey for each of the respondents (R1…R14) for each of the dilemmas (D1…D10). The average score for each dilemma is given in the column labeled 'R_avg'; the heuristic's outcome column with 'H'.

The correlation between the average respondent score and the heuristic outcome approximately turned out to be 0.810. This is a significant result, assuming a two-tailed 99% confidence interval.

In addition to this first analysis, we examined the relation between the average respondent score and each individual correspondent (**Corr R_avg**), as well as the relation between the heuristic outcome and each individual respondent (**Corr H**). The respective correlation coefficients are shown in Table 4.

In addition to this analysis, we interviewed respondents R2, R5, R7, R11 and R13 the day following their survey. Of particular interest was the opinion of respondent R11. He explained that in general he is in favor of splitting up activities in a workflow design in the smallest possible parts. His consideration is that at run-time execution of a workflow process, activities may be dynamically combined to be allocated to workers if this looks like a good idea under the circumstances. Taking a look at the results in Table 3 for respondent R11,

Figure 3: Screenshot of the web-survey


File Edit View Favorites Tools Help
Dilemma 6
Home
a
b
c
d
e
f
g
h
i
j
k
l
I prefer one large activity
No preference
I prefer two smaller activities

this explains the almost monotonic choice for splitting up activities. Correspondent R7 had different considerations, but could also explicitly support his deviating score.

Correspondents R2, R5 and R13—of whom the individual scores significantly corresponded with our cohesion metric—were much less outspoken about their considerations. They considered the dilemmas one by one, without a general design motive. Correspondent R5 admitted that she was highly intrigued by the relation between her opinion and the cohesion metric, while at the same time she could not explicitly support most of her decisions in retrospect.

Discussion

From the second part of the analysis it follows that the opinion of each individual correspondent reasonably well corresponds with the group's average (**Corr R_avg**): The opinion of 11 out of 14 respondents significantly corresponds with this average; for half of the correspondents this significance is high. This gives us some reassurance that comparing the average respondent's score is a good measure for reflecting the group's opinion. Combined

Table 3: Data and Analysis of the Web Survey (1 = combine, 2 = combine/split, 3 = split)

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**	**R_avg**	**H**
D1	3	3	3	3	3	3	1	3	2	3	3	3	3	3	2.8	3
D2	1	1	1	1	1	1	1	1	1	1	3	1	1	1	1.1	1
D3	3	3	2	3	3	3	2	3	1	3	3	2	3	1	2.5	3
D4	3	3	3	2	3	3	3	3	2	3	3	1	3	3	2.7	3
D5	1	1	1	1	1	1	1	3	1	1	3	1	1	1	1.3	1
D6	1	1	1	1	1	1	3	1	1	1	3	2	1	3	1.5	1
D7	3	3	3	3	3	3	1	3	1	3	3	2	1	3	2.5	2
D8	3	1	3	1	3	3	1	1	1	3	3	1	3	1	2.0	3
D9	3	1	3	2	3	3	1	3	2	3	1	2	1	1	2.1	3
D10	1	1	1	1	1	2	1	3	1	1	3	1	1	1	1.4	2

*Table 4: Additional Analysis of the WebSurvey (*** = sign. 99 %, ** = sign. 95 %, * = sign. 90 %)*

	R1	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**
Corr R_avg	0.912 (***)	0.877 (***)	0.855 (***)	0.862 (***)	0.912 (***)	0.875 (***)	0.209
Corr H	0.890 (***)	0.515 (*)	0.827 (***)	0.579 (**)	0.890 (***)	0.943 (***)	0.000

	R8	**R9**	**R10**	**R11**	**R12**	**R13**	**R14**
Corr R_avg	0.482	0.592 (**)	0.912 (***)	-0.048	0.565 (*)	0.706 (**)	0.534 (*)
Corr H	0.401	0.601 (**)	0.890 (***)	-0.306	0.311	0.749 (***)	0.047

with the highly significant correlation between the heuristic and the group's average opinion of 0.810, we cautiously conclude *a positive relation between our cohesion metric and the corresponding intuition of experts on this matter*. Obviously, the set of respondents and questions is very small. Furthermore, we have no hard evidence that the presented dilemmas look like real practical problems; it is based on the author's personal experiences only.

The considerations of the respondents give us some insight into the limits of a cohesion metric like the one we defined. When a design consideration is very specific, the cohesion metric may be a bad implementation. However, when these considerations are less explicit or mixed, then the cohesion metric seems like an attractive and valid quantification thereof. An expert group's opinion is then reasonably well reflected by the metric.

CONCLUSIONS

On the basis of our evaluation, the author is positive about the value of the cohesion metric for both distinguishing weakly cohesive activities and the support it can offer to decide between design alternatives. Obviously, these results must be interpreted with caution, as discussed earlier. In particular, the interviews with workflow designers showed that very specific design considerations are not well implemented by the cohesion metric. The presented cohesion metric should be used *when more explicit considerations* do not lead to a decisive result. In this sense, it can be used as the 'finishing touch' for a workflow design.

The possibilities to extend this research are many. A following step may be the use of the cohesion metric in question in an actual project, which involves the design of a workflow process in a real setting. Several of the respondents have indicated their willingness to cooperate within such a practical test. It would be a good opportunity to test the heuristic on real design dilemmas.

As stated before, the introduced cohesion metric only *supports* the designer in making a decision with respect to activity definition. It does not suggest any clustering or ordering itself. An extension of the heuristic so that it efficiently generates optimal activity definitions itself is the ultimate but challenging next step of this research. One interesting idea is to evaluate the cohesion of activities in existing workflow design which are considered by their owners as ‘well-designed’. This could result in an accumulation of empirical metrics scores on cohesive activities. These particular scores could then be used as a sort of ‘stop criterion’ when generating alternative workflow designs: As soon as all activities satisfy a minimal, empirically determined quality score, the search for yet more alternatives could be terminated. The gathering of these empirical data and experimentation with such a quality score must point out whether this approach sufficiently prunes the search tree for alternative workflow designs, which grows exponentially in the number of available operations.

Finally, we would like to extend the cohesion metric as described with notions for the ‘coupling’ degree between several activities. In software engineering, this is another important construct. It gives an indication how modules or classes incorporate a sense of mutual independence. The higher the exchange of calls and information exchange between modules or classes, the lower their independence. Clearly, the notions of coupling and cohesion are related to some level. We suspect that the translation of the concept of coupling to workflow processes may be less straightforward than it was the case for cohesion. After all, the drawbacks of highly dependent activities seem less severe than tightly coupled software modules.

ACKNOWLEDGMENTS

The author would like to thank Eric Verbeek of the Technische Universiteit Eindhoven for his assistance in preparing and carrying out the web-based survey. The author is also indebted to all consultant respondents.

REFERENCES

Aalst, W.M.P. van der. (2000). The application of Petri nets to workflow management. *The Journal of Circuits, Systems and Computers, 8*(1), 21-66.

Aalst, W.M.P. van der. (2000). Reengineering knock-out processes. *Decision Support Systems, 30*(4), 451-468.

Aalst, W.M.P. van der, & Hee, K.M. van. (2002). *Workflow management: Models, methods, and systems.* Cambridge: MIT Press.

Chidamber, S.R., & Kemerer, C.F. (1994). A metrics suite for object oriented design. *IEEE Transactions on Software Engineering, 20*(6), 476-493.

Crom, P.J.N. de, & Reijers, H.A. (2001). Using prototyping in a product-driven design of business processes. In A. D’Atri, A. Solvberg, & L. Wilcocks, (Eds.), *Proceedings of the Open Enterprise Solutions: Systems, Experiences, and Organizations Conference* (pp. 41-47). Rome: Luiss Edizioni.

Fischer, L. (Ed.). (2001). *Workflow handbook 2001*. Lighthouse Point: Future Strategies.

Grefen, P., Pernici, B., & Sanchez, G. (Eds.). (1999). *Database support for workflow management: the Wide Project.* Dordrecht: Kluwer.

Grefen, P., Vonk, J., & Apers, P. (2001). Global transaction support for workflow management systems: from formal specification to practical implementation. *VLDB Journal, 10*(4), 316-333.

Harder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. *Computing Surveys, 15*(4), 287-317.

Jablonski, S., & Bussler, C. (1996). *Workflow management: Modeling concepts, architecture, and implementation.* London: International Thomson Computer Press.

Reijers, H.A. (2003). *Design and control of workflow processes: Business process management for the service industry.* Berlin: Springer Verlag.

Reijers, H.A., Limam, S., & Aalst, W.M.P. van der. (2003). *Product-based workflow design. Journal of Management Information Systems, 20*(1), 229-262.

Seidmann, A., & Sundararajan, A. (1997). The effects of task and information asymmetry on business process redesign. *International Journal of Production Economics, 50*(2-3), 117-128.

Selby, R.W., & Basili, V.R. (1991). Analyzing error-prone system structure. *IEEE Transactions on Software Engineering, 17*(2), 141-152.

Sharp, A., & McDermott, P. (2001). *Workflow modeling: Tools for process improvement and application development*. Boston: Artech House Publishers.

Stevens, W., Myers, G., & Constantine, L. (1974). Structured design. *IBM Systems Journal, 13*(2), 115-139.

Uebersax, J. S. (1992). A review of modeling approaches for the analysis of observer agreement. *Investigative Radiology, 27*, 738-743.

Uebersax, J. S. (2001). *Statistical methods for rater agreement.* [Online]. Available: http://ourworld.compuserve.com/homepages/jsuebersax/agree.htm

Vessey, I., & Weber, R. (1984). Research on structured programming: an empiricist's evaluation. *IEEE Transactions on Software Engineering, 10*(4), 394-407.

Weyuker, E. (1988). Evaluating software complexity measures. *IEEE Transactions on Software Engineering, 14*(9), 1357-1365.

Chapter VI

Fuzzy Aggregations and Fuzzy Specializations in Fuzzy EER Model

Jóse Galindo, Universidad de Málaga, Spain

Angélica Urrutia, Universidad Católica de Maule, Chile

Mario Piattini, Universidad de Castilla-La Mancha, Spain

ABSTRACT

Some approaches about fuzzy ER/EER model have been published recently. Few of these works study how to relax constraints and other aspects expressed in the model. In this chapter our aim is to relax some semantic aspects which have not been studied in previous works and to extend the EER model with fuzzy capabilities. We use fuzzy quantifiers and fuzzy degrees which have been widely studied in the context of fuzzy sets and fuzzy query systems for databases. We will also examine the representation of these new features in an EER model and their practical repercussions. The studied extensions are: fuzzy aggregations and fuzzy aspects on specializations, such as fuzzy degrees, fuzzy completeness constraint, fuzzy cardinality constraint on overlapping specializations, fuzzy disjointed or overlapping constraints, fuzzy attribute defined specializations, fuzzy constraints in union types or categories and fuzzy constraints in shared subclasses (or intersection types). All these fuzzy extensions have a new meaning and offer greater expressiveness in conceptual design.

INTRODUCTION

Conceptual modeling or design is a fundamental phase in the design of any database (Elmasri & Navathe, 2000). In this phase of conceptual design the aim is to obtain the so-called conceptual schema, which is a concise description of the data required by users, including detailed description of the types of entities involved, the interrelationships existing between them and also some important constraints in these relationships.

The Enhanced Entity-Relationship Model (EER) (Connolly & Begg, 1998; Elmasri et al., 2000; Hammer & McLeod, 1981) is an extension of the Entity-Relationship Model (ER) (Chen, 1976). This study is based on the EER model published in Connolly et al. (1998), and Elmasri et al. (1985, 2000), which is one of the most modern, versatile and complete versions.

Fuzzy databases (Galindo, 1999; Medina et al., 1994; Petry, 1996) have also been widely studied with scant attention being paid to the problem of conceptual modeling. At the same time, the extension of the ER model for the treatment of fuzzy data (with vagueness) has been studied in some publications (Chaudhry et al., 1994, 1999; Chen & Kerre, 1998; Ma et al., 2001; Zvieli & Chen, 1986), but none of them refer to the possibility of expressing constraints flexibly by using the tools offered by fuzzy sets theory. Besides, these approaches are not exhaustive in other senses. Perhaps the most exhaustive fuzzy modeling tool is the FuzzyEER model (Galindo et al., 2001b, 2003, 2004; Urrutia et al., 2002a, 2002b, 2003) and in this chapter we expose some of its advantages.

Zvieli and Chen (1986) allow fuzzy attributes in entities and relationships and they introduced three levels of fuzziness in the ER model:

1. At the first level, entity sets, relationships and attribute sets may be fuzzy; namely, they have a membership degree to the model. For example, the fuzzy entity Radio may have a 0.7 importance degree as an integrating part of a car.
2. The second level is related to the fuzzy occurrences of entities and relationships. For example, an entity Young Employees must be fuzzy, because its instances, its employees, belong to the entity with different membership degrees.
3. The third level concerns the fuzzy values of attributes of special entities and relationships. For example, attribute Quality of a basketball player may be fuzzy.

The first level may be useful, but at the end we must decide whether such an entity, relationship or attribute will appear or will not appear in the implementation. The second level is useful too, but it is important to consider different degree meanings (membership degree, importance degree, fulfillment degree...). A list of authors using different meanings is included in Galindo et al. (2001a). The third level is useful, but it is similar to writing the data type of some attributes, because fuzzy values belong to fuzzy data types.

Chaudhry et al. (1994; 1999) propose a method for designing Fuzzy Relational Databases (FRDB) following the extension of the ER model of Zvieli et al. (1986), taking special interest in converting crisp databases into fuzzy ones. The way to do so is to define linguistic labels as fuzzy sets and to obtain the membership degree to each of them of the crisp value existing in the database.

In 1998, Chen et al. introduced the fuzzy extension of several major EER concepts (superclass, subclass, generalization, specialization, category and subclass with multiple superclasses) without including graphical representations. The proposal of Vert et al. (2000) is based on the notation used by Oracle and uses fuzzy sets theory to treat data sets as a collection of fuzzy objects, applying the result to the area of Geospatial Information Systems (GIS).

Finally, Ma et al. (2001) work with the three levels of Zvieli and Chen (1986) and they introduce a Fuzzy Extended Entity-Relationship (FEER) model to cope with imperfect as well as complex objects in the real world at a conceptual level. However, their definitions (of generalization, specialization, category and aggregation) impose very restrictive conditions. In addition, they also provided an approach to mapping an FEER model to a fuzzy object-oriented database schema.

All these works do not study how to relax the constraints expressed in the ER/EER model so that they can be made more flexible, because the constraints of the traditional model are either too restrictive or too permissive. Perhaps the first work relaxing constraint was recently published in Galindo et al. (2001b), studying, for example, fuzzy participation constraint and fuzzy cardinality constraint in a relationship.

Firstly, we will summarize the basic concepts of fuzzy logic, paying particular attention to fuzzy quantifiers. After it, we formalize how we will use fuzzy concepts in the following sections. Then, we will study each of the FuzzyEER extensions separately: fuzzy aggregations, fuzzy degrees in specializations, fuzzy completeness constraint on specializations, fuzzy cardinality constraint on overlapping specializations, fuzzy disjointed or overlapping constraints on specializations, fuzzy attribute defined specializations, fuzzy constraints in union types or categories and fuzzy constraints in shared subclasses. Finally, we outline some conclusions and suggest some research lines for the future.

FUZZY SETS: FUZZY QUANTIFIERS

In 1965, Lotfi A. Zadeh defined the concept of fuzzy set based on the idea that there are sets in which it is not totally clear whether an element belongs to the set or not. Sometimes an element belongs to the set to a certain degree, which is called membership degree. For example, the set of tall people is a fuzzy set because there is no height limit establishing the minimum height for a person to be considered tall.

A fuzzy set A is defined as a function of belonging μ_A which connects or pairs up the elements of a domain or discourse Universe U with elements of the interval [0,1]:

$$\mu_A (u): U \rightarrow [0, 1] \qquad (1)$$

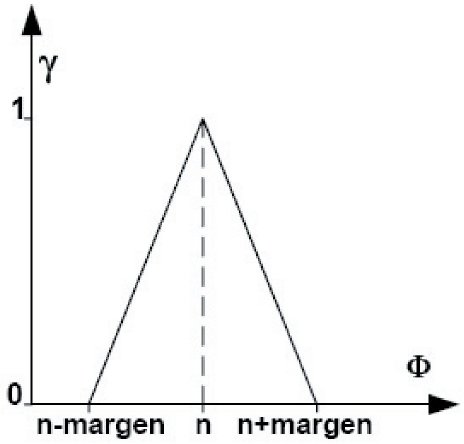
The closer μ_A(u) to the value 1, the greater the membership of the object $u \in U$ to the fuzzy set A. The values of membership vary between 0 (does not belong at all) and 1 (total belonging). A fuzzy set A can be represented as a set of pairs of values: each element u with its membership degree μ_A(u):

$$A = \{\mu_A (u) /u : u \in U\} \qquad (2)$$

A fuzzy number is a fuzzy set, where U is a numerical domain (normally the real numbers R). Figure 1 shows the membership function of the fuzzy number “Approximately n”. The margin value m indicates the limits of the fuzzy set. It is easy to observe that the nearer a number is to the value n, the greater its membership to “approximately n”. U is called “underlying domain” of the fuzzy set. The underlying domain may be ordered or non-ordered, and continuous or non-continuous (discrete).

From this simple concept a complete mathematical and computing theory has been developed which facilitates the solution of certain problems (Pedrycz et al., 1998). Fuzzy logic has been applied to a multitude of objectives such as: control systems, modeling, simulation, patterns recognition, information or knowledge systems (databases, expert systems...), computer vision, artificial intelligence, artificial life....

Figure 1: Function "Approximately n" (n(+-)m, where m is a margin)



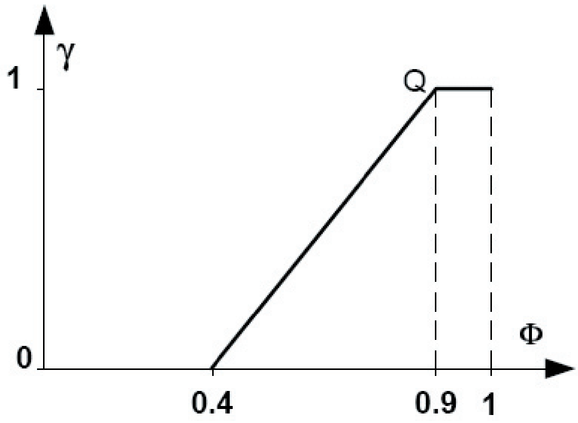
Fuzzy Quantifiers

Fuzzy or linguistic quantifiers (Galindo, 1999; Galindo et al., 2001a, Yager, 1983; Zadeh, 1983) allow us to express fuzzy quantities or proportions in order to provide an approximate idea of the number of elements of a subset (or fulfilling a certain condition) or of the proportion of this number in relation to the total of possible elements. Fuzzy quantifiers can be absolute or relative:

- **Absolute quantifiers** express quantities over the total number of elements of a particular set, stating whether this number is, for example, "large", "small", "many", "few", "very many".... Generalizing this concept, we can consider fuzzy numbers as absolute fuzzy quantifiers, in order to use expressions like "approximately between 5 and 10", "approximately_8".... Note that the expressed value may be positive or negative.
  In this case, we observe that the truth of the quantifier depends on a single quantity. For this reason, the definition of absolute fuzzy quantifiers is, as we will see, very similar to that of fuzzy numbers.
- **Relative quantifiers** express measurements over the total number of elements which fulfil a certain condition depending on the total of possible elements, so that the truth of the quantifier depends on two quantities. This type of quantifiers is used in expressions like "the majority", "the minority", "approximately half"....
  In this case, to evaluate the truth of the quantifier we need to find the total quantity of elements which fulfil the condition and consider this value with respect to the total quantity of elements which could fulfil it (including those which fulfil it and those which do not fulfil it).

In Zadeh (1983), absolute fuzzy quantifiers are defined as fuzzy sets in the interval $[0, +\infty)$ and relative quantifiers as fuzzy sets in the interval $[0,1]$. We have extended the definition of absolute fuzzy quantifiers to the interval $(-\infty,+\infty)$. That is to say that a quantifier Q is represented as a function Q whose domain depends on whether it is absolute or relative:

Figure 2: Relative fuzzy quantifier "almost all": Φ [0.4, 0.9] ↔ Q(Φ) = 2(Φ - 0.4)



$$\begin{array}{l} Q_{\text{abs}} : R \rightarrow [0, 1] \\ Q_{\text{rel}} : [0, 1] \rightarrow [0, 1] \end{array} \qquad (3)$$

where the domain of Q_{rel} is [0,1] because the division $a/b \in [0,1]$, where a is the number of elements fulfilling a certain condition and b is the total number of elements in existence.

In order to know the fulfillment degree of the quantifier over the elements which fulfil a certain condition, we apply the function Q of the quantifier to the value of quantification Φ:

$$\Phi = \begin{cases} a & \text{si } Q = Q_{\text{abs}} \\ a/b & \text{si } Q = Q_{\text{rel}} \end{cases} \qquad (4)$$

Thus, the fulfillment degree is $Q(\Phi)$. If $Q(\Phi)=1$, it indicates that this quantifier is completely satisfied. The value 0 indicates, on the other hand, that the quantifier is not fulfilled at all. Any intermediate value indicates an intermediate fulfillment degree for the quantifier.

Example 1. "Approximately_8" is an absolute fuzzy quantifier, defined as shown in Figure 1, with n=8, and the margin m=3, for example. "Almost_all" is a relative fuzzy quantifier, defined as shown in Figure 2.

THRESHOLDS, FUZZY QUANTIFIERS AND DEGREES FOR OUR APPLICATION

Applied in the context of databases, the usefulness of fuzzy quantifiers is shown by the flexibility it offers to carry out queries which involve these quantifiers, as for example in the division operation of relational algebra in fuzzy or classical databases (Galindo et al., 2001a). Applied in the context of conceptual data models, fuzzy quantifiers allow expressions about the number of instances which satisfy a given condition, or the proportion with respect to the total. We will study this in next sections. Of course, the quantifier Q must be previously defined in the data dictionary of the model.

In this context, we need a threshold $\gamma \in [0,1]$ indicating the minimum fulfillment degree that must be satisfied. This threshold will be written in square brackets: Q[γ]. For example, we may use "almost_all [0.2]", indicating that this fuzzy quantifier must be satisfied in a minimum degree of 0.2. Consequently, the underlining constraint requires that:

$$Q(\Phi) \geq \gamma \tag{5}$$

Every time the database is modified, the DBMS (DataBase Management System) computes Φ and checks whether Equation 5 is satisfied. The meaning of Φ will be defined in the next sections because it depends on where the fuzzy quantifier is used. In order to simplify the expression, we can set a default value for γ at 0.5, for example. If Q is an increasing function, then we can change Equation 5 because we obtain that:

$$\Phi \geq Q^{-1}(\gamma) \tag{6}$$

Similarly, if Q is a decreasing function, then what we get is:

$$\Phi \leq Q^{-1}(\gamma) \tag{7}$$

The last two equations may be useful because Q and γ are constants, whereas Φ is a varying value. Value Φ may change with every DML sentence (INSERT, DELETE or UPDATE).

In addition, another optional value δ can be established, which is greater than the threshold γ of minimum fulfillment degree, in the following way: Q[γ,δ] such that $\gamma<\delta$. The value δ is more restrictive than γ and it establishes that when the constraint is unfulfilled with this higher value, the DBMS will inform the user, but it will permit the modification of the database which is underway. Thus, if the quantifier is unfulfilled with a value between γ and δ, then the DBMS must warn the user (or only the database administrator). Therefore the warning message is generated when

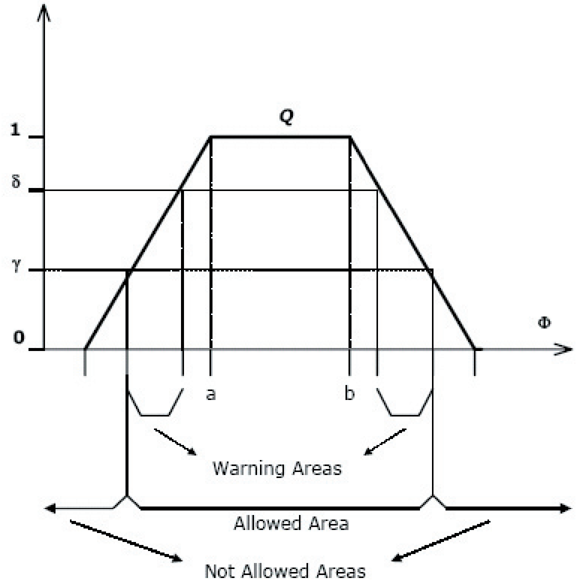
$$\delta \geq Q(\Phi) \geq \gamma \tag{8}$$

Figure 3 depicts a fuzzy quantifier with the thresholds γ and δ. Note that these thresholds divide the domain of Φ in three areas: the allowed area, the not allowed area and the warning area. The warning area is included in the allowed area. Note that the not allowed area is defined when Equation 5 is false.

Therefore, a fuzzy quantifier can be written in three ways:

1. Quantifier without threshold γ: Default threshold is $\gamma = 0.5$. For example, approx_2. For the purpose of simplicity we will use this form in the examples.
2. Quantifier with a threshold γ: For example, approx_8[0.25].
3. Quantifier with two thresholds γ and δ, with $\gamma<\delta$: For example, approx_3[0.5,0.6]. Both values would be close, in order to avoid too much warnings by the DBMS.

Figure 3: Thresholds γ and δ in a fuzzy quantifier "approximately between a and b", and its generated areas



Formalization of Quantifiers for Fuzzy Constraints

Fuzzy constraints may be used in two ways, and in both of them the constraint is denoted with an arc crossing the line where the constraint has effect. Besides, the arc is labeled with the quantifier or quantifiers according to the constraint type:

1. If the arc is labeled with the quantifier Q, this constraint establishes Equation 5, with Φ defined by Equation 4.
2. If the arc is labeled with the fuzzy (min,max) notation $(Q_{min};Q_{max})$ this constraint establishes the minimum and maximum number or proportion of elements fulfilling a certain constraint. In other words, fuzzy (min,max) notation $(Q_{min};Q_{max})$ establishes that:

$$\lambda_{min} \le \Phi_{min} \quad \wedge \quad \lambda_{max} \ge \varphi_{max} \tag{9}$$

where

$$\lambda_{min} = \min\{\alpha : \alpha = (Q_{min})^{-1}(\gamma_{min})\} \tag{10}$$

$$\lambda_{max} = \max\{\beta : \beta = (Q_{max})^{-1}(\gamma_{max})\} \tag{11}$$

where, γ_{min} and γ_{max} are the minimum thresholds for Q_{min} y Q_{max} respectively, and

$$\Phi_{min} = \begin{cases} a & \text{si } Q_{min} \text{ is absolute} \\ a/b & \text{si } Q_{min} \text{ is relative} \end{cases} \tag{12}$$

$$\Phi_{\max} = \begin{cases} a & \text{si } Q_{\max} \text{ is absolute} \\ a/b & \text{si } Q_{\max} \text{ is relative} \end{cases} \tag{13}$$

with a and b being the same values defined in previous case. To compute the warning area is easy, because we must use $\delta_{\min}$ and $\delta_{\max}$.

Fuzzy Degrees

Sometimes, fuzzy information is expressed with a degree. The domain of these degrees is usually limited to the interval [0,1], but other values can be allowed, as for example possibility distributions. We will use these degrees for measuring certain fuzzy components in aggregations and specializations.

Besides, the meaning of those degrees varies. Depending on this meaning the treatment of the data will be possibly different. The most important meanings of the grades are the following, and in Galindo (1999) and Galindo et al. (2001a), we found some authors who used these different meanings: **fulfilment** (a property can be complied with a certain degree between two ends), **membership** (which measures the level of membership or ownership of an object to a set), **importance** (different objects can have different importance, so that there are objects more important than others) and **uncertainty** (the degree expresses the security with which we know a specific data).

FUZZY AGGREGATIONS

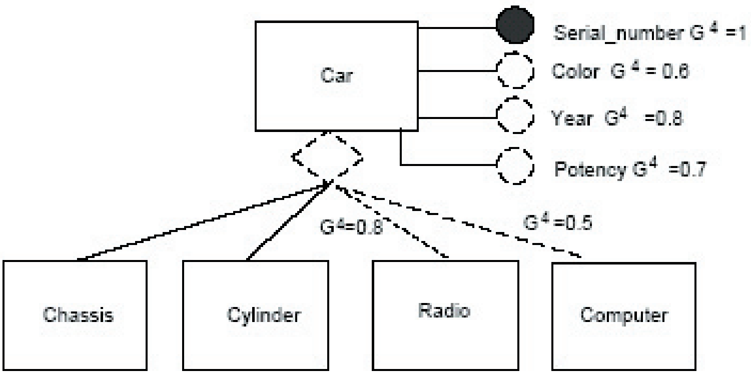
This approach is an extension of the first level by Zvieli and Chen (1986) applied to aggregations. De Miguel et al. (1999) define an aggregation like an entity which is composed of one set of different elements. They define two kinds of aggregations and we add a fuzzy degree to each element:

1. *Fuzzy aggregation of attributes:* This is the most common type of aggregation and it expresses that an entity is a set of attributes. Fuzzy aggregation of attributes is represented using circles with dashed lines for the graded attributes, indicating the degree of each one with: G^m=<degree>, where m is the meaning of this degree.
2. *Fuzzy aggregation of entities:* This aggregation expresses that each instance of an aggregated entity is composed of others' instances of others' entities. This aggregation is denoted by a rhombus with dashed line close to the aggregated entity. The other entities join the rhombus with a line labeled with: G^m=<degree>, where m is the meaning of this degree.

Example 2. Figure 4 models that a car has some attributes: serial number (the primary key), color, year, potency, etc. On the other hand, a car is composed of a chassis, an engine, radio and specialized computer, cylinder and other entities. Some of these elements (attributes and entities) have a membership degree to the model.

Thus, if we want a detailed model we can use all elements, but if we do not need such a detailed model we can get only elements with membership degree greater to 0.7,

Figure 4: Example 2. Fuzzy aggregations



for example. In this case, the model does not use some attributes (color and potency), and some entities (computer).

Fuzzy aggregation of attributes and fuzzy aggregation of entities are studied by Chen and Kerre (1998), and Ma et al. (2001) respectively, but their approaches are more limited. Besides, as we will see bellow we can use fuzzy cardinality constraints in aggregation.

FUZZY DEGREES IN SPECIALIZATIONS

This approach is an extension of the first level by Zvieli and Chen too. We can assign a degree to a specialization in two ways, and the meaning of this degree may be expressed in the model:

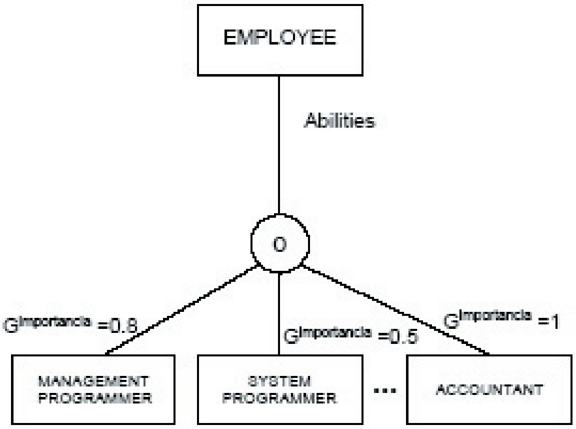
1. *Degree in the subclasses:* This degree expresses a fuzzy degree of one subclass in the specialization. It is denoted by G^m=<degree> labeling the line joining the subclass with the circle referred to as specialization circle, where m is the meaning of this degree.
2. *Degree in the specializations:* This degree expresses a fuzzy degree of all the specialization. It is denoted by G^m=<degree> labeling the specialization circle.

Example 3. Let us consider an entity Employee which is a superclass with various subclasses defining the abilities of the employees: Management Programmer, Systems Programmer, Internet Programmer, Analyst, Graphic Designer, Accountant, etc., just like Figure 5. These abilities have different importance denoted by the different degrees expressed in the model.

FUZZY COMPLETENESS CONSTRAINT ON SPECIALIZATIONS

The relationship between a class and all its subclasses can be *total*, if each member of the class (or superclass) must compulsorily be a member of one (or some) of the subclasses,

Figure 5: Example 3. Fuzzy degrees in a specialization



or *partial* if this condition is not necessary. The inverse is not possible since, by definition, each member of a subclass must be a member of the superclass.

Total participation is represented by a double line joining the superclass with the specialization circle, to which all the subclasses are joined using a single line. Partial participation is represented by a single line.

For our fuzzy model, this constraint can be fuzzy using a relative fuzzy quantifier (mainly, although they can also be absolute fuzzy quantifiers). This will be represented by an arc labeled with its fuzzy quantifier, crossing the line which joins the selected superclass with the circle.

Example 4. Let us consider the model in Figure 6 depicting an entity Employee which is a superclass with two subclasses defined by the attribute Contract Type: Permanent and Temporary. The arc and the relative fuzzy quantifier "almost all" (Figure 2) indicate that "Almost all employees must have a Permanent or Temporary contract, but other minority contract types may exist (work experience, grants...)". These other contract types are not included in the model for various reasons (unknown types, types without own attributes...).

In the previous example, the specialization is disjointed (with a "d" in the circle) since there cannot be an employee with various types of contracts. However, fuzzy completeness constraints can also be applied to overlapping specializations (with an "o" in the circle) as shown in the following example.

Example 5. Let us consider an entity Employee which is a superclass with various subclasses defining the abilities of the employees: Management Programmer, Systems Programmer, Internet Programmer, Analyst, Graphic Designer, Accountant, etc., just like Figure 7. The relative fuzzy quantifier like "almost all" indicates that "Almost all employees must have one or some of the abilities expressed in the subclasses."

Figure 6: Example 4. Fuzzy completeness constraint on an attribute-defined specialization with the defining attribute Contract_Type

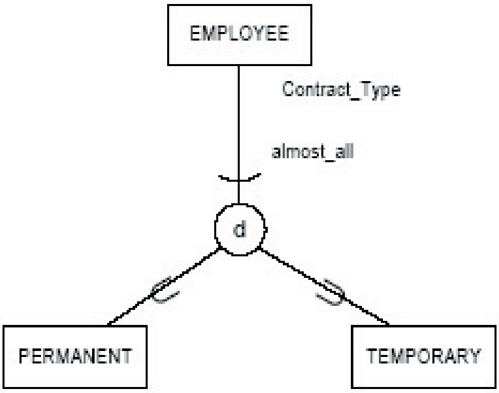
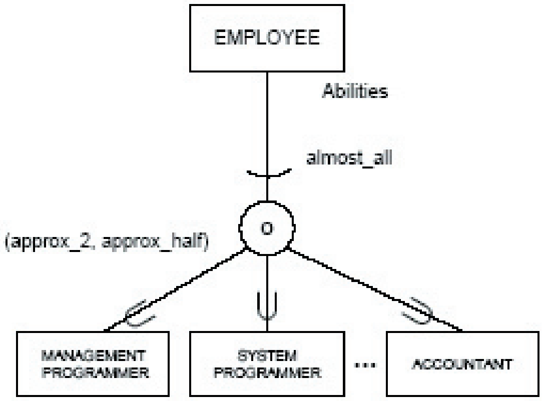


Figure 7: Examples 5 and 6. Fuzzy completeness constraint and fuzzy cardinality constraint on an overlapping specialization



In a new expression for fuzzy completeness constraints, we can use a fuzzy (min,max) notation instead of one quantifier. These minimum and maximum values restrict the quantity of superclass instances which belong to "any" subclass. This extension is not very useful.

FUZZY CARDINALITY CONSTRAINT ON OVERLAPPING SPECIALIZATIONS

In an overlapping specialization under the FuzzyEER model, we can also establish the minimum and maximum number of subclasses to which each member of the superclass can belong in a flexible manner. This can easily be expressed using the fuzzy (min,max) notation. The expression arising from this notation will be placed next to the circle containing the letter "o" (overlapping).

This fuzzy constraint has an effect on each superclass instance and must be satisfied by each one. In general, both min and max should be absolute quantifiers, although relative quantifiers will also be accepted (with regard to the total number of subclasses, value *b*).

Example 6. Continuing with Example 5, we can establish a fuzzy cardinality constraint on the overlapping specialization, such as: (approx_2, approx_half). This establishes the constraint whereby each employee must appear in a minimum of "approximately 2" skills and in a maximum of "approximately half" of existing skills (or subclasses).

This schema is depicted in Figure 7, too. Note that fuzzy quantifier almost always is a fuzzy completeness constraint and the (min,max) notation is used for a fuzzy cardinality constraint.

Finally, observe that the quantifiers can be of any type (absolute or relative) and each quantifier can also be followed, optionally of course, by one or two fulfillment degrees in square brackets [γ,δ], as explained previously.

The fuzzy cardinality constraint may be used also in the aggregation of entities. The fuzzy quantifier or the fuzzy (min,max) notation may label an arc crossing the line which joins one entity with the rhombus in the aggregation. Notice that the aggregated entity may be composed of some instances of one entity. For example, we can use fuzzy quantifier approx_6 constraining the number of cylinders of a car (see Example 2 and Figure 4).

FUZZY DISJOINTED OR OVERLAPPING CONSTRAINTS ON SPECIALIZATIONS

In specializations, the disjointed constraint specifies that the subclasses of the specialization must be disjointed. This means that an entity can be a member of at most one of the subclasses. If the subclasses are not constrained to be disjointed, it is an overlapping specialization.

Thus, it can be interesting to include to what extent the superclass instance belongs to each of the subclasses using linguistic labels ("a lot", "a little"...) or, more simply, membership degrees in the interval [0,1]. Note that it is to consider each subclass as a fuzzy subset of the superclass. Just like all fuzzy sets, its elements are not clearly defined, since each element can belong to the fuzzy set with a certain degree.

This extension can be applied on disjointed or overlapping specializations and such specializations will be represented with letter "f" (fuzzy) before the other letter in the circle, i.e., "fd" for fuzzy disjointed specializations and "fo" for fuzzy overlapping specializations. However, it does not force all subclasses to be fuzzy subsets: fuzzy subclasses are represented with dashed rectangles.

This definition has two points of view:

1. **From the point of view of subclasses**: Subclasses are fuzzy sets and their underlying domain is all superclass instances, i.e., each superclass instance has a membership degree to each subclass (including the value zero). Let *A* be a subclass of *S*. Then the fuzzy set of A is represented by the following equation (using the Equation 2 format):

$$\{\mu_A(u_1)/u_1, \mu_A(u_2)/u_2, \ldots \mu_{An}(u_n)/u_n\} \qquad (14)$$

where u_i, with $i = 1, 2, \ldots, n$, are all instances of superclass S, and $\mu_A(u_1)$ is the membership degree of u_i to subclass A.

2. **From the point of view of superclass instances**: Each superclass instance may belong to some subclasses. This membership is measured with a fuzzy set. The underlying domain of this fuzzy set is the set of all subclasses names. Let A_j, with $j = 1,2,\ldots m$ be the m subclasses of S. Then the fuzzy set of instance u_i is:

$$\{\mu_{A1}(u_1)/A_1, \mu_{A2}(u_2)/A_2, \ldots \mu_{An}(u_n)/A_n\} \qquad (15)$$

where $\mu_{Aj}(u_1)$, with $j = 1,2,\ldots m$, is the membership degree of u_i to subclass A_j. Note that in disjointed specializations the number of subclasses for a superclass instance is one.

Observe that both points of view work with fuzzy sets with a different discrete underlying domain.

Example 7. Figure 8 indicates that our conceptual schema is also concerned with storing to what extent each employee belongs to each of the subclasses. Thus, the system's programmers set is a fuzzy set (an employee can belong to this set with a certain membership degree), whereas the set of accountant is not a fuzzy set (an employee can or cannot belong to this set). This is the first point of view.

The second point of view starts with a particular employee: an employee who is an expert at programming management applications, although he may also be skilled in other types of applications and less skilled as an analyst, could be represented in the database by the following fuzzy set: {1/Management Programmer, 0.8/Systems Programmer, 0.3/Analyst}. Note that the underlying domain is the set of all subclasses names.

This will allow us to make selections of the type: "Find the name of the best management applications programmer amongst those who are not assigned to many projects and who is at least a regular analyst."

This constraint does not prevent the use of other fuzzy constraints (completeness or cardinality). However, when they are mixed with a fuzzy disjointed or overlapping constraint, they must be studied in order to define the method with which the DBMS ensures the fulfillment of these constraints:

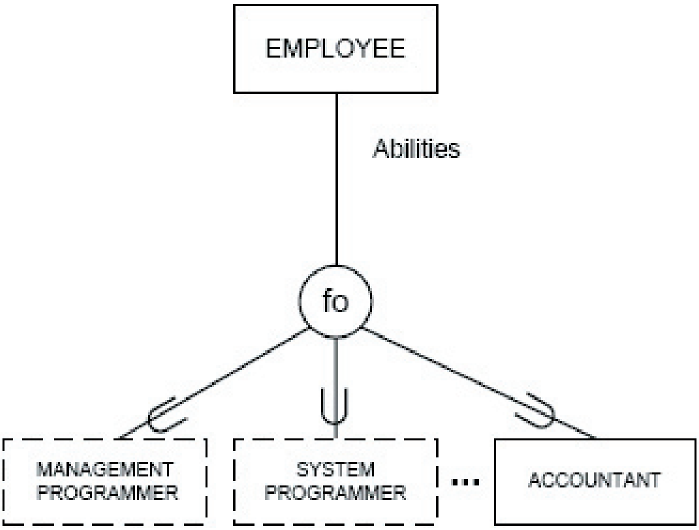
1. If a **fuzzy completeness constraint** exists then the DBMS must compute whether each superclass instance belongs to some subclass; for example, in order to decide if "almost all" superclass instances belong to some subclass. The problem is that membership is now fuzzy. Membership degree of an instance to the subclasses may be computed in various manners: a) By using the maximum membership degree of this instance to any subclass, i.e., the height (Pedrycz et al., 1998) of the second point of view fuzzy set, b) By using the fuzzy set cardinality (Pedrycz et al., 1998) of the second point of view fuzzy set (adding all membership degrees) or by using generalized measures, like the fuzzy set energy (De Luca et al., 1974). We can, certainly, set a minimum threshold in order to decide whether an instance belongs to some subclass.

2. If a **fuzzy cardinality constraint** exists then the DBMS must compute the number of subclasses to which each superclass instance belongs to; for example, in order to decide if the number of subclasses of a superclass instance is between "approximately_2" and "approximately_half" of existing subclasses. However, this number is not simple, because membership is now fuzzy. This problem may be solved in two ways: a) By using the fuzzy set cardinality (Pedrycz et al., 1998) of the second point of view fuzzy set or by using generalized measures, like the fuzzy set energy (De Luca et al., 1974), b) By counting the number of subclasses with a membership degree greater than a minimum value (usually zero). Once the DBMS has computed this number, the system must check if this number satisfies the fuzzy cardinality constraint.

FUZZY ATTRIBUTE DEFINED SPECIALIZATIONS

There are some kinds of fuzzy attributes, summarized in (Galindo et al., 2001a). Some models (Medina et al., 1994) and applications (Galindo et al., 1998; Galindo, 1999) use the following ones. The so-called fuzzy attributes Type 1 are totally crisp (traditional), but they have some linguistic trapezoidal labels defined on them, which allow us to make the query conditions for these attributes more flexible (cold, warm...). With these attributes we can use fuzzy queries in classic databases. Fuzzy attributes Type 2 admit crisp or fuzzy data over an ordered underlying domain. Fuzzy attributes Type 3 do not have an ordered underlying domain, for instance the hair color. On these attributes some labels are defined (fair, brown, red-haired...) and on these labels a similarity relation has yet to be defined. Thus, each two labels are equal (or similar) with a similarity degree in [0,1]. Besides fuzzy attributes, Type 3 admits fuzzy sets (or possibility distributions) on its underlying domain. An example of these fuzzy sets is {1/brown, 0.5/red haired, 0.2/fair}. In some contexts a fuzzy attribute

Figure 8: Example 7. Fuzzy overlapping specialization



Type 3 does not have a similarity relation defined in its domain. We name these attributes as fuzzy attributes Type 4.

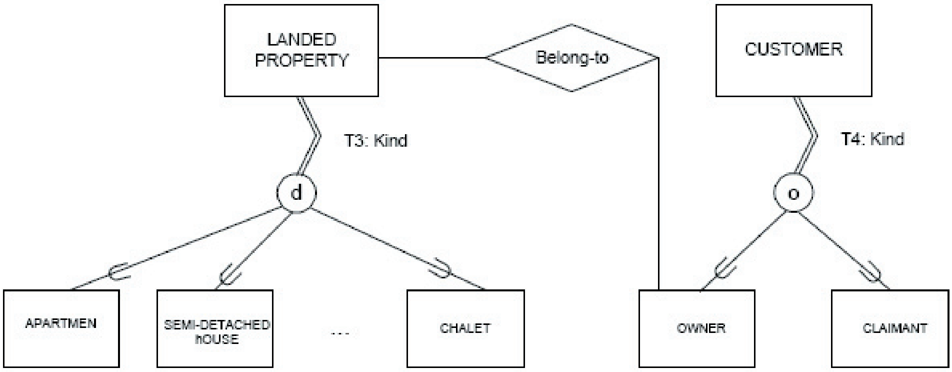
We define a fuzzy attribute defined specialization just like an attribute defined specialization (Elmasri et al., 2000) where this attribute is a fuzzy attribute. It is represented with an angled line joining the superclass with the circle. This line will be labeled with the name of the fuzzy attribute Type n, preceded by the text “Tn:”. This definition is independent of all constraints like fuzzy or crisp disjointed or overlapping specializations.

The following example shows two fuzzy attribute defined specialization (disjointed and overlapping). In one specialization, each pair of subclasses has a fuzzy similarity degree between them (Type 3). This property is useful to compare them and to search the more important instances in some queries. In the other specialization, the similarity relation does not exist (Type 4).

Example 8. The conceptual model represented in Figure 9 expresses that in a real estate agency, every landed property belongs to one subclass, which has its own attributes. Thus, this is a total disjointed specialization (double line and “d” inside the circle). Attribute Kind is a fuzzy attribute Type 3, because if one person is looking for a chalet, for example, then this customer is, possibly, interested in semi-detached houses because these two types are similar. Thus, this is taken into account in order to show to our customer all the relevant properties. In this sense, fuzzy queries are studied in Galindo et al. (1998, 1999) and Galindo (1999). Observe that subclasses are not fuzzy, because every landed property belongs only to one subclass.

Every landed property has an owner, which is a customer. Another kind of customer is a claimant who is looking for a landed property. The overlapping specialization makes it such that one customer may be owner and claimant at the same time. The fuzzy attribute Type 4, Kind, makes it possible to store a possibility distribution about the subclasses in order to express any fuzzy concept. In this example we are interested in measuring the urgency of the customer. Thus, a customer with the value {0.4/Owner, 1/Claimantg} is a customer who is looking for a landed property urgently and who is offering some property

Figure 9: Example 8. Two fuzzy attribute defined specializations



without urgency. Note that subclasses are not fuzzy, because a customer is or is not owner and/or claimant.

Example 9. Figure 11 includes another three examples of fuzzy attribute defined specializations using two fuzzy overlapping specializations and one disjointed specialization. The first one is a specialization with a total participation constraint (double line) and it establishes that all employees must belong to one or more categories. Besides, Category is a fuzzy attribute Type 3.

The second one is a specialization with a fuzzy participation constraint with the fuzzy quantifier almost all in the labeled arc: Almost all researches must belong to one or more research lines. Besides, Research Line is a fuzzy attribute Type 3.

The third one is a disjointed specialization with a total participation constraint and it establishes that all temporary employees are beginners or seniors, according to the antiquity. Subclasses are not fuzzy because we do not want to store the membership degree. Besides, a temporary employee cannot belong to both subclasses. The antiquity is a crisp and known value but we can make flexible queries using this attribute, i.e., it is a fuzzy attribute Type 1.

FUZZY CONSTRAINTS IN UNION TYPES OR CATEGORIES: PARTICIPATION AND COMPLETENESS

In the EER model we can also find the union types or categories (Elmasri et al., 1985; 2000). It represents the case when some different superclasses may be members of a special subclass (called category) or not. By definition, each member of the subclass or category must be a member of at least one of the superclasses. Furthermore, in partial categories it is possible that superclass instances do not belong to the category, because the category is a subset of the union of all superclasses.

Union types are represented with the union symbol inside a circle. Superclasses are joined to that circle by a line. Subclass or category is joined to that circle using a single line with the inclusion symbol. In this type of specialization it is possible to apply fuzzy constraints in two ways:

1. *Fuzzy participation constraint in one or more superclasses:* This constraint restricts the number of instances, in the union of any group of superclasses, which belong to the category. This is represented by an arc labeled with its fuzzy quantifier, crossing the line which joins the selected superclass with the circle. Normally, this fuzzy quantifier will be relative. For example, with the quantifier "almost all" on a superclass the constraint expresses that: "almost all superclass elements belong to the category". Another option is to join two or more superclasses with an arc indicating that the union of instances of those superclasses is constrained in participation. This constraint allows the use of the (min,max) notation indicating the minimum and maximum number of instances which belong to the category (using absolute or relative fuzzy quantifiers).

2. *Fuzzy completeness constraints in the category (on the union of all superclasses):* This constraint restricts the number of instances, of all superclasses (the union), which belong to the category. This is represented by an arc labeled with its fuzzy quantifier, crossing the line which joins the category with the circle. Normally, this fuzzy quantifier will be relative. For example, with the quantifier "almost all" on the category the constraint expresses that: "almost all elements of all superclasses belong to the category". This constraint allows the use of the fuzzy (min,max) notation too, indicating the minimum and maximum number of all superclasses instances which belong to the category. Notice that this second way is always referred to as all superclasses instances, i.e., to the union of all superclasses. This reason makes it such that relative fuzzy quantifiers are preferred in this constraint.

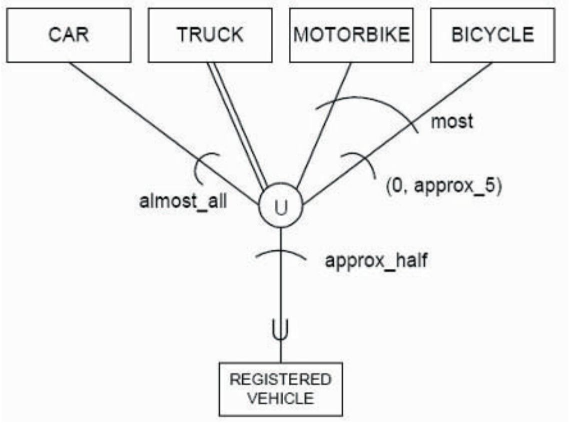
Example 10. Let us consider four entity types for vehicles: Car, Truck, Motorbike and Bicycle. Some vehicles may belong to the Registered Vehicle entity. Figure 10 depicts this model with some participation constraints: Almost all cars and all trucks must be registered vehicles. Besides, the model allows a maximum of approximately five bicycles to be registered vehicles. The arc labeled with the fuzzy quantifier Most indicates that most motorbikes or bicycles (its union) must be registered.

On the other hand, fuzzy completeness constraint establishes that approximately half of the existing vehicles must be registered vehicles.

FUZZY CONSTRAINTS IN SHARED SUBCLASSES: PARTICIPATION AND COMPLETENESS

A shared subclass (or intersection type) is a subclass with several superclasses (Elmasri et al., 2000). Each member of the subclass must be a member of all of the superclasses, i.e., the subclass is a subset of the intersection of all of the superclasses. A shared subclass

Figure 10: Example 10. Fuzzy constraints on a union type or category



is represented by joining it with all of its superclasses by a single line with the inclusion symbol. Another representation utilizes the intersection symbol inside a circle: Superclasses are joined to that circle by a line and the subclass is joined to that circle using a single line with the inclusion symbol.

Just as with union types, in this type of specialization it is possible to apply fuzzy constraints in two ways:

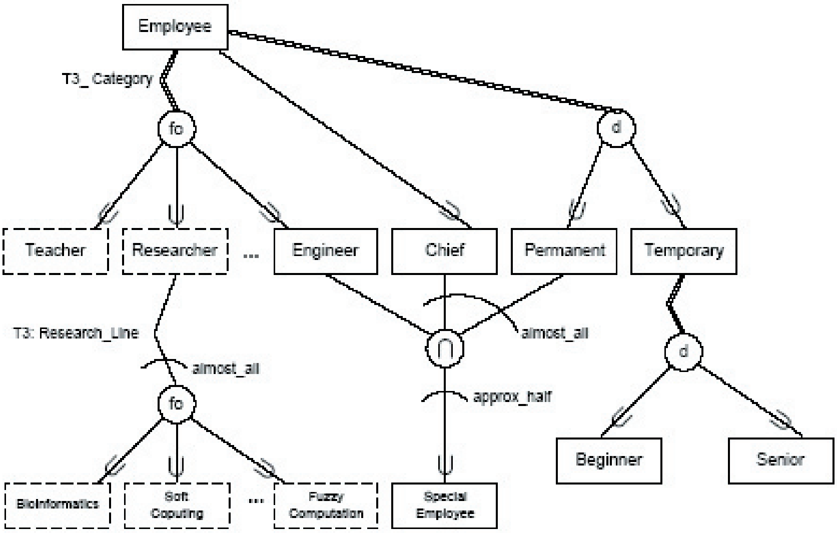
1. *Fuzzy participation constraint in one or more superclasses:* This constraint restricts the number of instances, in the intersection of any group of superclasses, which belong to the shared subclass. This is represented by an arc labeled with its fuzzy quantifier crossing the line which joins the selected superclass with the circle. This fuzzy quantifier would be relative. For example, with the quantifier "almost all" on a superclass the constraint expresses that: "almost all superclass elements belong to the shared subclass". Another option is to join two or more superclasses with the arc indicating that the intersection of instances of those superclasses is constrained in participation. This constraint allows the use of the fuzzy (min,max) notation indicating the minimum and maximum number of instances which belong to the shared subclass (using absolute or relative fuzzy quantifiers). Generally, the participation constraint is not useful, because one constraint on one superclass (or on several superclasses) depends on the membership of its instances to the other superclasses (remember that the subclass is a subset of the intersection).
2. *Fuzzy completeness constraints in the shared subclass (on the intersection of all superclasses):* This constraint restricts the number of instances, in the intersection of all superclasses, which belong to the shared subclass. This is represented by an arc labeled with its fuzzy quantifier, crossing the line which joins the shared subclass with the circle. Normally, this fuzzy quantifier will be relative. For example, with the quantifier "almost all" on the shared subclass the constraint expresses that: "almost all elements of the intersection of all superclasses belong to the shared subclass". This constraint allows the use of the fuzzy (min,max) notation too, indicating the minimum and maximum number of instances in the intersection (of all superclasses) which belong to the shared subclass. Notice that this constraint is always referred to as the intersection of all superclasses.

Example 11. Let us consider an entity for Special Employees with its own attributes (extra, payment, number of awards, motive...). A member of this shared subclass must be engineer, chief and a permanent employee. Figure 11 depicts this model with the following participation constraint: Almost all chiefs and permanent employees must be special employees. It is interesting to note how this constraint enforces that almost all chiefs and permanent employees must be engineers, too. It must be remembered that all special employees belong to Engineer superclass.

On the other hand, fuzzy completeness constraint establishes that approximately half of employees who are engineers, chiefs and permanent employees must be special employees.

In real models fuzzy constraints in the same specialization must be mixed with care. Observe that a fuzzy participation constraint embracing all superclasses is a fuzzy completeness constraint (both in union types and in intersection types).

Figure 11: Examples 9 and 11. Three fuzzy attribute defined specializations, two fuzzy overlapping specializations and fuzzy constraints in a shared subclass



CONCLUSIONS AND FURTHER RESEARCH

Fuzzy logic allows us to bring the operation of information systems closer to the working methods of humans. People control fuzzy concepts very often (terms like “almost all”, “the majority”, “approximately 8”...), which include a certain vagueness or uncertainty and which traditional information systems do not understand, and therefore cannot use.

Fuzzy databases (Galindo, 1999; Medina et al., 1994; Petry, 1996) have also been widely studied with the following main objectives: firstly, to allow the storage of imprecise or fuzzy data, and secondly, to allow the possibility of imprecise or fuzzy queries, using the existing data (whether imprecise or not). Traditionally, the application of fuzzy logic to databases has paid scant attention to the problem of conceptual modeling.

The extension of the EER model (Connolly et al., 1998; Elmasri et al., 2000) for dealing with fuzzy data has been studied in some publications (Chaudhry et al., 1994, 1999; Chen & Kerre, 1998; Ma et al., 2001; Zvieli & Chen, 1986), but these approaches are partial: They study or extend only some aspects or, in some cases only one aspect of EER model. For example, none of them refers to the possibility of extending constraints by using the tools offered by fuzzy sets theory. In this context, FuzzyEER model (Galindo et al., 2001b, 2003, 2004; Urrutia et al., 2002a, 2002b, 2003) is a tool for fuzzy modeling, based on EER model.

In this chapter, we have defined the FuzzyEER aspects related with aggregations and specializations: fuzzy aggregations, fuzzy degrees in specializations, fuzzy completeness constraint on specializations, fuzzy cardinality constraint on overlapping specializations, fuzzy disjointed or overlapping constraints on specializations, fuzzy attribute defined specializations, fuzzy constraints in union types or categories and fuzzy constraints in shared subclasses (or intersection types). The defined constraints can be represented using fuzzy quantifiers (Galindo, 1999; Galindo et al., 2001a; Yager, 1983; Zadeh, 1983) and the fuzzy (min,max) notation.

These fuzzy extensions have a novel meaning and offer great expressiveness to the conceptual model. However, we think that FuzzyEER can be extended even more. Besides, we must study possible problems and improvements in the implementation of the resulting model.

An interesting study to facilitate the task of using fuzzy quantifiers on the part of designers would be to classify the quantifiers which can be used in natural language, and study the relationship between them.

The next step will be the automatic implementation of the model, including the necessary triggers to activate the fuzzy constraints described, and the study of different tools to facilitate the query of stored data, especially with regard to the fuzzy belonging of a superclass to different subclasses. For this last objective we can use and extend the fuzzy query language FSQL (FuzzySQL), an extension of the popular SQL which allows dealing with imprecise data (Galindo et al., 1998; Galindo, 1999). We are now studying how subclasses can inherit properties of their superclasses with such fuzzy extensions.

Another research line is to achieve notational constructs to allow a greater selection of other fuzzy integrity constraints; for example, relaxing the constraints proposed in Davis et al. (1989).

Anther target is the modeling of a real application for a real estate agency, using all these ideas and some new ones. We started with the definition presented in Galindo et al. (1999) and one first approach is in Urrutia et al. (2002) and Urrutia (2003). Another research line was published in Aranda et al. (2002).

REFERENCES

Aranda, M.C., Galindo, J., & Urrutia, A. (2002). Museos digitales en Internet: Modelo EER difuso y recuperación de imágenes basada en contenido. IV *Turismo y tecnologías de la información y las comunicaciones (TuriTec'2002)* (pp. 411-425). Málaga (Spain), ISBN: 84-600-9813-3.

Chaudhry, N., Moyne, J., & Rundensteiner, E.A. (1994). A design methodology for databases with uncertain data. *7th International Working Conference on Scientific and Statistical Database Management* (pp. 32-41). Charlottesville, VA. Available: www.mitexsolutions.com

Chaudhry, N., Moyne, J., & Rundensteiner, E.A. (1999). An extended database design methodology for uncertain data management. *Information Sciences, 121*, 83-112.

Chen, G.Q., & Kerre, E.E. (1998). Extending ER/EER concepts towards fuzzy conceptual data modeling. *IEEE International Conference on Fuzzy Systems, 2,* 1320-1325.

Chen, P. (1976). *The Entity-Relationship Model-Toward a unified view of data. ACM Transactions on Database Systems (TODS), 1*(1), 9-36.

Connolly, T., Begg, C., & Strachon, A. (2001). *Data bases system, a practical approach to design, implementation and management* (2nd Edition). Addison Wesley.

Davis, J.P., & Bonnell, R.D. (1989). Modeling semantic constraints with logic in the EARL data model. *Proc. Fifth International Conference on Data Engineering* (pp. 226-233).

De Luca A., & Termini S. (1974). Entropy of L-Fuzzy Sets. *Information and Control, 24*, 55-73.

De Miguel, A., Piattini, M., & Marcos, E. (1999). *Diseño de bases de datos relacionales.* Rama.

Elmasri, R., & Navathe, S.B. (2000). *Fundamentals of database systems.* (3rd Edition). Addison-Wesley.

Elmasri, R., Weeldreyer, J., & Hevner, A. (1985). The category concept: an extension to the Entity-Relationship Model. *International Journal on Data and Knowledge Engineering, 1*(1), May.

Galindo, J. (1999). *Tratamiento de la imprecisión en bases de satos eelacionales: Extensión del modelo y adaptación de los SGBD actuales.* Ph. Doctoral Thesis, University of Granada (Spain). Available: www.lcc.uma.es

Galindo J., & Urrutia, A. (2003). Fuzzy extensions to EER specializations. *Eighth CAiSE/IFIP8, International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'03),* Velden, Austria (pp. 218-227).

Galindo, J., Medina, J.M., Cubero J.C., & García, M.T. (2001a). Relaxing the universal quantifier of the division in fuzzy relational databases. *International Journal of Intelligent Systems, 16*(6), 713-742.

Galindo, J., Medina, J.M., Cubero, J.C., & Pons, O. (1999). Management of an estate agency allowing fuzzy data and flexible queries. *EUSFLAT-ESTYLF Joint Conference*, Palma de Mallorca (Spain) (pp. 485-488).

Galindo, J., Medina, J.M., Pons, O., & Cubero, J.C. (1998). A server for fuzzy SQL queries. In T. Andreason, H. Christiansen & H.L. Larsen (Eds.), *Flexible query answering systems lecture notes in artificial intelligence (LNAI) 1495* (pp. 164-174). Springer.

Galindo, J., Urrutia, A., & Piattini, M. (2004). *Fuzzy databases: Modeling, design and implementation.* Hershey, PA: Idea Group Publishing (forthcoming).

Galindo, J., Urrutia, A., Carrasco, R., & Piattini, M. (2001b). Fuzzy constraints using the enhanced Entity-Relationship Model. Proceedings published by *IEEE-CS Press of XXI International Conference of the Chilean Computer Science Society (SCCC 2001)*, Punta Arenas (Chile) (pp. 86-94). ISBN: 0-7695-1396-4. ISSN: 1522-4902. Available: http://computer.org/proceedings/sccc/1396/13960086abs.htm

Hammer M., & McLeod D. (1981). Database Description with SDM: A Semantic Data Model. *ACM Transactions on Database Systems* (TODS) *6*, 3.

Ma, Z.M., Zhang, W.J., Ma, W.Y., & Chen Q. (2001). Conceptual design of Fuzzy Object-Oriented Databases Using Extended Entity-Relationship Model. *International Journal of Intelligent System, 16*(6), 697-711.

Medina J.M., Pons O., & Vila M.A. (1994). GEFRED. A Generalized Model of Fuzzy Relational Databases. *Information Sciences, 76*(1/2), 87-109.

Pedrycz, W., & Gomide, F. (1998). *An Introduction to Fuzzy Sets: Analysis and Design.* A Bradford Book. ISBN 0-262-16171-0. MA: MIT Press.

Petry, F. E. (1996). Fuzzy Databases: Principles and Applications (with chapter contribution by Patrick Bosc). In H.J. Zimmerman (Ed.), *International Series in Intelligent Technologies.* Kluwer Academic Publ. (KAP).

Urrutia, A. (2003). *Definición de un Modelo Conceptual para Bases de Datos Difusas.* Ph.D. Thesis, University of Castilla-La Mancha (Spain).

Urrutia, A., & Galindo, J. (2002a). Algunos Aspectos del Modelo Conceptual EER Difuso: Aplicación al Caso de una Agencia Inmobiliaria, XI Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF'2002) (pp. 359-364). León (Spain).

Urrutia, A., Galindo, J., & Piattini, M. (2002b). Modeling Data Using Fuzzy Attributes. *IEEE Computer Society Press, XXII International Conference of the Chilean Computer Science Society (SCCC 2002)* (pp. 117-123). Copiapo (Chile). ISBN: O-7695-1867-2 ISSN: 1522-4902. Available: http://computer.org/proceedings/sccc/1867/18670117abs.htm

Vert, G., Morris, A., Stock, M., & Jankowski, P. (2000). Extending Entity-Relationship Modelling Notation to Manage Fuzzy Datasets. *8th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU2000,* (pp. 1131-1138). Madrid, Spain.

Yager, R.R. (1983). Quantified Propositions of a Linguistic Logic. *International Journal of Man-Machine Studies*, *19*, 195-227.

Zadeh, L.A. (1965). Fuzzy Sets. *Information and Control, 8,* 338-353.

Zadeh, L.A. (1983). A Computational Approach to Fuzzy Quantifiers in Natural Languages. *Computer Mathematics with Applications*, *9*, 149-183.

Zvieli, A., & Chen, P. (1986). ER Modeling and Fuzzy Databases. *2nd International Conference on Data Engineering,* 320-327.

Chapter VII

Normalization of Relations with Nulls in Candidate Keys:

Traditional and Domain Key Normal Forms

George C. Philip, University of Wisconsin Oshkosh, USA

ABSTRACT

This chapter discusses normalization of relations when the candidate keys of a relation have missing information represented by nulls. The chapter shows that problems and confusion can arise in normalizing relations with nulls in candidate keys. Candidate keys with missing information commonly are found in relations that represent information on two entities with a one-to-one relationship between them. The current definition of Boyce-Codd Normal Form (BCNF) is ineffective in identifying poor designs in such relations that may have insertion/deletion anomalies. Domain Key Normal Form (DKNF) also suffers from the same problem. It is shown that the above problem can be corrected by incorporating the concept of entity integrity rule into the definitions of BCNF and DKNF. This chapter also shows that incorporating the entity integrity rule into the definition of either a relation or a candidate key does not provide a satisfactory solution to the problem.

INTRODUCTION

The relational database design concepts were developed without considering missing information in relations (Codd, 1986; Levene, 1999; Date 2000). Value of an attribute in a tuple may be missing for several reasons: 1) Value is applicable but it is unknown, 2) Value is not applicable, 3) Value does not exist, or 4) Other reasons, such as value is undefined (Date 2000). An example of value that is not applicable is the attribute driver license number for a ten-year-old child. If an adult does not have a driver license number, then the value does not exist. If an adult has a driver license number, but it is unknown, then the value is applicable but unknown.

A common method of representing missing values is using nulls (Codd, 1986). Other methods include using default values (Date, 1990), using a subset of the attribute domain (Lipski, 1979), and using variables or many different "null values" (Imielinski & Lipski, 1984). Missing information can create problems in querying data from relations (Imielinski & Lipski, 1984; Date, 1990). Several methods have been proposed to extend the relational operators to deal with missing values (Codd, 1986; Reiter, 1986; Sutton & King, 1995). Another group of studies examined the effect of nulls on the concept of functional dependency (Vassiliou, 1980; Vardi, 1986; Levene & Loizou, 1999). These studies have focussed primarily on missing values of the type "applicable but unknown".

The current paper examines the effect of nulls in candidate keys on normalizing a relational schema. The nulls considered in this paper are of the type "not applicable" or "does not exist". Specifically, this paper examines the effectiveness of Boyce-Codd Normal Form (BCNF) and Domain Key Normal Form (DKNF) in identifying insertion/deletion anomalies if missing values in candidate keys are represented by nulls. Candidate keys with nulls commonly are found in relations that represent information on two entities with a one-to-one relationship between them. It is shown that the current definition of Boyce-Codd Normal Form is ineffective in identifying poor designs in such relations. Domain Key Normal Form (DKNF) also suffers from the same problem. The paper identifies the source of the problem and offers a solution by incorporating the concept of entity integrity rule into the definitions of BCNF and DKNF. This paper also shows that incorporating the entity integrity rule into the definition of either a relation or a candidate key does not provide a satisfactory solution to the problem.

DESCRIPTION OF THE PROBLEM

To help explain the problem, we consider two entities, EMPLOYEE and COMPUTER, that have a (zero-or-one)-to-(zero-or-one) relationship between them. Thus, a computer has zero or one employee assigned to it at any given time. Similarly, an employee is assigned to zero or one computer at any time. Consider a relation:

ASSIGNMENT (ID, NAME, TITLE, COMPUTER_NO, MODEL, RAM).

In the above relation, ID, NAME, and TITLE represent the identification number, the name, and the title of the employee, respectively. ID is the only unique identifier of the employee. COMPUTER_NO is the only unique identifier of the computer assigned to the employee. MODEL and RAM represent the model, and the amount of memory of the employee's computer, respectively. Figure 1 shows a sample state of the relation.

Figure 1: A state of the relation ASSIGNMENT

ID	NAME	TITLE	COMPUTER_NO	MODEL	RAM
E1	A. Adams	Manager	null	null	null
E2	B. Brown	V.P.	null	null	null
E3	C. Carlos	Manager	C1	Model1	128
E4	J. Jones	Sales Rep	C2	Model1	64
E5	J. Jones	Accountant	C3	Model2	64
null	null	null	C4	Model2	128

Since some employees may not have a computer, the corresponding tuples in ASSIGNMENT do not have any value for COMPUTER_NO, MODEL, and RAM. Similarly, since some computers do not have employees assigned to them, the corresponding tuples do not have any value for ID, NAME and TITLE. Here, null may represent value that does not exist or value that is not applicable. For example, one employee may not be eligible for a computer (not applicable), while another employee may be eligible, but no computer was assigned (does not exist).

The above design is not a good one. The relation suffers from insertion and deletion anomalies. If ID is selected as the primary key, then information on a computer cannot be inserted if the computer is not assigned to an employee, as in the case of the computer identified by C4. By assumption, if an employee leaves the organization, the employee's computer may not be assigned to anyone. In such cases, deleting the tuple for an employee will result in losing the information on the corresponding computer. However, deleting tuples of employees who do not have a computer does not result in losing information on any computer. Similar problems exist if COMPUTER_NO or COMPUTER_NO+ID is selected as the primary key. Thus, ASSIGNMENT in its current form suffers from insertion and deletion anomalies. These anomalies could be removed by decomposing the relation into two relations by taking projections:

1) EMPLOYEE (ID, NAME, TITLE, COMPUTER_NO),
2) COMPUTER (COMPUTER_NO, MODEL, RAM).

Or,

1) EMPLOYEE (ID, NAME, TITLE),
2) COMPUTER (COMPUTER_NO, MODEL, RAM, ID).

Though ASSIGNMENT has insertion and deletion anomalies that can be removed by decomposition, applying the current popularly-used definitions of relation, determinant, candidate key, and BCNF leads to the conclusion that the above relation is in BCNF (and also in 4th and 5th normal forms), as explained in the next section.

Before analyzing ASSIGNMENT further, we discuss the relevance of relations such as ASSIGNMENT that combine information on multiple entities, and the practical importance of one-to-one relationships. Database design based on the popular top-down approach uses three steps (Elmasri & Navathe, 2000): 1) identify the entities and their relationships, 2) apply the mapping rules to create relations from entities, and 3) perform the normalization procedure to validate the design. Ideally, the designer should identify the entities and their relationships correctly, and hence, should not create relations that combine information on multiple entities. Under this ideal condition, step 3 (normalization) may not be necessary. However, in practice, all designers may not identify entities correctly. For example, an inexperienced designer might view an order as one entity with attributes Order Id, Order Date, Customer Id, Item Id, and Quantity. Similarly, the attributes Id, Name, Title, Computer_No, Model and RAM might be viewed as part of a single entity that represents the assignment of employees to computers. Applying the mapping rules to such entities would result in relations that represent information on multiple entities, similar to ASSIGNMENT. Normalization is important in identifying such cases. In the bottom-up approach that typically starts with a collection of attributes belonging to multiple entities, it is even more likely to produce relations like ASSIGNMENT. Normalization rules are viewed as a formal framework to minimize insertion, deletion, and update anomalies. Hence, it would be desirable for these rules to stand on their own without depending on the ability of the designer to identify the entities correctly.

How important are one-to-one relationships in the real world? Relationships that are identified in business applications as one-to-one often may not be "pure" one-to-one relationships, if all possible current and future exceptions are considered. Many relationships identified as one-to-one might be one-to-many or many-to-many relationships, in theory. However, if the number of instances that are exceptions to the one-to-one relationship are small enough, or the chances of having to store such exceptions in the database is small, it might be desirable to treat such relationships as one-to-one, to better meet the objectives of physical design like improving performance and resource requirements. For example, consider the entities FACULTY and OFFICE in the database for a large university. Only one or two offices have more than one faculty. Similarly, it is very uncommon for a faculty to have more than one office. If the current situation is expected to continue, treating the relationship as one-to-one could provide certain benefits without making significant sacrifices on data redundancy: 1) Compared to treating the relationship as many-to-many, treating it as one-to-one doesn't require an associative entity to represent the relationship, 2) Compared to treating the relationship as one-to-many, treating it as one-to-one gives more flexibility in placing the foreign key on FACULTY or OFFICE based on search patterns and/or presence of nulls in foreign keys. Thus, one-to-one relationships become important in the practice of database design, though the number of "pure" one-to-one relationships may be small.

TEST FOR BOYCE-CODD NORMAL FORM (BCNF)

A commonly accepted definition is that a relation is in BCNF if and only if every determinant is a candidate key (Connolly & Begg, 2002; Date, 2000; Hoffer, Prescott & McFadden, 2002; Kroenke, 2002; Rob & Coronel, 2002; Watson, 1999). The actual wording of the definitions presented in this section may vary among different authors, but the meaning remains the same. The properties of a *relation* are: 1) There are no duplicate tuples, 2) Tuples are unordered, 3) Attributes are unordered, and 4) All attributes are atomic. A *determinant* is any set of attributes on which another set of attributes is fully functionally dependent. A set of attributes Y is *fully functionally dependent* on another set of attributes X if it is functionally dependent on X and not functionally dependent on any subset of X. A set of attributes Y is *functionally dependent* on another set of attributes X, that is, X -> Y, if for every valid instance of X, the values of X uniquely determine the values of Y (Codd, 1972; Connolly & Begg, 2002; Dutka & Hanson, 1989; Hoffer, Prescott & McFadden, 2002). Or, for any two tuples, t1 and t2, if t1[X] = t2[X], then t1[Y] = t2[Y], where t[X] represents the projection of t on X (Date, 2000; Elmasri & Navathe, 2000; Rob & Coronel, 2002). That is, whenever two tuples agree on their X values, they also agree on their Y values. A set of attributes X is a candidate key if all other attributes of the relation are fully functionally dependent on X (Codd, 1972; Hoffer, Prescott & McFadden, 2002; Rob & Coronel, 2002; Ullman & Widom, 1997).

An alternate definition of BCNF using the term superkey is that a relation, R, is in BCNF if whenever a nontrivial functional dependency X -> Y holds, then X is a superkey of R (Elmasri & Navathe, 2000; Dutka & Hanson, 1989; Ullman & Widom, 1997). A set of attributes is a *superkey* of a relation if those attributes functionally determine all other attributes of the relation (Rob & Coronel, 2002; Ullman & Widom, 1997). The functional dependency X -> Y is nontrivial if Y is not a subset of X. This paper uses the earlier definition of BCNF that is simpler to apply. As shown later in this section, the results would be the same using both definitions since they state the same concept.

Is ASSIGNMENT in BCNF? To answer this question, we check whether ASSIGNMENT qualifies as a relation, and if it does, whether every determinant of ASSIGNMENT is a candidate key. First, ASSIGNMENT is a relation since it has all the properties of a relation. In particular, it meets the important requirement that there are no duplicate tuples, since every tuple has a unique value for ID or COMPUTER_NO.

Second, both ID and COMPUTER_NO are determinants. Attribute ID is a determinant since the functional dependency ID -> {NAME, TITLE} holds. By assumption, every employee has a unique identification number represented by ID that determines the name and title. The left hand side of the dependency, ID -> {NAME, TITLE}, is null only in the trivial case when the right hand side also is null. Attribute COMPUTER_NO is another determinant since the functional dependency, COMPUTER_NO -> {MODEL, RAM}, holds. There are no other determinants.

If ID and COMPUTER_NO are determinants, are they also candidate keys? ID is a candidate key if all other attributes of ASSIGENEMNT are fully functionally dependent on ID. We already established that ID -> {NAME, TITLE}. Does the functional dependency ID -> {COMPUTER_NO, MODEL, RAM} hold when ID may be null in some tuples in which the attributes on the right hand side, COMPUTER_NO, MODEL, and RAM, are not null? For every valid instance of ID, the value of ID uniquely determines the values of COMPUTER_NO, MODEL, and RAM. That is, every employee has a unique computer, if the employee has one. Further, ASSIGNMENT satisfies the requirement that whenever two tuples agree on their ID values they also agree on their values of COMPUTER_NO, MODEL, and RAM. Thus, ID -> {COMPUTER_NO, MODEL, RAM} holds even though ID may be null in certain tuples. Hence, ID is a candidate key of ASSIGNMENT. Whether a functional dependency holds when the left-hand side may be null but the right hand side is not is discussed in more detail in the section entitled “The Solution”.

If ID is a candidate key, for similar reasons, COMPUTER_NO also is a candidate key. Thus, all the determinants of ASSIGNMENT are candidate keys. Hence, ASSIGNMENT is in BCNF. The same result is obtained if the alternate definition of BCNF based on the term superkey is used. Since ID and COMPUTER_NO are the only determinants, for every nontrivial functional dependency, X->Y, the left-hand side X includes ID or COMPUTER_NO. But, ID and COMPUTER_NO also are candidate keys. Thus X is a superkey, leading again to the conclusion that ASSIGNMENT is in BCNF though it suffers from insertion and deletion anomalies that can be removed by decomposition. Third normal form (3NF) and BCNF were introduced to eliminate such anomalies.

To determine whether ASSIGNMENT is also in 4th and 5th normal forms (4NF and 5NF), we will use simple criteria developed by Date and Fagin (1992). These state that a relation is in 4NF if it is in BCNF and it contains some simple keys, and it is in 5NF if it is in BCNF and every key is simple. Using these criteria, ASSIGNMENT is in 5NF since it is in BCNF and every key is simple.

THE SOURCE OF THE PROBLEM

A root cause for the insertion and deletion problems is that ASSIGNMENT violates one of the necessary conditions to satisfy the entity integrity rule, which specifies that no component of the primary key should have nulls. A necessary condition to meet this requirement is that at least one candidate key of a relation should not have nulls. However, the definitions of BCNF, relation, functional dependency, determinant, or candidate key do not require the database designer to apply the principle of the entity integrity rule in determining whether a relation is in BCNF.

This example is not an isolated case. A sufficient condition under which a relation in BCNF suffers from insertion/deletion anomalies can be stated as follows:

A relation, R, that is in BCNF would suffer from insertion/deletion anomalies if the relation contains information on two entities, E_1 and E_2, including their identifiers, and their relationship when the relationship between the two entities is (zero-or-one)-to-(zero-or-one).

The reasoning presented earlier using ASSIGNMENT can be summarized for the general case. Let $\{A_1,A_2,\ldots,A_n, B_1,B_2,\ldots,B_n\}$ be the schema of R where $\{A_1,A_2,\ldots,A_n\}$ represents attributes of E_1, and $\{B_1,B_2,\ldots,B_n\}$ represents attributes of E_2. Let X be the identifier of E_1. Let Y be the identifier of E_2. Since the relationship between E_1 and E_2 is (zero-or-one)-to-(zero-or-one), the functional dependencies $X\text{->}\{A_1,A_2,\ldots,A_n, B_1,B_2,\ldots,B_n\}$ and $Y\text{->}\{A_1,A_2,\ldots,A_n, B_1,B_2,\ldots,B_n\}$ hold. Thus, X and Y are candidate keys and there are no other candidate keys. The (zero-or-one)-to-(zero-or-one) relationship between E_1 and E_2 means that both X and Y may have nulls, resulting in insertion/deletion anomalies irrespective of whether X, Y, or the combination of X and Y is selected as the primary key. However, relation R would be in BCNF when there are no additional functional dependencies among the non-key attributes of R, since X and Y are the only determinants. Thus, whether R is in BCNF or not in BCNF, it suffers from insertion/deletion anomalies when the relationship between E_1 and E_2 is (zero-or-one)-to-(zero-or-one). The above reasoning holds true even when entities E_1 and E_2 may have more than one identifier. The problem also exists in the more general case when a relation contains information on two or more entities, including their identifiers and their relationships when the relationships between each pair of entities is (zero-or-one)-to-(zero-or-one).

THE SOLUTION

To help detect the violation of the requirement that at least one candidate key should not have nulls, the database designer needs to consider this requirement explicitly in applying the definition of BCNF. It is tempting to suggest that the entity integrity rule be incorporated into the definition of either a relation or a candidate key. We first consider these two cases.

Entity Integrity Rule, Relations, and Candidate Keys

One way to ensure that a relation that is in BCNF will have at least one candidate key without nulls is to modify the properties of a relation by adding a new requirement that a relation should have at least one candidate key that does not have nulls. Based on this requirement, ASSIGNMENT and similar "relations" would not qualify as relations, indicating that they are not good designs. However, in many cases, only relations that are in 3NF or BCNF will meet the new requirement for a relation. For example, ASSIGNMENT would have to be decomposed to two normalized relations (EMPLOYEE, COMPUTER) before it meets the new requirement. As a second example, consider a relation, CUST_ORDER:

CUST_ORDER (ORDER_ID, ORDER_DATE, CUST_ID, CUST_NAME, CUST_PHONE)

A customer may have zero, one, or many orders. An order belongs to exactly one customer. Here, ORDER_ID is the only candidate key. Since a customer may not have an order, the candidate key may have nulls in some tuples. Thus, CUST_ORDER is not a relation, based on the new requirement that at least one candidate key of a relation must not have nulls. If CUST_ORDER is not a relation, then the designer cannot apply the definition of BCNF to normalize it. Obviously, decomposing CUST_ORDER into two normalized relations, ORDER_HEADER (ORDER_ID, ORDER_DATE, CUSTR_ID) and CUSTOMER (CUSTR_ID, CUST_NAME, CUST_PHONE), would make both relations meet the new requirement for a relation. But that would mean normalizing the database before applying the normalization rules. Hence, incorporating the entity integrity rule into the definition of a relation is not a satisfactory solution.

A second way to ensure that a relation that is in BCNF will have at least one candidate key without nulls is to modify the definition of candidate key by adding the requirement that no component of a candidate key should have nulls. To be consistent, this requirement also should apply to determinants; that is, no component of a determinant should have nulls. This would mean that in the functional dependency X->Y, required for X to be a determinant or candidate key, no component of X can be null, except in the trivial case when Y is null. That is, for every valid instance of X, the value of X uniquely determines the value of Y, *and for every valid instance of Y, there is a corresponding value of X.* Under this requirement, ID and COMPUTER_NO still are determinants. However, ID is not a candidate key. In some tuples, ID may be null when {COMPUTER_NO, MODEL, RAM} is not null. Hence, under the new definition of candidate key, ASSIGNMENT is not in BCNF, consistent with the fact that it suffers from insertion/deletion anomalies. However, the additional requirement for candidate keys results in inconsistencies in applying BCNF in certain cases as discussed below.

Consider two relations, ASSIGNMENT_1, and ASSIGNMENT_2, both of which have the same attributes as ASSIGNMENT, but differ in a basic assumption regarding the relationship between employees and computers. For ASSIGNMENT_1, it is assumed that every computer is assigned to an employee, but an employee may not have a computer. Thus, ID does not contain any nulls, but COMPUTER_NO may be null in certain tuples. For ASSIGNMENT_2, it is assumed that every computer is assigned to an employee, and every employee has a computer. Hence, ID and COMUPTER_NO do not contain any nulls. In both relations, if ID is selected as the primary key, then there is no insertion anomaly. The result of deleting an employee's information also is the same in both relations. The corresponding computer has to be re-assigned to another employee, since, by assumption, every computer is assigned to an employee. If candidate keys are not allowed to have nulls, then COMPUTER_NO in ASSIGNMENT_1 is not a candidate key, while it is a candidate key in ASSIGNMENT_2. This means that ASSIGNMENT_1 is not in BCNF while ASSIGNMENT_2 is in BCNF, though insertion and deletion are identical in both relations. Thus, requiring that candidate keys should not have nulls, or that the left-hand side of the functional dependency should not be null when the right hand side is not null, leads to inconsistent results in applying BCNF. Hence, candidate keys may have nulls. This is in agreement with the popular view in the literature that candidate keys can have nulls, as pointed out by Date (2000, p. 595): "…alternate keys can apparently have nulls allowed". This conclusion supports our earlier determination that ID -> COMPUTR_NO, MODEL, RAM holds though ID may be null in certain tuples where COMPUTR_NO, MODEL, and RAM are not null.

Incorporate Entity Integrity Rule into BCNF

A third and recommended option is to apply the requirement that at least one candidate key of the relation should not have nulls, as part of checking whether a relation is in BCNF. To help the designer do this, this requirement is incorporated into the definition of BCNF. The modified definition of BCNF is:

A relation is in BCNF if, and only if, 1) every determinant is a candidate key, and 2) ***at least one of the candidate keys does not have any nulls****.*

The additional requirement that at least one of the candidate keys does not have any nulls is an essential pre-requisite to satisfy the entity integrity rule. Incorporating this requirement into BCNF forces the designer to explicitly apply the essence of the entity integrity rule without selecting a primary key.

Now we examine the effect of modifying the definition of BCNF on normalization of different relations. Applying the modified definition of BCNF leads to the conclusion that ASSIGNMENT is not in BCNF, since both candidate keys, ID and COMPUTER_NO, contain nulls. This conclusion is consistent with the fact that ASSIGNMENT suffers from insertion/deletion anomalies that can be removed by decomposition. Under the new definition, both ASSIGNMENT_1 and ASSIGNMENT_2 are in BCNF since the candidate key ID does not contain any nulls. This is consistent with the fact that both relations do not suffer from insertion and deletion anomalies, as discussed earlier. The designs of ASSIGNMENT_1 and ASSIGNMENT_2, of course, may not be desirable. Combining two entities into a single relation lacks intuitive appeal. Deleting an employee's information results in the consistently cumbersome process of reassigning the computer to another employee. These areas, however, are not meant to be covered by normalization.

Next, we examine whether the additional requirement incorporated into BCNF falsely classifies a relation as not in BCNF when it does not have any insertion, deletion, or update anomalies. The additional requirement that at least one candidate key of a relation should not have nulls affects the normalization of only those relations that have nulls in all candidate keys. Such relations will have nulls in the primary key, resulting in problems in inserting and possibly in deleting certain tuples. Thus the additional requirement affects only those relations that have at least insertion problems.

It should be noted that current decomposition algorithms that split relations into BCNF may not handle nulls adequately. Dealing with such cases is an area of further research.

Other Relations with Nulls in Candidate Keys

Relations that represent only one entity also may have nulls in all candidate keys and suffer from insertion problems. These relations will be classified as not in BCNF if the new definition of BCNF is applied. Two examples illustrate the two cases: 1) A relation has multiple candidate keys; 2) A relation has a composite candidate key.

The first example involves a relation that represents a single entity, and has multiple candidate keys. A relation, VISITOR, represents visitors to a country from a neighboring country:

VISITOR(PASSPORT_NO, NATIONAL_ID, NAME, ADDRESS).

Each visitor is required to have a unique passport number or a unique national identification number. Thus, a visitor may have a PASSPORT_NO, or a NATIONAL_ID, or both. NAME, ADDRESS, or a combination of the two, is not unique. VISITOR qualifies as a relation, since no two tuples can be identical. However, in its current form, VISITOR is not a good design. If PASSPORT_NO is selected as the primary key, then information on visitors without PASSPORT_NO cannot be inserted. Selecting NATIONAL_ID or the combination PASSPORT_NO + NATIONAL_ID as the primary key also has similar insertion problems. However, unlike ASSIGNMENT, this insertion problem cannot be eliminated by decomposition of VISITOR by taking projections. Hence, BCNF is not expected to identify this problem, but there is no harm if it does. Applying the current definition of BCNF, VISITOR is in BCNF. Under the modified definition of BCNF, VISITOR is not in BCNF. Thus, in such relations, the modified definition would result in identifying some insertion anomalies that cannot be removed by decomposition. Once the insertion problem is identified, the problem may be fixed by adding a new attribute VISITOR_ID, for example, as a surrogate key. A second option that merits further investigation is preventing the creation of relations like VISITOR by modifying the mapping rules.

The second example uses a relation that represents a single entity, and has a single composite candidate key. The relation, COURSE, represents different courses offered by an organization: COURSE (COURSE_NAME, DATE, INSTRUCTOR).

There are two types of courses: 1) one-day traditional classroom courses, and 2) Internet courses. COURSE_NAME represents the unique name for a course, and DATE represents offering date of the course. Internet courses are available all the time. Hence, the attribute, DATE, is not applicable for Internet courses. By assumption, COURSE_NAME + DATE is the only candidate key. VISITOR is not a good design. If COURSE_NAME + DATE is selected as the primary key, then information on Internet courses, which do not have a value for DATE, cannot be inserted. Again, this insertion problem cannot be removed by decomposition of COURSE by taking projections. Under the modified definition of BCNF, COURSE is not in BCNF since it does not have a candidate key without nulls.

Thus, a somewhat harmless and potentially beneficial side effect of the additional requirement in the modified BCNF is that it identifies certain insertion anomalies that cannot be removed by decomposition. The major benefit of incorporating the additional requirement is that in relations that represent two entities with a one-to-one relationship between them, the modified definition helps to identify insertion/deletion anomalies that can be eliminated by decomposition, while the current definition may not help to identify them.

THIRD NORMAL FORM

A relation that is in BCNF also should be in third normal form (3NF). Using the current definition of BCNF, ASSIGNMENT is in BCNF. Is it also in 3NF? One group of definitions uses the term primary key. An example is: *A relation is in 3NF if it is in second normal and no nonprime attribute is transitively dependent on the primary key* (Elmasri & Navathe, 2000). Applying this group of definitions leads to the conclusion that ASSIGNMENT is not in 3NF, or at least it is not possible to determine whether it is in 3NF, since it does not have a valid primary key. Thus, this definition of 3NF is able to identify the insertion/deletion anomaly problems that are not detected by BCNF. Hence, these definitions of 3NF do not require any change.

A more general definition that does not use the term primary key is: A relation is in 3NF if whenever a nontrivial functional dependency X -> A holds, then either X is a superkey of R, or A is a prime attribute (Elmasri & Navathe, 2000). An attribute is a prime attribute if it is part of a candidate key. Thus, the only difference between 3NF and BCNF is that in 3NF, the right hand side of the functional dependency is allowed to be a prime attribute. Since ASSIGNMENT is in BCNF, it is also in 3NF under the relaxed requirements. Hence, this definition of 3NF needs to be modified:

A relation is in 3NF if whenever a nontrivial functional dependency X -> A holds, then 1) either X is a super key of R, or A is a prime attribute, and ***2) at least one of the candidate keys does not have any nulls.***

DOMAIN KEY NORMAL FORM

An alternative or a supplement to using the traditional normal forms is the Domain-Key Normal Form (DKNF) proposed by Fagin (1981) as the ideal or ultimate normal form. A relation is in DKNF if every constraint (including dependencies) can be inferred by simply knowing the attributes and their domains, and the set of keys. Thus, if a relation schema is in DKNF, then the DBMS should be able to enforce all constraints of the relation schema by enforcing the domain and key constraints. A relation schema is defined to be a set of attributes, along with their constraints. DKNF has the conceptual superiority that it is based on the primitive concepts of domains and keys, whereas traditional normal forms are based on functional, multivalued, or join dependencies. However, DKNF is not popularly used by practitioners due to practical limitations, including the lack of simple well-defined methods to achieve DKNF. Hence, it is important that the traditional normal forms be able to correctly identify design problems, irrespective of whether DKNF can identify them. Next, we examine whether DKNF identifies the problems with the design of ASSIGNMENT.

Is ASSIGNMENT in DKNF? It is assumed that the relation schema of ASSIGNMENT has only simple constraints: Attribute RAM must be an integer, and all other attributes are character strings with a specified limit on length. These constraints can be enforced by enforcing the domain constraints that can be imposed by the DBMS on individual attributes.

The schema of ASSIGNMENT also includes the constraints represented by the two functional dependencies:

ID -> {NAME, TITLE, COMPUTER_NO, MODEL, RAM} and
COMPUTER_NO -> {ID, NAME, TITLE, MODEL, RAM}.

These constraints can be enforced by the DBMS by enforcing the key constraints on ID and COMPUTER_NO. A key is defined as an attribute such that no two tuples have the same value for the attribute. Thus, if there are no other constraints, all constraints can be enforced by domains and keys, implying that ASSIGNMENT is in DKNF.

An additional real-world constraint that may not be evident could be that, for each tuple, either ID or COMPTER_NO must not be null. Since each employee has an ID and each computer has a COMPUTER_NO, it would be unrealistic to have a tuple that has nulls in ID and COMPUTER_NO. Furthermore, if ID and COMPUTER_NO can be null in the same tuple, then there could be two or more such tuples with identical values for the rest of the fields which are not required to be unique. Under such conditions, it is not clear that ASSIGNMENT meets the requirement for a relation that there are no duplicate tuples, since comparison of two nulls evaluate to the "unknown" truth value (Date, 2000). The constraint that either ID or COMPUTER_NO must not be null cannot be enforced by specifying the domain for individual attributes, or, by enforcing key constraints on ID, COMPUTER_NO, or ID+COMPUTER_NO, implying that ASSIGNMENT is not in DKNF.

The reason why ASSIGNMENT appears to be not in BKNF is the existence of a single constraint involving two attributes (ID and COMPUTER_NO cannot be null in the same tuple). It is not the existence of any functional dependency that cannot be implied by a key. The constraint involving ID and COMPUTER_NO may not be readily evident to the designer, making it difficult to identify the design problem in ASSIGNMENT by applying DKNF. If the DBMS can enforce the constraint that either ID or COMPUTER_NO must not be null in each tuple, then ASSIGNMENT would be in a weaker normal form that is based on the concept of DKNF, though ASSIGNMENT is not a good design.

Applying DKNF to the relation COURSE (COURSE_NAME, DATE, INSTRUCTOR) presented earlier yields the result that COURSE is in DKNF, though COURSE has insertion problems. Here, the only constraint is the functional dependency, COURSE_NAMEA+DATE -> INSTRUCTOR. This constraint can be enforced by specifying COURSE_NAMEA+DATE as a key.

In order to make it easy to identify design problems in relations like ASSIGNEMNT and COURSE it would be desirable to incorporate into the definition of DKNF the principle implied by the entity integrity rule that at least one candidate key must not have nulls:

*A relation is in DKNF if 1) every constraint (including dependencies) can be inferred by simply knowing the attributes and their domains and the set of keys, **and 2) at least one of the keys does not have any nulls**.*

With the above definition, it is easy to see that all three relations ASSIGNMENT, VISITOR, and COURSE presented earlier do not meet the modified requirement for DKNF. This result is consistent with the fact that they are not good designs since they suffer from at least insertion anomaly as defined by Codd (1972).

SUMMARY

This chapter has shown some of the problems in applying the normalization theory when all the candidate keys of a relation have nulls, but in each tuple, at least one candidate key has a unique value. Applying the current definitions of BCNF or DKNF to such relations may not help the designer to detect insertion and/or deletion anomalies that are associated with poor designs. A basic problem is that there is nothing in the definition of a relation, BCNF or DKNF, that guarantees that a normalized relation satisfies the entity integrity rule. Three possible solutions to the problem were considered: 1) Modify the definition of candidate key to include the requirement that a candidate key should not have nulls; 2) Incorporate the essence of the entity integrity rule into the definition of a relation; 3) Incorporate the essence of the entity integrity rule into the definitions of BCNF and DKNF. It is shown that the first two solutions have negative side effects. The third method provides a solution to the problem without creating such side effects. In essence, the modified definition guarantees that a relation that is in BCNF or DKNF will have at least one candidate key that does not have nulls. This, in turn, helps to eliminate the insertion/deletion anomalies caused by nulls in the primary key.

REFERENCES

Codd, E.F. (1972). Further normalization of the database relational model. *Database systems, Courant Computer Science Symposia series, 6,* 34-64. Englewood Cliffs, NJ: Prentice Hall.

Codd, E.F. (1986). Missing information (applicable and inapplicable) in relational databases. *SIGMOD Record*, *15*(4), 53-74.

Connolly, T., & Begg, C. (2002). *Database systems*. Reading, MA: Addison-Wesley.

Date, C.J. (1990). NOT is not "Not"! (Notes on three-valued logic and related matters. In C.J. Date. *Relational database writings, 1985 – 1989* (427– 450). Reading, MA: Addison-Wesley.

Date, C.J. (2000). *An introduction to database systems*. Reading, MA: Addison-Wesley.

Date, C.J., & Fagin, R. (1992). Simple conditions for guaranteeing higher normal forms in relational database systems. *ACM Transactions on Database Systems*, *17*(3), 465-476.

Dutka, A.F., & Hanson, H.H. (1989). *Fundamentals of data normalization*. Reading, MA: Addison-Wesley.

Elmasri, R., & Navathe, S. (2000). *Fundamentals of database systems.* Reading, MA: Addison-Wesley.

Fagin, R. (1981). A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems*, *6*(3), 387-415.

Hoffer, J.A., Prescott, M.B., & McFadden, F.R. (2002). *Modern database management.* Reading, MA: Addison-Wesley.

Imielinski, T., & Lipski, W. (1984). Incomplete information in relational databases. *Journal of the ACM*, *31*(4), 761-791.

Kroenke, D.M. (2002). *Database processing*. Upper Saddle River, NJ: Prentice-Hall.

Levene, M., & Loizou, G. (1999). Database design for incomplete relations. *ACM Transactions on Database Systems*, *24*(1), 80-126.

Lipski, W. (1979). On semantic issues connected with incomplete information databases. *ACM Transactions on Database Systems*, *4*(3), 262-296.

Reiter, R. (1986). A sound and sometimes complete query evaluation algorithm for relational databases with null values. *Journal of the ACM, 33*(2), 349-370.

Sutton, D., & King, P. (1995). Incomplete information and the functional model. *The Computer Journal*, *38*(1), 31-41.

Ullman, J.D., & Widom, J. (1997). *A first course in database systems*. Upper Saddle River, NJ: Prentice Hall.

Vardi, M.Y. (1986). On the integrity of databases with incomplete information. *Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, 252-256.

Vassiliou, Y. (1980). Functional dependencies and incomplete information. *Proceedings of the International Conference On Very Large Databases*, 260-269.

Watson, R.T. (2002). *Data management.* New York: Prentice Hall.

Chapter VIII

Regression Test Selection for Database Applications

Ramzi A. Haraty, Lebanese American University, Lebanon

Nashat Mansour, Labanese American University, Lebanon

Bassel A. Daou, University of Ottawa, Canada

ABSTRACT

Database applications features such as Structured Query Language programming, exception handling, integrity constraints, and table triggers pose difficulties for maintenance activities, especially for regression testing that follows modifying database applications. In this chapter, we address these difficulties and propose a two-phase regression testing methodology. In phase 1, we explore control flow and data flow analysis issues of database applications. Then, we propose an impact analysis technique that is based on dependencies that exist among the components of database applications. This analysis leads to selecting test cases from the initial test suite for regression testing the modified application. In phase 2, we propose two algorithms for reducing the number of regression test cases. The Graph Walk algorithm walks through the control flow graph of database modules and selects a safe set of test cases to retest. The Call Graph Firewall algorithm uses a firewall for the inter-procedural level. Our experience with this regression testing methodology shows that the impact analysis technique is adequate for selecting regression tests and that phase 2 techniques can be used for further reduction in the number of these tests.

INTRODUCTION

Software maintenance involves changing programs due to errors, alterations in user requirements or changes in the hardware/software environment. Regression testing is an important activity of software maintenance, which ensures that the modified software still satisfies its intended requirements (Hartmann & Robson, 1989). It attempts to revalidate modified software and ensure that new errors are not introduced into the previously tested code. Regression testing involves four issues: change impact identification, test suite maintenance, test strategy, and test case selection. Change impact identification involves locating all the modules and other program segments that are affected by the modification. Test suite maintenance attempts to keep the test suite status current and reusable for future revalidation. Test strategy involves finding a test sequence for retesting the software. Test case selection attempts to reduce the cost of regression testing by selecting a subset of the test suite that has been used during the application development. This subset of tests is then used to test modified programs (Rothermel & Harold, 1998).

In database applications a number of new features are supported, such as Structured Query Language (SQL) statements, table constraints, exception handling, and table triggers. These features introduce new difficulties that hinder regression test selection. In this work, we concentrate on impact analysis and test selection for SQL-based systems. Regression testing is necessary for assuring the quality of a system after modifying it. Ad hoc regression testing involves either rerunning all the test cases that are included in the test suite determined during the initial development of software (Select-All approach) or selecting a random subset of this initial test suite (Select-Random approach). But, the Select-Random approach is unreliable, since it might miss selecting test cases that reveal adverse effects of modifications. Hence, the Select-Random approach might compromise the quality of the modified system. On the other hand, the Select-All approach is expensive in terms of time and cost, since it usually includes many test cases that do not reveal the impact of the modification made to the system. Therefore, it is important to use regression testing methods that reduce the number of selected test cases in order to save time and money, especially for large software systems, while maintaining the quality of the system (Wong et al., 1997).

SQL, the standard query language, is a declarative language used for the manipulation of table data in database applications. It stands as the heart of database applications modules (ISO/IEC 9075, 1992). The usage of SQL in a procedural context has its implications. We categorize these implications into three categories: control dependencies, data flow dependencies, and component dependencies. The nature of SQL and the existence of table constraints lead to using exception handling techniques in database modules. Exception handling complicates control flow dependencies between statements in database modules. This complexity should be handled in the process of applying control flow-based regression testing techniques. Moreover, table triggers firings because of modifying SQL statements create implicit inter-modular control flow dependencies between modules. These dependencies should be explored for performing inter-module regression testing.

The manipulation of database tables by different modules, using SQL, leads to a state-based behavior of modules. It also creates data flow dependencies between the modules. The dynamic behavior of SQL, in which the exact table rows manipulated is not known until run-time, makes it very difficult to trace such data dependencies. Furthermore, SQL manipulates database components such as tables and views. These facts create component dependencies between the various components handled by SQL statements and the modules in which the statements are located. These component dependency relations are transitive. Whenever a change is made to one component, this transitivity introduces a ripple effect of change.

In this chapter, we propose a new two-phase methodology for regression testing SQL-based database applications. Phase 1 involves detecting modifications and performing change impact analysis. The impact analysis technique localizes the effects of change, identifies all the affected components, and selects a preliminary set of test cases that traverse modified components. Phase 2 involves running a test case reduction algorithm to further reduce the regression test cases selected in phase 1. We present two such algorithms. The first algorithm, Graph Walk, is a control flow-based regression testing technique that utilizes control flow information, component dependencies, and impact analysis results. The second algorithm, Call Graph Firewall, utilizes data flow dependencies and is an adaptation of firewall-based regression testing techniques at the inter-procedural level. Furthermore, we develop a prototype maintenance tool and use it to empirically validate our proposed methodology.

The remainder of this chapter is organized as follows: The next section includes a discussion of the structure of database applications and control flow issues of database modules. This is followed with a section addressing the data-flow dependencies due to the manipulation of data stored in database tables. Next, we present the impact analysis. We then present the test case reduction algorithms and empirically investigate the applicability of the methodology using the tool. Finally, we present related work and conclude the chapter.

CONTROL FLOW MODELING

Background

Database systems have been accepted as a vital part of the information system infrastructure. They can be considered a mature technology whose characteristics have been covered in past manifestos. Although there are different variations of database systems implementation, we will limit our scope to relational database systems because relational database systems are widespread and the relational concepts are standardized.

SQL remains the most accepted and implemented interface language for relational database systems. SQL is designed to be a comprehensive language that includes statements for data definition, queries, updates, and view definition.

Lately, extensions to the SQL language were introduced. These extensions allow client (application program) requests to the server to perform lengthy, complex operations, with only the final results returned to the client. These SQL extensions were in the form of stored procedures and procedural language constructs that allowed significant application logic to be stored and executed on the server instead of on the client. Persistent Stored Modules (PSMs) were published as an international standard in the form of a new part to the SQL-92 standard. This standard—ISO/IEC 9075-4 (which appeared in 1995)—was an extension to standard SQL for procedural language constructs, based on the best language concepts. Control flow analysis of PSM code is different from that of conventional programming languages. Building control flow graphs for database modules differs from building control flow graphs for conventional software. This difference results from the extensive usage of exceptions and condition handlers and the nature of the SQL language that is a key feature of database modules. Therefore, we should devise new modeling techniques to model the control transfers that are available in database modules.

The semantics of all SQL statements make them behave like micro-transactions in that they either execute successfully, or they have no effects at all on the stored data, as described in the ISO/IEC 9075 standard of 1992.

The SQL-PSM standard allows specifying one or more condition handlers for any given compound statement, as mentioned in the ISO/IEC 9075-4 standard of 1995. In general the handler action either handles the condition—in which case, the type of the handler determines the subsequent behavior of the compound statement containing the condition handler—or it leaves the condition unhandled, causing it to be propagated outward, either to another compound statement in which the compound statement containing the condition is nested or to the application that invoked the routine.

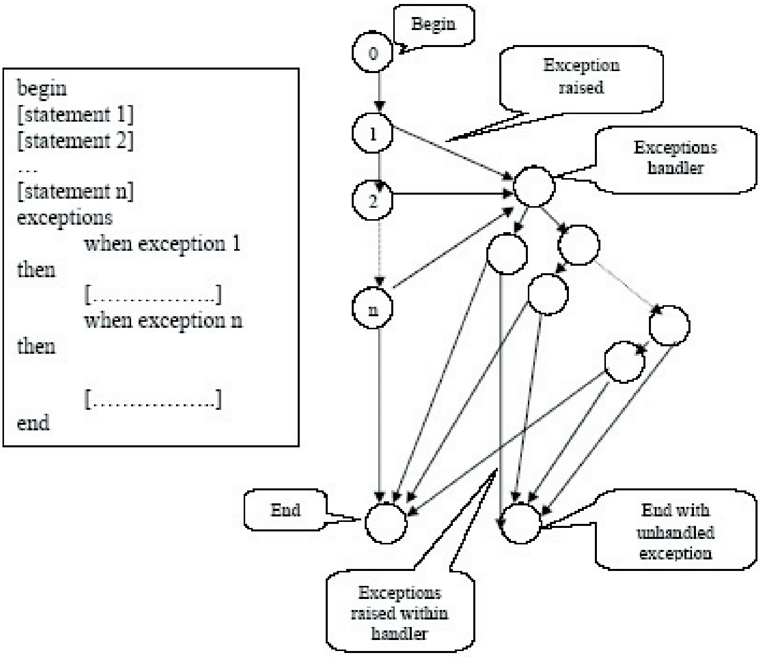
A database module consists of one compound statement in which other compound statements are nested. Each compound statement has its exception handler. Each condition has its compound statement. During execution, if an exception is raised from an SQL statement then the control is transferred from the current statement to the exception handler according to the type of the exception raised.

Suggested Technique

A node in the control flow graph should represent each statement. These statements are either SQL statements, control statements or others. Each node, especially those representing SQL statements, has two possible outcomes—either a success or a failure, with an exception raised. Nodes containing control statements have more than one success outcome (most of the time two outcomes, as in the case of *if* statements). The exceptions raised belong to a large list of possible exceptions that might be raised in the database environment, like duplicate value on index, value could not be null, or others. Because we cannot limit or predict the type of exception that could be raised by a statement, we prefer to represent the control transfers due to exceptions with one link. This link is later routed to the proper destination according to the type of the exception.

A compound statement contains a list of statements with one exception handler for all of these statements. A node represents each of these statements. The compound statement

Figure 1: Flow graph modeling of compound statements



```
begin
[statement 1]
[statement 2]
...
[statement n]
exceptions
          when exception 1
then
          [.................]
          when exception n
then

          [.................]
end
```

Begin
0
Exception raised
1
Exceptions handler
2
n
End
End with unhandled exception
Exceptions raised within handler

contains two end statements: one for successful endings, and the other for unhandled exception results. If exception handling is not available, then all the exception links of these nodes will be linked to the unhandled exception end node. If exception handling is available then a primary handler switch node to which all the exception links of the compound statement nodes are linked models the exception handler. A predicate node that checks for the type of the exception models each specific exception handler. The exception predicate has two links: the first one to the start node of the exception handler block and the second to the next handled exception. The primary handler switch is mainly the predicate node of the first exception handler. If the handler is the last in the block then the second link is made to the unhandled exception end node of the compound statement. Statements inside the handler block of each exception handler are modeled like other statements in the compound block. However, the exception link is directly linked to the unhandled exception end node of the compound statement. The end nodes of the exception handlers' blocks are linked to the successful end node. Figure 1 further explains modeling of control flow transfers of a compound statement.

The inner block is treated by the outer block like a single node with two outcomes expected. The nodes that have been linked to the success end should be linked to the node of the next statement in the outer block. Similarly, nodes that have been linked to the unhandled exception end node should now be linked to the primary handler switch of the outer block.

DATA FLOW MODELING

Background

Data flow testing methods focus on the occurrences of variables within the program. Each variable occurrence is classified as either definition occurrence or use occurrence. A definition occurrence of a variable is where a value is bound to the variable. A use occurrence to a variable is where the value of the variable is referenced. Each use occurrence is classified as being a computational use or predicate use. If the value of the variable is used to decide whether a predicate is true for selecting execution paths, the occurrence is a predicate use. Otherwise, it is used to compute a value for defining other variables or as an output value (Rapps & Weyuker, 1985).

The database plays an important role in holding the state of computation in database modules. Other statements in the same module or other modules use the data generated by a statement; thus creating data flow relations. The main source of data in a relational database is tables. The data in these tables are created, deleted, updated, or retrieved. The manipulation of this data is done through the use of SQL statements. Each table is composed of a fixed number of named fields or columns. A table consists of a set of records; each record has its own values in each column. The SQL data manipulation language handles in one statement a particular column value in a given row, some columns of a given row, or all the column values of a row. It can also handle, in one statement, a given column, a group of column values, or all the column values of a group of rows. It can also handle all the columns of all the rows in the table.

Traditionally, data flow dependencies are created through the manipulation of variables in which each variable is defined with a value that could be used later or rewritten. However, in database applications we are dealing with a set of tables with multi-columns and multi-rows. To define the data flow relations created from the database usage we should decide on a level of granularity of the database variables in which we can trace their definition, and subsequently, their use.

Suggested Technique

One choice of the level of granularity is to consider each table in the database as a variable and handle all types of table usages as either a definition of the table or a retrieval of values. However, most of the time only parts of a table are handled in a given SQL statement. Another level of granularity is to consider each row in the table as a separate variable and trace the data flow relations that exist from the usage of each row separately. This situation is similar to the problem of defining the control flow relations created by linked lists, where each node in the linked list is dynamically created, modified, and deleted. In the case of database tables each row is dynamically created, deleted, modified, or retrieved. This implies that we cannot statically identify the possible data flow relations that could exist between rows. This is because the row usage is determined by evaluating the restricting conditions of the SQL statement performing the data manipulation.

A more moderate solution between the table and row level of granularity is the column level. Since the number of columns is fixed and columns are used in SQL statements using their unique names, we can determine the column usage statically. A drawback of this choice is the fact that it does not discriminate between the usage of one particular column value belonging to some row and the usage of the same column but of a different row. Discriminating between such usages leads us back to the problem of row-level data flow dependencies.

SQL statements use columns directly and indirectly or, in other words, explicitly and implicitly. These usages are either definition or retrieval. A table participating in master detail relations has a group of its columns referencing the primary key columns of the master table. Whenever these columns are defined the database implicitly checks that the master table contains a record that has its primary key column values matching the foreign key column values of the newly added record. So, whenever a new record is created the primary key columns of the master table are used. Conversely, whenever a master record is deleted the detail tables are checked to see whether there exist records with foreign key column values matching with the primary key column values of the master record being deleted.

We differentiate between five main usages of database columns. They are: delete, insert, reference, select, and update. Reference and select usages are computational usages and are denoted as c-use. Update, delete, and insert usages are define usages and are denoted as d-use. However, notice that in all of the previous define categories the result of the definition is dependent on the initial values of the columns defined, because the columns contain multi-values and zero or more of its values retain their initial values. Therefore, whenever there is a define usage of a column there is also a computation usage.

The list of various cases of column usages includes:

1. Explicit usage
   a. Explicit retrieval
      i. In the selection list of SELECT SQL statements and SELECT sub-queries.

        - ii. In the condition of the SQL statements and SELECT sub-queries.
    - b. Explicit definition
        - i. INSERT column list.
        - ii. Columns to set in UPDATE statement.
2. Implicit usage
    - a. Implicit retrieval
        - i. Usage of * abbreviations to indicate whole table columns.
        - ii. Reference of master key column.
        - iii. Reference of detail foreign key column.
    - b. Implicit definition
        - i. Column of table used in DELETE statements.
        - ii. Columns not listed in the INSERT statement set to null value or given a default value if it is available.

IMPACT ANALYSIS

Software impact analysis estimates what will be affected in software or related documentation if a proposed software change is made (Arnold & Bohner, 1996). Impact analysis information can be used for planning changes, making changes, accommodating certain types of software changes, and tracing through the effects of changes. Impact analysis provides visibility into the potential effects of changes before the changes are implemented.

Although it is relatively easy to understand most of the database structures and modules, understanding their combined effect or combined functionality is difficult. The complex relationships between database objects make it difficult to anticipate and identify the ripple effects of changes. Data dependencies, control dependencies, and component dependencies make it difficult to generate tests to adequately retest the affected elements. Our impact analysis technique is based on a reverse engineering approach designed to extract the database components and their relationships. This information is used to automatically identify the changes and the effects of those changes. In this section, we present phase 1 of our regression testing methodology, which includes modification detection and impact analysis. In this phase we localize the effects of change, identify all affected components, and select a preliminary set of test cases that traverse modified components.

Change Identification

Change identification is the first step in change impact analysis. We differentiate between two types of changes in the database application environment:

(a) **Code Change**: This involves changes that can be made to the code of the database modules. This is similar to any change made to any module written in any other language. Addition, deletion, and modification to particular statements inside a module are examples of code change.

(b) **Database Component Change**: This change involves the changes that could be made to the definition of the database components in general. It also includes the changes that could be made to the definition of database modules.

Code change identification can be made at different levels. The more details we want, the more sophisticated the change identification tool should be. With more details, identifying change impact will be easier and more informative.

Change Impact Identification

A change made to one component affects other database components due to component dependencies. Therefore, to identify the impact of change, we should identify the dependencies that exist between database application components and then find the ripple effect of change due to the transitivity of the dependency relations. The Component Firewall technique presented below is used to determine all the affected database components.

Component Dependency

Each type of database objects is handled separately to determine the dependencies it creates. In Table 1, we give an example of the dependencies that exist between database components.

Component Firewall

A Component Firewall is a set of affected modules when some changes are made to any of the database components. A database component is marked as modified and is included in the Component Firewall if one of the following conditions is satisfied:

(a) Its definition is modified.
(b) It is deleted.
(c) It is dependent on a modified or deleted component.

Table 1: An example of database components dependency

Database Component	**Dependent Component**
Table	Primary key constraint
	Foreign key constraint
	Check constraint
	Index
	View
	Synonyms
	Trigger
	SQL-PSM code
	SELECT statement
	INSERT statement
	UPDATE statement
	DELETE statement

(d) It became dependent on new or modified components in the new system, such as triggers and constraints.

The requirements of implementation of the Component Firewall technique are:

(a) Modeling the dependency relations of both old and new schemas, and
(b) Determining the modified, deleted, and new components.

All database components selected by the Component Firewall algorithm are marked as affected components. Affected module components are determined in order to select the test cases that traverse them, which make up the results of phase 1.

In Figure 2, we sketch an outline of the Component Firewall building algorithm. This algorithm takes the old and new schemas and returns a list of components that construct the Component Firewall.

Module *Compare* is responsible for performing change identification. It takes the old and new database schemas and returns two lists of components: one for the modified and deleted components and the other for the newly added ones. Module *Transitive_Closure* takes a list of components and the database schema and returns the transitive closure of the dependent components. If the dependency relation is modeled using a directed graph, then the transitive closure could be computed through finding the components reachable from modified components using depth first search.

As an example, we present part of a database application used in a commercial bank to pay checks drawn on accounts maintained by the bank. The bank keeps track of customer

Figure 2: The Component Firewall algorithm

```
Component_Firewall(old_schema, new_schema)

Denote L to be the list of components in the firewall.
Denote ML to be the list of modified and deleted components.
Denote NL to be the list of new components.

Compare(old_schema, new_schema, ML, NL)
For each modified component C in ML
    Add C to L
    For each component X dependent on C in new_schema
        If X belongs to old_schema then
            Add X to L

For each new component C in NL
    For each dependent component X on C in new_schema
        If X belongs to old_schema then
            Add X to L

L := Transitive_Closure(L, old_schema)

Return L
```

accounts and ensures that issued checks are not reported stolen. Modules *Pay_Check*, *Stolen_Check*, and *Update_Balance* are used to implement this functionality.

The tables used in this example are:

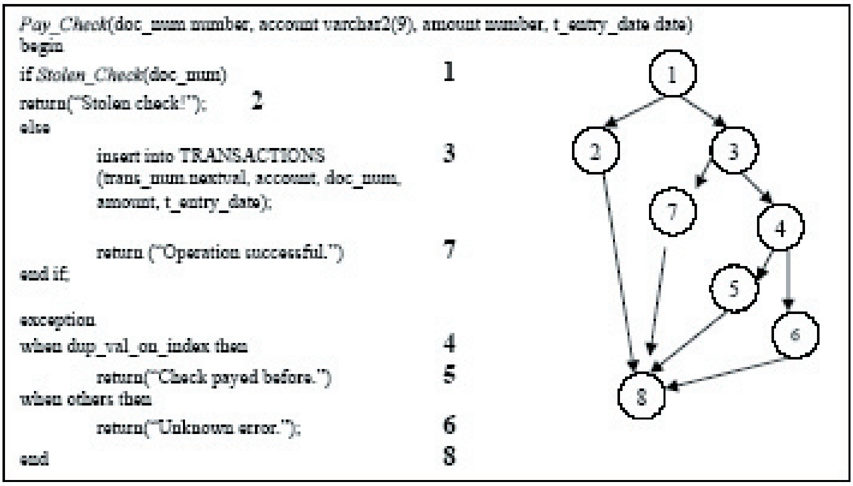
- **ACCOUNTS** (acc_number, acc_name, balance, status): holds the accounts' information.
- **TRANSACTIONS** (trans_num, account, doc_num, amount, t_entry_date, value_date): used to keep track of paid checks.
- **STOLEN_CHECKS** (check_num, s_entry_date): used to register the numbers of the checks reported stolen.

Figure 3 depicts module *Pay_Check* and its control flow graph. The code is written in PL/SQL Oracle's implementation of SQL-PSM. The circles denote nodes in the control flow, and arrows denote possible control transfer. This control flow analysis is statement based, where a node represents each statement.

In this control flow analysis, the inter-module extensions are not presented, although in this module there are implicit and explicit calls to other modules. Statement 1 calls function *Stolen_Check*, and statement 3 triggers the firing of *Update_Balance* trigger, which is set on the TRANSACTIONS table. The component usages are not shown as well. Statement 3 explicitly uses table TRANSACTIONS and columns trans_num, account, doc_num, amount, and t_entry_date. Statement 3 implicitly uses column value_date that is not present in the insert list. It implicitly uses table ACCOUNTS by reference since column account is part of a foreign key referencing table ACCOUNTS. This usage implies that table ACCOUNT is checked to see whether the account record being used exists.

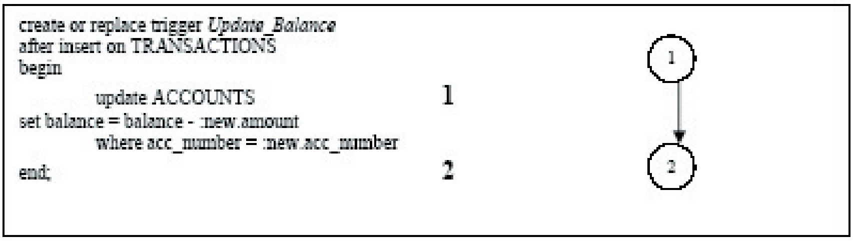
Figures 4 and 5 list trigger *Update_Balance* and function *Stolen_Check* respectively and the control flow graph of each module.

Figure 3: Function Pay_Check and its corresponding control flow graph



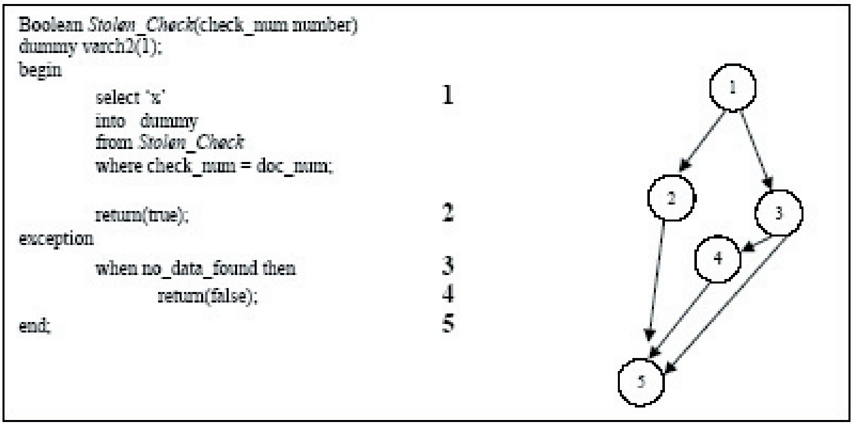
```
Pay_Check(doc_num number, account varchar2(9), amount number, t_entry_date date)
begin
if Stolen_Check(doc_num)                                    1
return("Stolen check!");          2
else
            insert into TRANSACTIONS                        3
            (trans_num.nextval, account, doc_num,
            amount, t_entry_date);

            return ("Operation successful.")                7
end if;

exception
when dup_val_on_index then                                  4
            return("Check payed before.")                   5
when others then
            return("Unknown error.");                       6
end                                                         8
```

Figure 4: Trigger Update_Balance and its corresponding flow graph



```
create or replace trigger Update_Balance
after insert on TRANSACTIONS
begin
            update ACCOUNTS                         1
set balance = balance - :new.amount
            where acc_number = :new.acc_number
end;                                                2
```

Figure 5: Function Stolen_Check and its corresponding control flow graph



```
Boolean Stolen_Check(check_num number)
dummy varch2(1);
begin
            select 'x'                              1
            into   dummy
            from Stolen_Check
            where check_num = doc_num;

            return(true);                           2
exception
            when no_data_found then                 3
                     return(false);                 4
end;                                                5
```

Figure 6: Component dependency graph

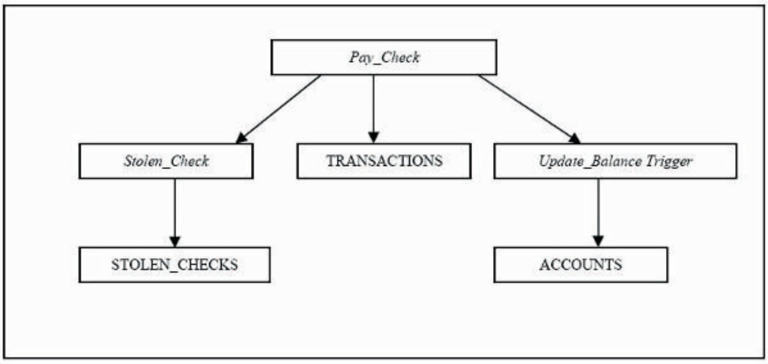


Table 2: Test cases and the statement test trace

Test Cases	Statement Trace		
	Pay_Check	*Stolen_Check*	*Update_Balance*
T1	1, 2, 8	1, 2, 5	
T2	1, 3, 7, 8	1, 3, 4, 5	1, 2
T3	1, 3, 4, 5, 8	1, 3, 4, 5	
T4	1, 3, 4, 5, 6, 8	1, 3, 4, 5	1, 2
T5		1, 2, 5	
T6			1, 2
T7		1, 3, 4, 5	
T8			1, 2

Figure 6 summarizes the dependency relations existing between the various components. Next, we demonstrate how the Component Firewall algorithm works by proposing a set of changes to the system. In Table 2 we define part of the testing scenarios and test cases that have been used during the testing of the modules in this example.

Case 1

Change the definition of trigger *Update_Balance*. Instead of "after insert", change it to "after delete". As a result of this modification, component *Pay_Check* is not any more dependent on component *Update_Balance*. Consequently, all the test cases passing through it are included in the retest list in addition to the test cases passing through component *Update_Balance*. Thus, the retest list will consist of test cases T1, T2, T3, T4, T6, and T8.

Case 2

Add a not-null constraint to value_date column in table TRANSACTIONS. This column is not listed within the insert list of the insert statement in module *Pay_Check*. However, in our impact analysis we include this column as an implicit usage since it is set to null. By adding not-null constraint to this column, the insert statement is using implicitly a modified component. Therefore, component *Pay_Check* will be affected because it is dependent on a modified component. Consequently, all test cases passing through it are included in the retest list. These are test cases T1, T2, T3, and T4.

Case 3

Statement 2 in function *Stolen_Check* is to be modified. The statement, return(true), becomes return(false). Therefore, component *Stolen_Check* is affected and component *Pay_Check* that is dependent on it is marked as affected, and consequently all test cases passing through these affected components are included in the retest list. These are test cases T1, T2, T3, T4, T5, and T7.

TEST CASE REDUCTION ALGORITHMS

In the impact analysis phase (phase 1), the test cases traversing the modules that are included in the firewall are selected for regression testing. However, this results in a relatively large number of selected test cases, since the Component Firewall does not distinguish between the modification-revealing test cases and the non-modification revealing ones. Therefore, it is useful to explore new techniques to reduce the number of test cases selected in phase 1 by concentrating on modification revealing tests. In this section, we discuss two such techniques. The first technique is the Graph Walk technique. This technique works when statement trace and statement components usages are available. In addition, it works at both the inter-module level and intra-module level. The second technique is Call Graph Firewall. It is an adaptation of the firewall regression testing technique proposed by Leung and White for procedural programs (Leung & White, 1990a, 1990b). It works at the inter-module level, utilizing the Call Graph of the database application and selecting test cases based on the data flow dependency resulting from the various usages of database tables.

Graph Walk Technique

In the Graph Walk technique, we use control flow graphs of all modules in the application and its modified version, and trace-information linked to control flow nodes. We also utilize the dependency created between statements and various database components.

Applying this technique to a module, we traverse the control flow of the module and its modified version. When a pair of nodes N and N* in the graphs of the original module and its modified version are discovered (i.e., the statements associated with N and N* are different), this technique selects all tests from the test suite that reach N in the original program. For two nodes N and N* to be different, at least one of the following conditions must be satisfied:

(a) N and N* are lexically different,
(b) N uses a modified component,
(c) N uses a component that is not used by N*, or
(d) N* uses a component that is not used by N.

To extend the technique to the inter-module level, we should change condition (b) to become: N uses a modified non-module component. Moreover, for each module call linked to a control flow graph node N we should perform the Graph Walk algorithm recursively on this module and intersect the result with the test cases passing through node N.

In Figure 7 we give the *Graph Walk* algorithm. It takes two modules as parameters and returns a list of test cases to retest. This algorithm is based on the *Compare* algorithm that works on the control flow graph nodes. The *Compare* algorithm takes two control flow nodes: one from the original module and the other from the modified version, and it returns the test cases passing through the original node that should be retested. The *Compare* algorithm is presented in Figure 8. It calls the *Is_Different* algorithm that checks whether two control flow nodes satisfy one of the four conditions listed earlier. Figure 9 gives the details of the *Is_Different* algorithm.

To optimize the performance of the Graph Walk in the inter-module level, we save the results of the *Graph Walk* algorithm for each module. This will prevent performing the

Figure 7: The Graph Walk algorithm

```
Graph Walk (M, M1) : T

if module M is not visited then
    mark M as visited
    S  := Start_Node(M)
    S1 := Start_Node(M1)
    Retest(M) := Compare(S, S1)

T := Retest(M)

return T
```

algorithm more than once in case there was more than one call to the same module in the module *Call Graph*.

The *Compare* algorithm recursively calls itself on the successor nodes of the current nodes to traverse all the graph nodes of the original and modified modules. It also collects test cases that should be called due to module calls. Term Test(N) denotes the test cases that reach node N. To prevent the algorithm from running endlessly we mark the visited nodes.

Function *Component*(N) in the *Is_Different* algorithm returns the database components used by node N. In the first step of the *Is_Different* algorithm, we check whether the nodes are lexically different. In the next steps, we check for differences resulting from the usage of database components.

Next, we discuss the results of applying the *Graph Walk* reduction algorithm on the previous example for each modification case.

Figure 8: The Compare algorithm

```
Compare (N1, N2) : T

mark node N1 as visited

if Is_Different(N1, N2) then
    return test(N1)

for each module M1 in CALL(N1)
    let M2 be the corresponding module in the modified application
    T := T U (test(N1) ∩ Graph Walk(M1, M2))

for each successor node s1 of N1 and corresponding node s2 in N2
    if s is not visited then
        T := T U Compare(s1, s2)

return T
```

Figure 9: The Is_Different algorithm

```
Is_Different(N1, N2) :

if N1 != N2 then
    return true

for each component C in Component(N1)
    if C is modified then
        if C is a module then
            if C does not belong Component(N2)
                return true
        else
            return true

for each component C in Component(N2)
    if C does not belong Component(N1)
        return true

return false
```

Case 1

Change the definition of trigger *Update_Balance*. Instead of "after insert", change it to "after delete". This modification makes the trigger refrain from firing when statement 3 in *Pay_Check* module is executed, and thus the statement 3 is affected and all test cases executing this statement are included in the retest list. These test cases are T2, T3, and T4.

Case 2

Add a not-null constraint to value_date column in table TRANSACTIONS. This column is not listed within the insert list of the insert statement in module *Pay_Check*. However, in our impact analysis we include this column as an implicit usage since it is set to null. By adding not-null constraint to this column, the insert statement is using implicitly a modified component. Thus, all test cases executing this statement should be executed. Similar to case 1, statement 3 in module *Pay_Check* is affected and the same test cases are selected. These test cases are T2, T3, and T4.

Case 3

Statement 2 in function *Stolen_Check* is to be modified. The statement, return(true), becomes return(false). This modification implies that all test cases executing statement 2 should be added to the retest list, which are test cases T1 and T5.

Call Graph Firewall

Leung and White (1990a) present a selective regression testing technique for interprocedural testing that deals with both code and specification changes. Their technique determines where to place a firewall around modified code modules. Then, it selects unit tests for modified modules that lie within the firewall and integration tests for groups of interfacing modules that lie within the firewall. Leung and White (1992) extend their technique to handle interactions involving global variables.

The firewall technique selects all units and integration tests of modules that lie within the firewall. Because not all of these tests necessarily execute modified code, the technique selects non-modification-traversing tests. It handles multiple modifications in a single pass of its algorithm.

Implementing the firewall concepts for database applications requires three elements:

(a) Database application Call Graph.
(b) Data flow dependencies between interfacing modules resulting from database tables usages.
(c) List of modified database modules.

Call Graph links a database module to all the modules that it calls. It should include links to table triggers modules in case the module contains statements that cause these triggers to execute. In order to find the data flow dependencies between interfacing database modules, we first have to find the table usages in each module and then use Call Graph to find define-use associations between table usages.

To find modified database modules, we perform impact analysis using the Component Firewall impact analysis technique. However, not all the modules selected by impact analysis are selected. We divide these modules into two sets. The first set of modules contains the modules that have been included because of modifications made to their code, or because they use modified non-module components. All modules in this set are considered by the Call Graph Firewall regression testing technique as modified. The rest of the modules are included in the second set and are not considered modified.

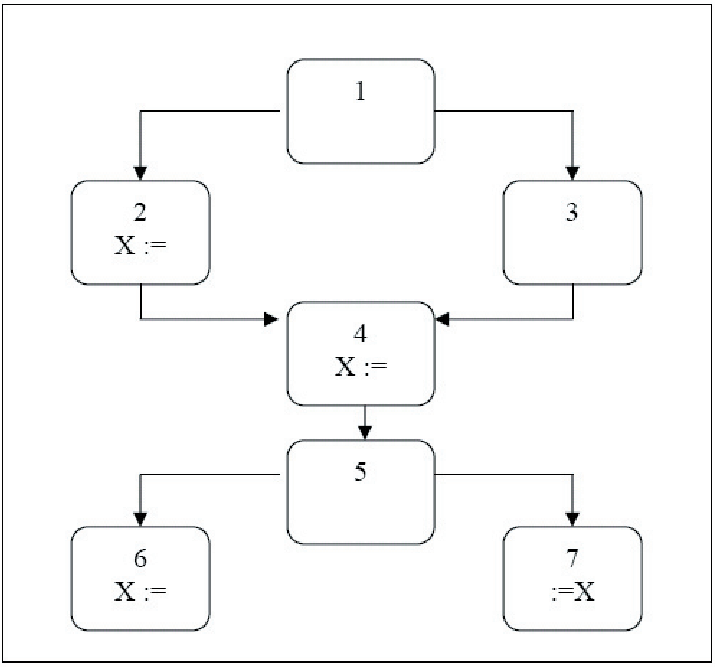
The new version of the database application is used only to determine the list of modified modules. If such a list is available from other sources then the new version of the database application is not needed.

Call Graph Firewall deals with two types of test cases: integration tests and unit tests. It determines which interfacing module couples need integration testing, so all the test cases passing through a selected couple should be selected. The table data flow information is used to determine:

(a) Modules in need of unit testing other than the directly modified modules.
(b) Interfacing module couples in need of integration testing to limit propagation of modification effects.

For example, consider the Call Graph of Figure 10. Suppose that a modification on column **X** has been done at module 7. Let set **E** be the set of nodes constituting the firewall, and let set **W** be the set of node couples that need integration testing as a result of change. Applying the algorithm leads to the following: the backward walk identifies module **4** where a definition of column **X** is found. Then, modules **2** and **3** with their arcs **(2, 4)** and **(3, 4)**, to the definition in module 4 are added to **W.** Module **1** with its arcs **(1, 2)** and **(1, 3)** is added to the firewall **E**. Next, a depth first search procedure is applied, leading to the addition of module **7** to **W** because a c-use of column **X** is found. Also, the predecessor of module **7**, module **5,** is added to W. Therefore, the firewall is composed of the components **1, 2** and

Figure 10: Call Graph example



3 and **W** is the component composed of **(2, 4)**, **(3, 4)** and **(5, 7),** for which every test case passing through these pairs must be retested.

EMPIRICAL RESULTS

Support System

We have implemented a database applications maintenance tool as a support system for the empirical work. The tool helps database application maintainers understand these applications, identify code changes, support software updates, and enhance and detect change effects. It mainly helps create a test environment and select regression test cases to be rerun when a change is made to the application using our two-phase regression testing methodology. The system is implemented for Oracle database applications programmed using PL/SQL language. Our maintenance tool is composed of five parts: module analysis, database analysis, test environment setup, Impact Analysis and regression test selection, and test case reduction. Test case reduction simply refers to including phase 2 algorithms.

Module Analysis

Module analysis involves building syntax trees for a module and then using these syntax trees to gather control flow information and database components usage information. The module information gathered is displayed to help in understanding the module's functionality. Performing all these tasks requires a mechanism of storing the results in one step and passing them to the succeeding steps. Our tool is built using the object-oriented paradigm. Thus, the mechanism used to store information is based on object-oriented structures.

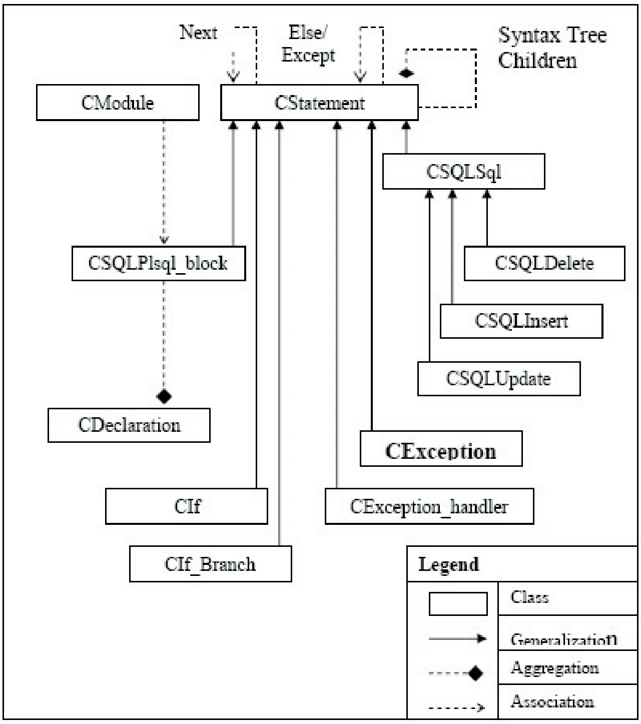
In Figure 11 we show part of the object model used to hold the information gathered while performing the module analysis.

Database Application Analysis

A database application is analyzed to determine component dependencies, call dependencies, and data flow dependencies that exist between its components. All gathered information is then displayed to help the maintainer visualize the features of the application.

The tool performs module analysis on all the application's modules. Database component usages are summarized for each module. Figure 12 shows the object model used to model database components and their relations and dependencies. After the analysis of all modules, we obtain a call graph of the database application in which we know the modules called by a certain module and all the modules calling it as well. Then, the tool displays all the gathered information in a hierarchical fashion.

Figure 11: Part of the object model for the syntax tree and control flow information



Having the call graph and the component usages, which include column usages, we can now perform inter-procedural data flow analysis. The result of this analysis is used later for the call graph firewall regression testing technique.

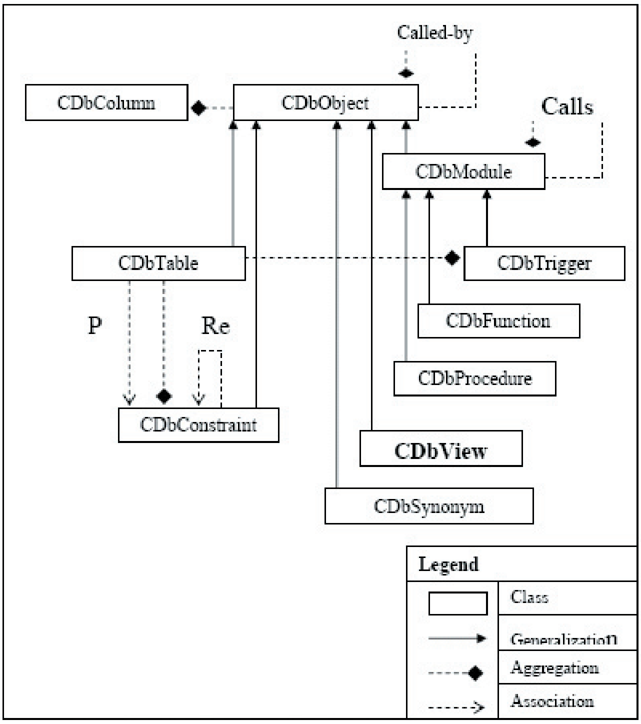
Test Environment Setup

The tool also supports testing efforts. It creates a new version of the database application that generates module and statement test traces when the test cases are executed. The tool tackles the problem of documenting database applications test cases by adding to the application input and output logging capabilities. The tool also uses its module and database application display utilities to display statement and module test coverage.

Change Impact Analysis

The tool is capable of performing change impact analysis by handling two database applications concurrently. It first connects to the original database application and then it connects to its modified version. It performs analysis on both versions of the application.

Figure 12: The object model of the database components



The results of this analysis are used in finding the modified database components and the effects of this modification on other database components.

Experimental Design and Procedure

To empirically investigate the use of our regression testing methodology, we use a prototype of a payroll database application with an initial suite number of test cases used to test its various modules and constructs. We propose random modifications to the application, thus creating ten (modified) versions of the application, M1 – M10. Then, we study each version using our maintenance tool and report the affected modules and the test cases that should be rerun for regression testing. The test suite used to test this application contains fifty test cases determined using a specification-based test adequacy criterion.

For evaluating and comparing the regression test selection techniques, we use two metrics: (i) the (percentage) number of tests (ST) selected by a technique from the initial test suite for rerunning, and (ii) the (percentage) number of modification-revealing tests missed by a technique (MMRT). Obviously, the underlying assumption is that a good regression testing technique selects a small number of tests (ST) to reduce the time of regression testing, and yet does not miss selecting the tests that reveal the modifications made to the database application.

The experiment is made of two parts. In the first part, we analyze the database application and prepare the test trace information. This part involves the following steps:

(a) Use the tool to construct syntax trees, control flow graphs, component dependency information, and data flow information.
(b) Use the tool to create a new version of the application and generate a test trace.
(c) Run all tests on the new version and collect trace information.

In the second part, a new copy of the application for each proposed modification is created. For each modified version of the application:

(a) Perform database application analysis.
(b) Perform Impact Analysis.
(c) Run Graph Walk regression testing.
(d) Run Call Graph Firewall regression testing.

Results

In Table 3, we present the results of applying our regression testing methodology on the ten program versions, M1 – M10. For each version, we show: (a) the number of test cases in the initial test suite (selected by the Select-All approach), (i.e., 50) and the number of tests that reveal the modifications (MRT) made to the application, (b) the ST and MMRT values due to the use of a Select-Random technique, (c) the ST and MMRT values due to the use of our proposed phase 1 – Impact Analysis technique, and (d) the ST and MMRT values due to the use of the phase 2 techniques for further test reduction, Graph Walk and Call Graph Firewall. All ST values are normalized with respect to 50, whereas MMRT values are normalized with respect to MRT. The ST value of Select-Random is set to be 28% after observing that all ST values (except for M1) of our proposed techniques are less

Table 3: Summary of results

Versions	Select-All		Select-Random		Phase 1 Impact Analysis		Phase 2 Test Reduction			
	Number of Tests	MRT	ST	MMRT	ST	MMRT	Graph Walk		Call Graph Firewall	
							ST	MMRT	ST	MMRT
M1	50	10	28%	60%	36%	0%	36%	0%	22%	20%
M2	50	5	28%	60%	28%	0%	28%	0%	16%	20%
M3	50	4	28%	75%	8%	0%	8%	0%	8%	0%
M4	50	4	28%	100%	8%	0%	8%	0%	8%	0%
M5	50	8	28%	62%	28%	0%	28%	0%	14%	25%
M6	50	12	28%	58%	28%	0%	28%	0%	14%	8%
M7	50	2	28%	100%	8%	0%	8%	0%	8%	50%
M8	50	10	28%	70%	28%	0%	24%	0%	24%	10%
M9	50	5	28%	60%	28%	0%	10%	0%	10%	40%
M10	50	2	28%	100%	28%	0%	4%	0%	10%	50%

than or equal to 28%. This choice is made to facilitate the comparison of the MMRT values between the Select-Random and the three proposed techniques.

Discussion of the Results

Table 3 clearly shows that the Impact Analysis, Graph Walk, and Call Graph Firewall techniques provide significant reduction in the number of selected tests in comparison with the Select-All approach. For example, Impact Analysis provides an average reduction of 77% with a minimum of 64% and a maximum of 92%. Graph Walk and Call Graph Firewall provide average reductions of 82% and 87%, respectively. Therefore, the three proposed techniques are certainly useful for saving regression testing time.

In addition to their test reduction capabilities, both the Impact Analysis and Graph Walk are safe. That is, they do not miss modification-revealing tests. However, Graph Walk produces further reduction in ST where the modification affects modules that involve selection and branching. When faced with simple code modules, Graph Walk and Impact Analysis have similar reduction capabilities. But, since the Graph Walk technique works at the statement level, it offers more reduction in ST as the code becomes larger and involves deep branching hierarchy of modules. Versions M8-M10 are examples in which the modifications made affect branching parts of the code.

The Call Graph Firewall technique is not safe. It produces the best reduction in ST, but at the expense of the MMRT value. Call Graph Firewall uses data flow information to further reduce the tests selected by Impact Analysis. This may be advantageous for fast regression testing in the case where Impact Analysis provides high values of ST. But, it is not recommended for relatively small ST values as it might miss 50% of the modification revealing tests.

Comparing Impact Analysis, Graph Walk, and Call Graph Firewall with Select-Random shows that the three proposed techniques are definitely better. They offer less ST values and are certainly more reliable, as Select-Random misses 58-100% of the modification revealing tests. These high MMRT values occur despite allowing Select-Random to select 28% of the initial tests. This ST value is comparable to the three proposed techniques for some versions (e.g., M2, M5, and M8) or has advantage over them for other versions (e.g., M3 and M7). In particular, Select-Random might miss all modification-revealing tests for small MRT values.

RELATED WORK

Numerous regression testing algorithms and approaches have been proposed for procedural and object-oriented programs. Rothermel, Harrold, and Dedhia (2000) provide a regression testing method for C++ software based on control flow analysis of C++ source code. The method handles some object-oriented and C++ features such as polymorphism, dynamic binding, and passing objects as parameters. Rothermel, Yntect, Chu, and Harrold (2001) use test case prioritization in regression testing. They provide a survey of test case prioritization techniques and perform empirical studies with some of these techniques to evaluate how effective they are in improving fault detection. In case safe regression testing techniques proved not feasible, prioritization is chosen as a cost effective substitute. Bible, Rothermel, and Rosenblum (2001) provide a comparitive empirical study of two safe regression test selection techniques implemented in two regression testing tools: the TestCube (Chen, Rosenblum& Vo, 1994) and the DejaVu (Rothermal & Harrold, 1997). The precision and relative cost effectiveness of these techniques are evaluated and compared to the cost of retesting using other techniques. Harrold, Jones, Li, and Liang (2001) present a safe regression testing selection technique for Java applications. The technique handles Java language features such as polymorphism, dynamic binding, and exception handling. The authors also described a tool for implementing their technique. The tool provides empirical results for checking the effectiveness of their technique in test case reduction. Beydeda and Gruhn (2001) present a black box based regression testing technique. They represent a software system by a domain model and test case selection is based on changes to the model. Their approach is divided into two phases. In phase one, test case selection is performed on old test cases. In phase two new test cases are generated to test newly added parts.

Other algorithms invloving regression testing include: incremental slicing algorithm (Agrawal, Horgan & Krauser, 1993), slicing algorithms based on data flow testing and incremental data flow analysis described by Gupta, Harrold, and Soffa (1996) and Harrold and Soffa(1988), firewall-based approaches presented by Hsia et al., (1997), Kung et al. (1995), Leung and White (1990a, 1990b), and Leung and White (1992), stochastic search algorithms (Mansour & El-Fakih, 1999), safe algorithm based on module dependence graph described by Rothermel and Harrold (1997, 1998), semantic differencing approach (Binkley, 1997), and textual differencing approach (Vokolos & Frankl, 1998). However, to the best of our knowledge, database programs have not been specifically dealt with in regression testing research. SQL-based database programs support a number of features that do not exactly apply in the cases of procedural and object-oriented programs. Examples of these features are: SQL statements, table constraints, exception programming, and table triggers. These features introduce new difficulties that hinder regression test selection.

CONCLUSIONS AND FURTHER WORK

We presented a two-phase regression testing methodology for SQL-based systems. In phase 1, we suggested techniques for modification detection and modification Impact Analysis, in which we determined affected modules and test cases traversing them. In phase 2, we presented two alternative algorithms for reducing the test cases selected in phase 1. The Graph Walk algorithm is statement-based and extends to the inter-procedural level. The Call Graph Firewall algorithm is based on firewalls for database applications. In addition, we developed a support system and used it for the experimental work.

In phase 1, we showed that exceptions could be modeled using existing control flow constructs with some alteration. We presented control flow modeling techniques that take into consideration nested compound statements with exception handling. Also, we proposed a data flow analysis method that uses database interactions and is based on identifying the usage of table columns. Furthermore, we provided a detailed analysis of the component dependencies that exist between various database components. We found that control flow analysis, Call Graph modeling, data flow analysis, dependency analysis and Impact Analysis are useful for regression test selection.

From the empirical results, we conclude that: (i) the proposed techniques are better than the Select-All approach in saving regression testing time and are more reliable than the Select-Random approach in selecting modification revealing tests; (ii) Impact Analysis is very effective in localizing the effects of modifications and is useful in a preliminary selection of regression test cases; (iii) the Graph Walk technique is particularly successful in reducing the number of selected tests for code modifications while ensuring the selection of modification revealing tests; (iv) the Call Graph Firewall technique for test reduction is most useful when applied to modular applications that include a hierarchy of module calls. However, it may miss some modification revealing tests.

Based on the work presented in this chapter, further research tasks can be pursued. We tackled the database application part that resides fully on the server. Ideas presented can be extended to include client programs interfacing with the database components using SQL and PSM calls. These client programs could be visual interfaces or part of three-tier applications. Moreover, complexities created by cursors like data flow and component dependencies need further research. In addition, dynamic analysis of table row usage by SQL statements can be used to perform row level dynamic data flow analysis.

REFERENCES

Agrawal, H., Horgan, J.R., & Krauser, E.W. (1993). Incremental regression testing. *Proceedings of International Conference on Software Maintenance*, 348-357.

Arnold, R.S., & Bohner, S. A. (1996). *Software change impact analysis*. IEEE Press.

Beydeda, S., & Gruhn, F. (2001). An integrated testing technique for component-based software. *Proceedings of the ACS/IEEE International Conference on Computer Systems and Applications*.

Bible, J., Rothermel, G., & Rosenblum, D.S. (2001). A comparative study of coarse and fine-grained safe regression test selection techniques. *ACM Transactions on Software Engineering and Methodology*, *10*(2), 149-183.

Binkley, D. (1997). Semantics guided regression test cost reduction. *IEEE Transactions on Software Engineering*, *23*(8), 498-516.

Chen, Y., Rosenblum, D., & Vo, K.P. (1994). TestCube: a System for selective regression testing. *Proceedings of the 16th International Conference on Software Engineering*, 211-220.

Gupta, R., Harrold, M.J., & Soffa, M.L. (1996). Program slicing-based regression testing techniques. *Journal of Software Testing, Verification and Reliability*, *6*(2), 83-111.

Harrold, M.J., & Soffa, M.L. (1988). An incremental approach to unit testing during maintenance. *Proceedings of International Conference on Software Maintenance*, 362-367.

Harrold, M.J., Jones, J.A., Li, T., & Liang, D. (2001). Regression test selection for Java software. *Proceedings of the ACM Conference on Object Oriented Programming, Systems, Languages, and Applications*.

Hartmann, J., & Robson, D.J. (1989). Revalidation during the software maintenance phase. *Proceedings of International Conference on Software Maintenance*, 70-79.

Hsia, P., Li, X., Kung, D.C., Hsu, C-T., Li, L., Toyoshima, Y., & Chen, C. (1997). A technique for the selective revalidation of OO software. *Journal of Software Maintenance*, *9*, 217-233.

ISO/IEC 9075. (1992). *Information Technology – database languages – SQL*.

ISO/IEC 9075-4. (1995). *Information Technology – database language – SQL Part 4: Persistent Stored Modules (SQL/PSM)*.

Kung, D.C., Gao, J., Hsia, P., Wen, F., Toyoshima, Y., & Chen, C. (1995). Class firewall, test order, and regression testing of object-oriented programs. *Journal of Object-Oriented Programming*, *8*(2), 51-56.

Leung, H.K.N., & White, L. (1990a). A study of integration testing and software regression at the integration level. *Proceedings of International Conference on Software Maintenance*, 290-300.

Leung, H.K.N., & White, L. (1990b). Insights into testing and regression testing global variables. *Journal of Software Maintenance*, *2*, 209-221.

Leung, H.K.N., & White, L. (1992). A firewall concept for both control-flow and data-flow in regression integration testing. *Proceedings of International Conference on Software Maintenance,* 262-271.

Mansour, N., & El-Fakih, K. (1999). Simulated annealing and genetic algorithms for optimal regression testing. *Journal of Software Maintenance*, *11*, 19-34.

Rapps, S., & Weyuker, E.J. (1985). Selecting software test data using data flow information. *IEEE Transactions on Software Engineering, 24*(6), June, 401-419.

Rothermel, G., & Harrold, M.J. (1997). A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, *6*(2), 173-210.

Rothermel, G., & Harrold, M.J. (1998). Empirical studies of a safe regression test selection technique. *IEEE Transactions on Software Engineering, 24*(6), June, 401-419.

Rothermel, G., Harrold, M.J., & Dedhia, J. (2000). Regression test selection for C++ Software. *Journal of Software Testing, Verification, and Reliability*, *10*(2), June.

Rothermel, G., Untech, R.H., Chu, C., & Harrold, M.J. (2001). Prioritizing test cases for regression testing. *IEEE Transactions on Software Engineering*, *27*(10), 929-948.

Vokolos, F.I., & Frankl, P.G. (1998). Empirical evaluation of the textual differencing regression testing technique. *Proceedings of the International Conference on Software Maintenance*, 44-53.

Wong, W.E., Horgan, J.R., London, S., & Agrawal, H. (1997). A study of effective regression testing in practice. *Proceedings of the 8th IEEE International Symposium on Software Reliability Engineering (ISSRE'97)*.

Chapter IX

An Attempt to Establish a Correspondence between Development Methods and Problem Domains

Oscar Dieste, Universidad Complutense de Madrid, Spain

Marcela Genero, Universidad de Castilla-La Mancha, Spain

Natalia Juristo, Universidad Politecnica de Madrid, Spain

Ana M. Moreno, Universidad Politecnica de Madrid, Spain

ABSTRACT

Most development methods need to be adapted before they can be used in a specific development project. This is because each method can be applied to a series of paradigmatic problems, but, as a problem moves further away from the ideal, the effectiveness of each method gradually decreases. Although development method adaptation has been a recurrent theme in the literature, no work has been published that proposes any sort of criterion or metric that can be used to assess the fitness of any one method to a particular problem. Therefore, in this chapter, we propose a new approach that can be used to calculate the fitness of methods to particular problems.

INTRODUCTION

There are now a wide variety of software development methods and techniques (Bubenko, 1986). Although it is chancy to generalize, most of these methods and techniques are considered to be general-purpose and, therefore, are specifically designed to operate in a wide range of domains and to deal with an ample variety of different problems (Glass & Vessey, 1995).

For some years now, however, results have been published that contradict the supposed universality of development methods and techniques. Accordingly, the survey by Hardy, Thompson and Edwards (1995) concerning the use of structured methods and techniques in the UK, indicates that 88% of methods and techniques undergo some sort of adaptation before being used in each particular development project. Russo, Wynekoop and Walz (1995) offer similar figures (88.6%) for the likewise structured methods and techniques in use in the USA. Things do not appear to be much different as regards object orientation, as notation adaptation is routine practice. Accordingly, UML, the most popular object-oriented modeling language today, has three built-in extension mechanisms—stereotypes, tag definitions and constraints (Fowler & Scott, 1999)—which can be used to adapt the representation capabilities of the language to better represent particular domains and problems (Fowler & Kobryn, 2002).

From percentages like the above, we can deduce that almost all the methods and techniques should be adapted before being used in practice, which means that the generality of these methods and techniques is merely a guise. The need for method and technique adaptation reflects the fact that they are in some measure specific for particular problems. Specificity should be taken to mean that the methods and techniques are primarily oriented to solving a paradigmatic problem type, namely, the problems for which they were designed. As the problems addressed in practice move away from the paradigmatic problem, the methods and techniques become less effective and need to be adapted before being used.

Method and technique adaptation is a field in which various results have been published over the last ten years, as indicated. However, no criterion has yet been proposed that can be used to decide to how well suited a method or technique is for a given problem P (Glass & Vessey, 1998). Subjective assessment by developers usually fills in for the missing formal criteria. However, this procedure is neither systematic nor repeatable, and therefore can be qualified as not very engineering-like.

Moreover, such a criterion could be used to establish a methods and techniques hierarchy with respect to their fitness for a problem P, making it possible to identify and select the best-suited method or technique. Additionally, a fitness measure would be equally important for adapting the method or technique, as it could be used to formally assess the situation before and after adaptation.

Therefore, in this chapter, we propose a method that can be used to assess how well suited a method or technique is to a given problem P. For this purpose, we will proceed as follows. The following section will review the different alternatives that have been proposed in this respect in the scientific literature. This is followed with a description of the proposed assessment method. Next, the chapter will discuss possible extensions and future improvements. The chapter will end with some conclusions.

BACKGROUND

There are two factors that lead to method and technique specificity: (1) the process; that is, the prescribed procedural steps for their use and (2) the conceptual models used by the methods and techniques (Glass & Vessey, 1998). The process is usually adapted to particular problems on the basis of experience (Hardy et al., 1995), although work on the systemization of this type of adaptation has been published over the last ten years, leading to the creation of the field known as Method Engineering (Brinkkemper & Joosten, 1996; Hofstede & Verhoef, 1997).

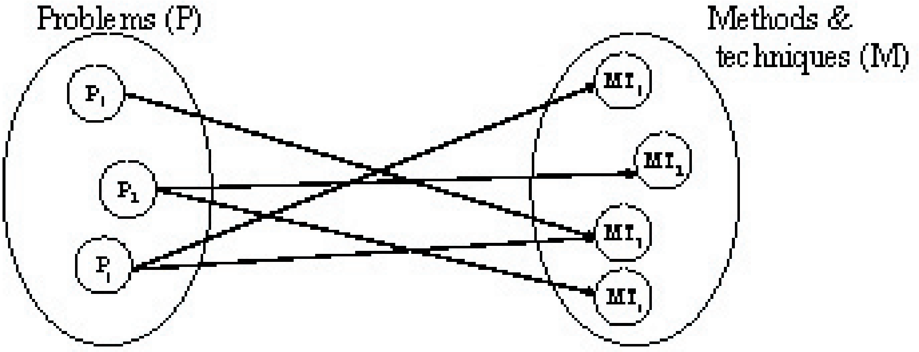
Conceptual models, however, can be said to cause greater method and technique specificity. This is because the conceptual models define the problem domain aspect on which the methods and techniques focus (Davis, 1993). Accordingly, for example, the basic conceptual model of the structured methods and techniques is the data flow diagram, which can record transformations that occur in the problem domain through its basic constructors: processes, data flows, etc. On the other hand, the dominant model in the object-oriented methods and techniques is the class diagram, which has constructors that can account for objects, classes, attributes, etc., related to the problem to be solved.

When a particular problem does not fit the conceptual model used by a method and technique, this model should be adapted. Two main, partially overlapping, lines of research have been pursued in this respect. The first is characterized by the conception of increasingly richer conceptual models, which can express a greater diversity of problem domain aspects and are, therefore, less specific. Models like *i** (Yu, 1995), KAOS (Lamsweerde, Dardenne, Delcourt & Dubisy, 19991) or EM (Kirikova & Bubenko, 1994) are within this line. A second line of research is characterized by the explicit definition of metamodels, which, in some cases, can extend the representation capabilities of the conceptual models. One of the most significant examples of this line of work is the ConceptBase tool (Nissen, Jeusfeld, Jarke, Zemanek & Huber, 1996), based on the TELOS knowledge representation language (Mylopoulos, Borgida, Jarke & Koubarakis, 1990).

It is clear then that there is profound interest in adapting the different methods and techniques to each particular problem addressed. As yet, however, no criterion or formal metric has been defined that can be used to identify when a method or technique is suitable for a particular problem (Glass & Vessey, 1998). This is largely due to the difficulty of comparing problems with methods and techniques. This comparison is equivalent to establishing a correspondence between two sets: a set P of problems and a set M of methods and techniques, as shown in Figure 1.

There are two main problems that need to be solved to be able to establish the above-mentioned correspondence:

Figure 1: Procedure for determining method and technique fitness



- Methods and techniques have not been characterized and classified beyond the typical division into structured, real-time and object-oriented methods and techniques, although the particularities (prescribed process, models used, stakeholders involved, etc.) of each individual method and technique are usually well known. Some papers that attempt to undertake a classification are Webster (1988), Zave (1990) and Firth, Pethia, Roberts, Mosley and Dolce (1987). However, these papers do not agree on either the classification criteria or the methods and techniques considered, which means that they are only partially useful. There being no satisfactory classification, it is enormously difficult to identify the fitness of methods and techniques to problems, as each method and technique has to be worked on individually, which is not very systematic, difficult to extrapolate and very costly in terms of time and effort. There is no categorization and classification of problems. This means that the problem domain aspects that are relevant for examining fitness are not known (Glass & Vessey, 1995).

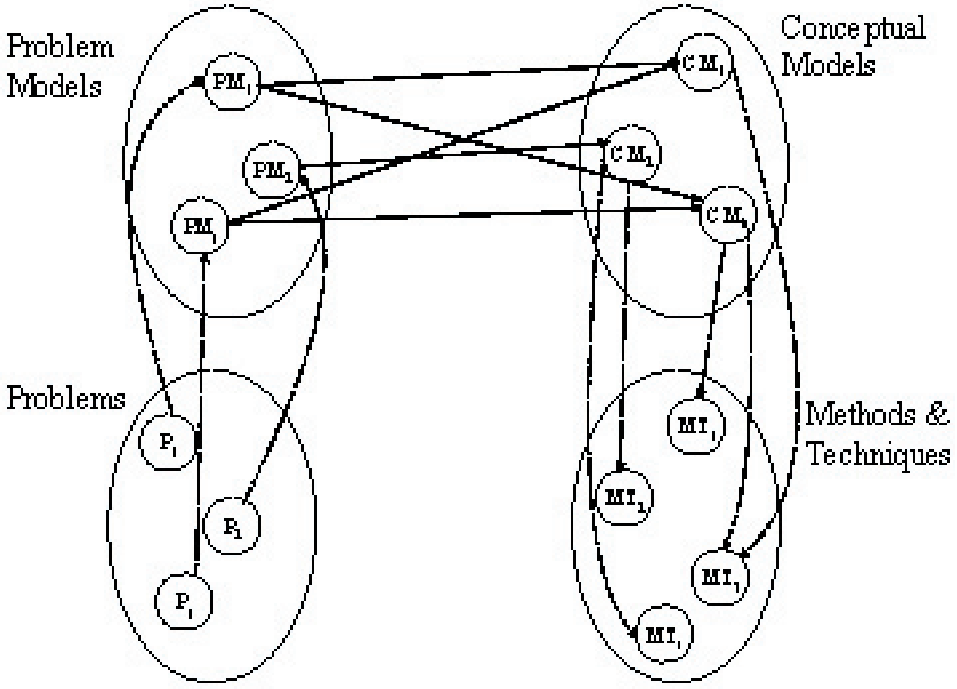
Because of the above-mentioned difficulties, that is, the absence of satisfactory catalogues of methods and techniques and of problems, it is far from easy to identify criteria of correspondence that can be used to relate methods and techniques to problem domains. With some simplifications, however, it is possible to come up with a strategy that can be used to establish the correspondence between methods and techniques and problems. The strategy proposed here has been formalized as a method for calculating method and technique fitness, as discussed in the following section.

PROPOSAL FOR DETERMINING METHOD AND TECHNIQUE FITNESS

Due to the difficulties discussed in the preceding section, some simplifications need to be made with regard to how to establish a correspondence between a set P of problems and a set M of methods and techniques. These simplifications, as shown in Figure 2, are as follows:

1. Rather than working on the set M, opt to use the set of conceptual models (CM) used by the methods and techniques in set M. This means that much fewer elements need to be considered, as many of the methods and techniques use the same, or very similar, conceptual models, which means that the number of elements is smaller in set CM than in set M (Dieste, Moreno & López, 1999).
2. Identify the fundamental elements of the problems belonging to P. Fundamental elements should be taken to mean the aspects that characterize each problem, that is, partially or totally differentiate one problem from another. To identify the fundamental elements of a problem, the problem needs to be examined and understood or, in other words, modeled. Ordinary modeling is no good, however. Indeed, we would be going around in circles, as explained later on, if we used the classical conceptual models, like the data flow diagram, the class diagram or the state transition diagram to model a problem P. Therefore, we need a new model type, which has been termed generic conceptual model and which can model the problem without overlooking any of its characteristics or forcing it to fit any particular conceptual model. Using the generic conceptual model, the problems in P can be mapped to a set of models of P (PM).

Figure 2: Simplified procedure for determining method and technique fitness



The strategy that we will follow to examine method and technique fitness is to establish a criterion of correspondence between PM and CM, based on the comparison of the generic conceptual model of each problem P and the conceptual models used by the methods and techniques. This correspondence to be established between PM and CM is precisely what stipulates that the set PM should not be constructed on the basis of classical conceptual models, like the class diagram or the data flow diagram. If this were the case, the problem P modeled in PM would be biased towards the conceptual model used, which means that the correspondence between PM and CM would be predetermined by the modeling process; that is, PM and CM would be the same set. On the other hand, the generic conceptual model can be used to compare, as will be shown in section 3.1, the characteristics of the problems modeled in PM with the models in CM that can express these characteristics, thus establishing, therefore, the correspondence between PM and CM and, ultimately, between P and M.

Accordingly, it is possible to come up with a method for determining the fitness of conceptual models and, ultimately, methods and techniques for a given problem. This method, called problem-oriented analysis method (POAM), uses a generic conceptual model to map the set P to the set PM, as well as to compare the sets PM and CM. The use of the generic conceptual model has been formalized in a well-defined procedure, which can be used to systematically determine conceptual model fitness. The association of methods and techniques and conceptual models, which can be used to pass from the set M to the set CM.

Generic Conceptual Model

The generic conceptual model (GCM) is a set of representation formalisms that can record relevant information about the problem from a neutral viewpoint, that is, not in any way linked to the future implementation. Specifically, the GCM is composed of four different representation formalisms:

- *Concept map:* This is the principal representation formalism of the GCM. It is a graphic formalism, inspired by conceptual maps, derived from the work of Ausubel on Learning Theory and Psychology and later formalized by Novak and Gowin (1984). The concept map can set out and describe concepts and associations between concepts present in the problem domain. From the viewpoint of form, concept maps are, as shown in Figure 3, quite similar to semantic nets. However, there are profound differences of substance between the two model types. Specifically, concept maps and semantic nets differ with regard to the way that the two formalisms are designed to represent the knowledge. Semantic nets are designed to describe knowledge non-ambiguously; that is, it should be possible to ascribe a well-defined meaning to any node and link, although this is not possible in some cases (Woods, 1975). On the other hand, concept maps are intrinsically ambiguous; that is, it is neither possible nor desirable *a priori* to ascribe any particular conceptual or computational meaning to concepts and associations. Additionally, concept maps can be used to build combinations of concepts and associations (called *propositions*) of varying complexity, with an expressiveness approximating natural language. Finally, concept maps can be structured hierarchically, similarly, albeit founded on different theoretical principles, to data flow diagram hierarchies (Dieste, 2003).
- *Identificative dictionary:* This is a tabular representation formalism that can record, during the early phases of analysis, concepts and associations, and is designed to make the concept map easier to use. Table 1 shows an example.

The apparent duplication in recorded information in Table 1 (two different entries for room, ward and complaint) is due to a GCM technicality: Two different elements in the concept map are considered different even if they bear the same name. This rule prevents

Figure 3: Example of a concept map

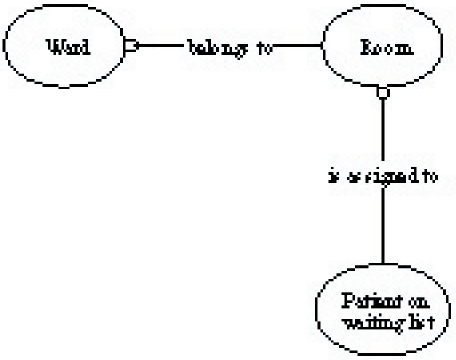


Table 1: Identificative dictionary

Element	Description
Hospital 123	Admits patients
Patients	From waiting list From emergency department
Waiting list patient	Is assigned a room Suffers complaint Is admitted from waiting list
Emergency department patient	Is assigned a room Suffers complaint Is admitted from emergency department
Doctor	Is reference physician of waiting list patient Is specialized in complaint
Emergency doctor	Is reference physician of emergency department patient Is specialized in complaint Treats patient in emergency department
Room	Belongs to ward
Ward	Is assigned to a complaint
Complaint	

information loss, primarily in the early stages of analysis, as information that appears to be similar may turn out to be very different later.

- *Descriptive dictionary:* This is a likewise tabular representation formalism, which is used, unlike the identificative dictionary, during the later phases of analysis. Its function is to record refined information about the problem domain, which will later enable the identification of which CM and, therefore, which method and technique are best suited for solving the problem under analysis. Table 2 shows an example.
- *Narrative text:* This is a textual representation formalism that can be used to transcribe the information recorded in the concept map and the dictionaries. The narrative description can be automatically derived from the concept map and dictionaries, which has some clear benefits for model validation. The text is very understandable for end users, and, as there is a direct relationship between the narrative description and the other representation formalisms, the comments and corrections made by the users can be fed back into the concept map and the dictionaries. Table 3 shows an example.

Of these formalisms, the principal one is the concept map, as it supports problem examination and understanding. Additionally, transformation rules have been defined between the different GCM representation formalisms (Dieste, 2003), which means that the concept map, dictionaries and narrative text can represent the same information in different ways (graphs, tables and text). Tables 1, 2 and 3 actually do represent the same information as the concept map in Figure 4.

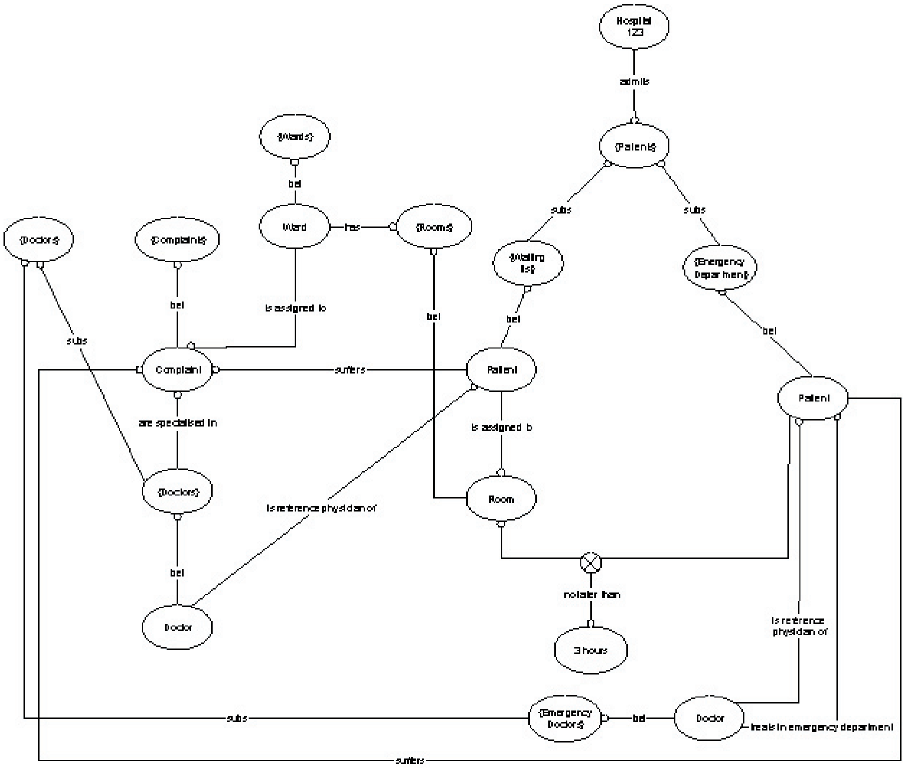
Table 2: Descriptive dictionary

Statements			
Sets:	Rooms		
	Wards		
	Complaints		
	Patients		
	Doctors 1		
	Doctors 2		
	Emergencydoctors		
Subsets:	Waiting list	subs	Patient
	EmergencyDept	subs	Patients
	Doctors 2	subs	Doctors 1
	Emergencydoctors	subs	Doctors_1
Individuals:	Complaint	bel	Complaints
	Room	bel	Rooms
	Patient 1	bel	Waiting list
	Patient 2	bel	EmergencyDept
	Ward	bel	Wards
	Doctor 1	bel	Doctors 2
	Doctor 2	bel	Emergencydoctors
Definitions			
Index	**Definition**		
Propositions			
Index	**Association**	**Concept-1**	**Concept-2**
1	Hospital 123	Admits	Patients
2	Doctors 2	is specialized in	Complaint
3	Pacient_1	Suffers	Complaint
4	Pacient 1	is assigned	Room
5	Ward	Has	Rooms
6	Ward	is assigned to	Complaint
7	Doctor 1	is reference physician of	Pacient 1
8	Doctor_2	treats in emergency dept.	Pacient_2
9	Doctor 2	is reference physician of	Pacient 2
10	Patient_2	is assigned	Room
11	Patient 2	Suffers	Complaint
12	p10	no later than	3 hours

Table 3: Narrative text

Hospital 123 admits patients from the waiting list or from the emergency department.
Waiting list patients are assigned a room and suffer a complaint and are admitted from the waiting list.
Emergency department patients are assigned a room and suffer a complaint and are admitted from the emergency department.
Doctor is the reference physician of the waiting list patient and is specialized in the complaint.
Emergency doctor is the reference physician of the emergency department patient and is specialized in the complaint and treats the emergency patient in the emergency department.
Room belongs to ward.
Ward is assigned to complaint.

Figure 4: Concept map

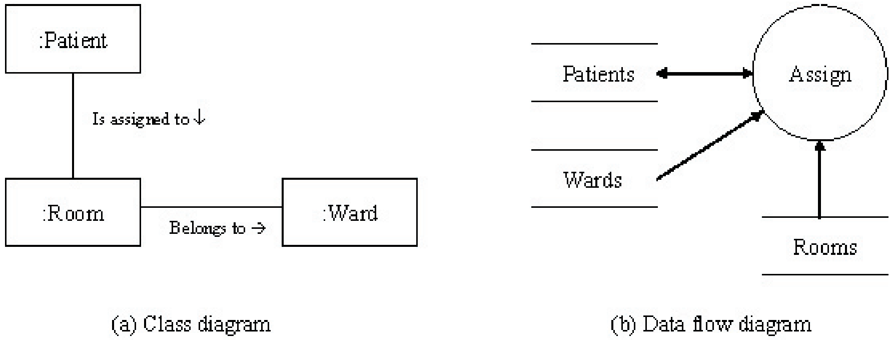


Note that each representation is similar to, while at the same time, slightly different from, the others. This is due to the fact that each GCM representation mechanism focuses on different aspects of the information acquired. The concept map highlights, primarily, the associations between the different elements, whereas the identificative dictionary emphasizes the meaning of each element. Finally, the narrative text locates elements and associations at the same level, easing communication with the less technically competent participants in the analysis.

Because of its intrinsic ambiguousness, the GCM, and the concept map in particular, can record the problem domain information without any sort of conceptual or computational constraint such as those imposed by the traditional conceptual models (Dieste, Genero, Juristo, Maté & Moreno, 2003), thus enabling the set P of problems to be mapped to the set PM of problem models. As mentioned, all conceptual models categorize the domain of discourse using constructors with a well-defined, that is, non-ambiguous, meaning. Therefore, the problem domain is filtered from the very start of analysis according to the viewpoint permitted by the conceptual model.

Accordingly, for example, the information recorded in the concept map shown in Figure 3 can be paraphrased as, "Patient is assigned to Room. Room belongs to Ward." If this same information were recorded in a class diagram or a data flow diagram, the result would be as shown in Figure 5(a) and (b), respectively: only the domain aspects that each conceptual model is capable of recording are considered during analysis. Specifically, in Figure 5(a), information related to the manual or automatic process of assigning a patient

Figure 5: Information represented by means of a class diagram and a data flow diagram for the sentence, "Patient is assigned to Room. Room belongs to Ward."



(a) Class diagram

(b) Data flow diagram

to a room (the phrase "is assigned to") has been lost, and this has been replaced by an association class, rewriting the problem in static code. In Figure 5(b), on the other hand, the relationship between patients and rooms is lost, and the information gathered is transcribed functionally, which, ultimately, indicates merely that the patients have some kind of attribute that is updated on the basis of wards and rooms.

Moreover, the intrinsic ambiguousness of the concept map enables the categorization to be carried at the end of the analysis, when all the relevant information has been acquired and correctly conceptualized. Accordingly, in the concept map shown in Figure 3, there is no constraint that demands that "patient" be modeled in advance as a class or external entity or "is assigned to" as a relationship or a process. This decision is postponed until the problem domain is well enough understood and the best-suited development paradigm has been selected. This view of analysis is vaguely similar to the one proposed by Ceri (1983) and Mayr and Kop (1998), who also use generic representation formalisms to record the problem domain information before going on to create the conceptual models.

By modeling the problem according to the concept map representation schemes, we have mapped set P to set PM. It remains, therefore, to define the correspondence between the sets PM and CM.

Determining Conceptual Model Fitness

As mentioned above, the correspondence between PM and CM is established by comparing the generic conceptual model of each problem P (or the respective concept map) with the conceptual models used by the methods and techniques. Being based on distinct theoretical foundations, however, the concept map and the classical conceptual models, such as the class diagram or the data flow diagram, cannot be compared directly. On the one hand, the concept map records ambiguous information, which is, therefore, susceptible to different interpretations. On the other, the conceptual models, albeit to different extents, record the information on the problem using a strict semantics and, therefore, a single meaning.

Owing to this impedance mismatch, the theoretical foundations of the concept map and the conceptual models need to be approximated, that is, assimilated. This is achieved by disambiguating the concept map. The ambiguity of the concept map is removed by ascribing a given interpretation to each concept and association in the concept map; that is, each concept and association is ascribed to the aspect (classes, relationships, processes, states, etc.) of the real world to which it refers. For this purpose, an interpretation procedure has to be applied to the GCM. The interpretation procedure can be used to assign an interpretation, that is, a well-defined meaning or semantics, to all the elements recorded in the GCM. The interpretation procedure is based on a requirements representation formalism proposed by Davis, Jordan and Nakajima (1997). This formalism, termed "Canonical Model", provides a set of building blocks (called "elements" and "links") that can be used to represent the information contained in a wide range of conceptual models. This means that it can be used as a lingua franca, which averts having to deal with each conceptual model separately.

From the Canonical Model, which was profoundly restructured to meet the objectives of our research, we have been able to define a set of tables of interpretation that operate on the information gathered in the concept map (Dieste, 2003). An example of these tables is shown in Table 4, which states all the possible combinations among elements and links. There are other additional tables that are used to consider propositions (Dieste, 2003), but are not included here for reasons of space.

The interpretation tables are used according to a set of interpretation rules, which are completely formalized in an algorithm and are highly independent of the analyst who is doing the interpreting. However, it is not always possible to achieve full independence, and the analyst should decide, depending on his/her knowledge of the GCM, which particular interpretation is the best suited. This happens when two or more elements of the Canonical Model can be assigned to any given GCM element, where the ambiguity cannot be eliminated algorithmically. After interpretation, the GCM is called the Requirements Canonical Model (RCM), as the GCM can now be read unambiguously, as a description of what should be future software system operation. An example is given in Table 5.

Having removed the ambiguity of the concept map, this has a well-defined meaning; that is, each concept and association can be read as a constructor (classes, relationships, processes, states, etc.) of one or more conceptual models. As each concept and association can be understood as constructor of one or more conceptual models, the concept map and the conceptual models can be directly compared, which means that the correspondence between the PM and CM sets is established. This many-to-many correspondence, viewed from the CM set side, indicates which problem domain aspects reflected in the concept map each conceptual model is capable of representing. The number of these aspects can be considered as a measure of the suitability of the conceptual models to problems insofar as it refers to the expressiveness, for a given problem P, of a set CM of conceptual models.

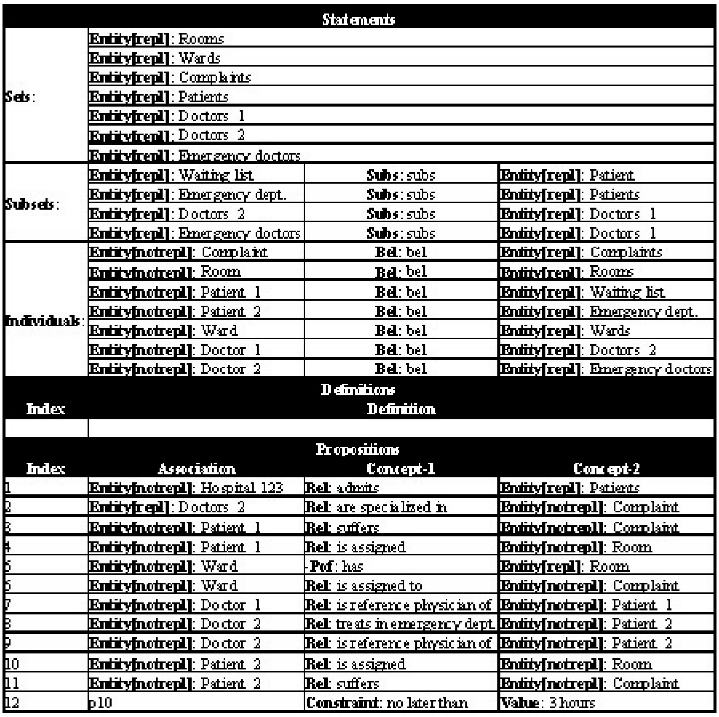
This number, which has been called fitness, is defined as the ratio between the propositions that a given conceptual model can represent and the total number of RCM propositions. A series of identification tables are used to identify how many RCM propositions a given conceptual model can represent. The identification tables are complementary to Table 4, as they identify which conceptual models can express each element-link-element combination in Table 4. By way of an example, Table 6 shows the identification table for the class diagram, although there are additional tables, approximately three for each conceptual model (Dieste, 2003).

The identification tables can relate each CRM proposition to the conceptual models that can express this proposition. Once all the propositions have been considered, fitness is calculated as the weighted sum of the propositions each conceptual model can express. For example, Table 7 shows the fitness calculation for the most popular conceptual models, such

Table 4: Possible combinations among elements and links when applying the interpretation procedure (This table is symetrical.)

	entity [repl]	entity [notrepl]	process	predicate	transition	message	constraint	value	statespace
Entity [repl]	spec subs pof rel activate								
entity [notrepl]	spec pof rel bel activate	spec pof rel activate							
Process	pof sends receives	pof sends receives - activate	spec pof activate						
Predicate	operand	operand	- simulate	operand					
Transition	stimulus response	stimulus response	stimulus response	stimulus response	stimulus response				
Message	- sends - receives	- sends - receives	- sends - receives	- operand	- stimulus - response	pof			
Constraint	- operand	- operand	- pof	- operand	- stimulus - response	- operand	operand		
Value	pof	pof	affect - sends -receives pof	- operand	- stimulus - response	- operand	- operand	spec pof	
Statespace	pof	pof	- sends - receives pof	- operand	- stimulus - response	- operand	- operand	hval	Pof

Table 5: Requirements Canonical Model



	Statements		
Sets:	**Entity[repl]**: Rooms		
	Entity[repl]: Wards		
	Entity[repl]: Complaints		
	Entity[repl]: Patients		
	Entity[repl]: Doctors 1		
	Entity[repl]: Doctors 2		
	Entity[repl]: Emergency doctors		
Subsets:	**Entity[repl]**: Waiting list	**Subs**: subs	**Entity[repl]**: Patient
	Entity[repl]: Emergency dept.	**Subs**: subs	**Entity[repl]**: Patients
	Entity[repl]: Doctors 2	**Subs**: subs	**Entity[repl]**: Doctors 1
	Entity[repl]: Emergency doctors	**Subs**: subs	**Entity[repl]**: Doctors 1
Individuals:	**Entity[notrepl]**: Complaint	**Bel**: bel	**Entity[repl]**: Complaints
	Entity[notrepl]: Room	**Bel**: bel	**Entity[repl]**: Rooms
	Entity[notrepl]: Patient 1	**Bel**: bel	**Entity[repl]**: Waiting list
	Entity[notrepl]: Patient 2	**Bel**: bel	**Entity[repl]**: Emergency dept.
	Entity[notrepl]: Ward	**Bel**: bel	**Entity[repl]**: Wards
	Entity[notrepl]: Doctor 1	**Bel**: bel	**Entity[repl]**: Doctors 2
	Entity[notrepl]: Doctor 2	**Bel**: bel	**Entity[repl]**: Emergency doctors
	Definitions		
Index	**Definition**		
	Propositions		
Index	**Association**	**Concept-1**	**Concept-2**
1	**Entity[notrepl]**: Hospital 123	**Rel**: admits	**Entity[repl]**: Patients
2	**Entity[repl]**: Doctors 2	**Rel**: are specialized in	**Entity[notrepl]**: Complaint
3	**Entity[notrepl]**: Patient 1	**Rel**: suffers	**Entity[notrepl]**: Complaint
4	**Entity[notrepl]**: Patient 1	**Rel**: is assigned	**Entity[notrepl]**: Room
5	**Entity[notrepl]**: Ward	-**Pof**: has	**Entity[repl]**: Room
6	**Entity[notrepl]**: Ward	**Rel**: is assigned to	**Entity[notrepl]**: Complaint
7	**Entity[notrepl]**: Doctor 1	**Rel**: is reference physician of	**Entity[notrepl]**: Patient 1
8	**Entity[notrepl]**: Doctor 2	**Rel**: treats in emergency dept.	**Entity[notrepl]**: Patient 2
9	**Entity[notrepl]**: Doctor 2	**Rel**: is reference physician of	**Entity[notrepl]**: Patient 2
10	**Entity[notrepl]**: Patient 2	**Rel**: is assigned	**Entity[notrepl]**: Room
11	**Entity[notrepl]**: Patient 2	**Rel**: suffers	**Entity[notrepl]**: Complaint
12	p10	**Constraint**: no later than	**Value**: 3 hours

as the data flow diagram (DFD), the entity-relationship diagram (ER), the class diagram (CD), the state transition diagram (STD), the statechart (SCT) and use cases (UC).

Table 7 shows that the best-suited conceptual model is the class diagram, as its fitness is greater than all the other conceptual models (or, at least, of all the ones that have been considered in the calculation). The fitness value of the class diagram is 0.71, which means that it can express 71% of all the RCM propositions.

Determining Method and Technique Fitness

As mentioned above, different methods and techniques employ different conceptual models, such as data flow diagrams, entity-relationship diagrams, use cases, state transition diagrams, etc. This means that, once we have determined the fitness of a model, this fitness can be extrapolated to the methods and techniques that use this model. For example, after determining the fitness of a data flow diagram, we can consider that all the methods and techniques that use this model, that is, all the structured methods and techniques, will be equally fit. The same could be said for the other models, such as use cases or state transition diagrams with respect to object-oriented and real-time methods and techniques, respectively.

But the situation is not as simple as this, because most methods and techniques use more than one conceptual model, with the aim of expressing different viewpoints about the problem domain. Accordingly, for example, the structured methods and techniques use, for example, DFD and ER, whereas the object-oriented methods and techniques use, among others, CD and UC. This means that we will have to consider all the conceptual models used jointly rather than each one separately to determine the fitness of a method or technique.

Table 6: Identification table for the class diagram

	entity [repl]	entity [notrepl]	process	predi-cate	transi-tion	message	constraint	value	statespace
Entity [repl]	spec subs pof rel								
entity [notrepl]	spec pof rel bel	spec pof rel							
Process	pof	pof							
Predicate									
Transition									
Message	- receives	- receives							
Constraint									
Value	pof	pof							
Statespace	pof	pof							

Apart from the fitness of the individual models, Table 7 can be slightly modified to calculate the fitness of the methods and techniques considered as sets of models. For this purpose, we use the same conceptual model identification tables as described. However, the fitness calculation for methods and techniques differs in that the weighted sum is calculated considering all the conceptual models used by a given method or technique. For example, Table 8 shows the fitness calculation for two generic, structured and object-oriented methods, characterized by the DFD-ER and CD-CU models, respectively.

Table 8 indicates that the best-suited method, with a fitness of 0.71, is the object-oriented method. This matches the results of Table 7, where the best-suited conceptual model was the class diagram. Note, however, that this will not necessarily be true in every case.

Table 7: Determination of conceptual model fitness

	Models			DFD	ER	CD	DTE	STC	UC
	Entity[repl]: Rooms			X	X	X			X
	Entity[repl]: Wards			X	X	X			X
	Entity[repl]: Complaints			X	X	X			X
	Entity[repl]: Patients			X	X	X			X
	Entity[repl]: Doctors_1			X	X	X			X
	Entity[repl]: Doctors_2			X	X	X			X
	Entity[repl]: Doctors			X	X	X			X
	Entity[repl]: Waiting list	Subs: subs	Entity[repl]: Patient			X			
	Entity[repl]: Emergency dept.	Subs: subs	Entity[repl]: Patients			X			
	Entity[repl]: Doctors_2	Subs: subs	Entity[repl]: Doctors_1			X			
	Entity[repl]: Emergency doctors	Subs: subs	Entity[repl]: Doctors_1			X			
	Entity[notrepl]: Complaint	Bel: bel	Entity[repl]: Complaints						
	Entity[notrepl]: Room	Bel: bel	Entity[repl]: Rooms						
	Entity[notrepl]: Patient_1	Bel: bel	Entity[repl]: Waiting list						
	Entity[notrepl]: Patient_2	Bel: bel	Entity[repl]: Emergency dept.						
	Entity[notrepl]: Ward	Bel: bel	Entity[repl]: Wards						
	Entity[notrepl]: Doctor_1	Bel: bel	Entity[repl]: Doctors						
	Entity[notrepl]: Doctor_2	Bel: bel	Entity[repl]: Emergency doctors						
p1	Entity[notrepl]: Hospital 123	Rel: admits	Entity[repl]: Patients		X	X			
p2	Entity[repl]: Doctors_2	Rel: are specialized in	Entity[notrepl]: Complaint		X	X			
p3	Entity[notrepl]: Patient_1	Rel: suffers	Entity[notrepl]: Complaint		X	X			
p4	Entity[notrepl]: Patient_1	Rel: is assigned	Entity[notrepl]: Room		X	X			
p5	Entity[notrepl]: Ward	-Pof: has	Entity[repl]: Rooms			X			
p6	Entity[notrepl]: Ward	Rel: is assigned to	Entity[notrepl]: Complaint		X	X			

Table 7: Determination of conceptual model fitness (continued)

p7	Entity[notrepl]: Doctor_1	Rel: is reference physician of	Entity[notrepl]: Patient_1		X	X			
p8	Entity[notrepl]: Doctor_2	Rel: treats in emergency dept.	Entity[notrepl]: Patient_2		X	X			
p9	Entity[notrepl]: Doctor_2	Rel: is reference physician of	Entity[notrepl]: Patient_2		X	X			
P10	Entity[notrepl]: Patient_2	Rel: is assigned	Entity[notrepl]: Room		X	X			
P11	Entity[notrepl]: Patient_2	Rel: suffers	Entity[notrepl]: Complaint		X	X			
P12-1	Constraint: no later than	Operand:	P10						
P12-2	Constraint: no later than	Operand:	Value: 3 hours						
Fitness				.23	.55	.71	.0	.0	.23

Finally, it should be mentioned that it is possible, although not strictly necessary from the viewpoint of method and technique fitness calculation, to derive the conceptual models (such as the class diagram or data flow diagram) from the information contained in the GCM. This derivation is a fully deterministic task, because the RCM has a well-defined meaning in terms of constructors like classes, processes, states, etc., which are the same constructors as used by the conceptual models.

A derivation procedure based on the use of a set of derivation tables and rules has been defined to get the conceptual models used by the different development methods and techniques. There are as many tables as there are possible target conceptual models (Dieste, 2003). Each derivation table contains all the possible combinations of Canonical Model elements that can be expressed in a given conceptual model, along with the expression of this combination in the particular format used by the conceptual model in question (graphs, text, tables, etc.). These tables and rules can be used to get fragments of the desired conceptual model from the propositions it expresses. For example, Table 9 shows a fragment of the derivation table for the class diagram.

For example, from the proposition Entity[notrepl]: Hospital 123 Rel: admits Entity[repl]: Patients, we can get the fragment shown in Table 10, as the derivation table contains an entry Entity[repl] Rel Entity[repl]".

The different fragments can then be assembled, unambiguously, to get the final version of the desired conceptual model. The diagram output for the case examined in the example given in Table 9 is shown in Figure 6.

The diagram shown in Figure 6 can be later modified to improve or add to diagram aspects and make the resultant class diagram clearer and simpler.

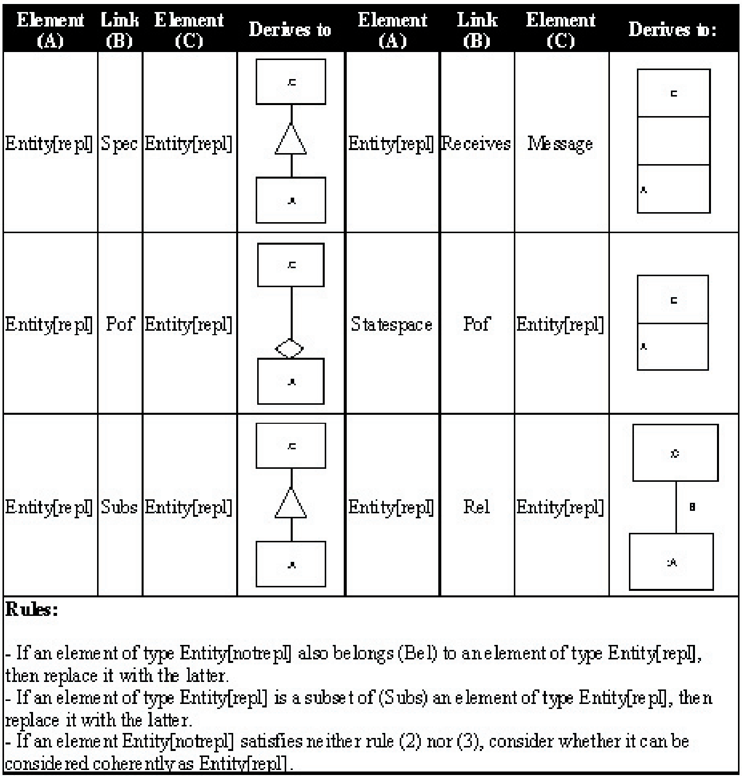
Table 8: Determination of method and technique fitness

	MT Models			Structured		Object Oriented	
				DFD	ER	CD	UC
	Entity[repl]: Rooms			X	X	X	X
	Entity[repl]: Wards			X	X	X	X
	Entity[repl]: Complaints			X	X	X	X
	Entity[repl]: Patients			X	X	X	X
	Entity[repl]: Doctors_1			X	X	X	X
	Entity[repl]: Doctors_2			X	X	X	X
	Entity[repl]: Doctors			X	X	X	X
	Entity[repl]: Waiting list	Subs: subs	Entity[repl]: Patient			X	
	Entity[repl]: Emergency dept.	Subs: subs	Entity[repl]: Patients			X	
	Entity[repl]: Doctors_2	Subs: subs	Entity[repl]: Doctors_1			X	
	Entity[repl]: Emergency doctors	Subs: subs	Entity[repl]: Doctors_1			X	
	Entity[notrepl]: Complaint	Bel: bel	Entity[repl]: Complaints				
	Entity[notrepl]: Room	Bel: bel	Entity[repl]: Rooms				
	Entity[notrepl]: Patient_1	Bel: bel	Entity[repl]: Waiting list				
	Entity[notrepl]: Patient_2	Bel: bel	Entity[repl]: Emergency dept.				
	Entity[notrepl]: Ward	Bel: bel	Entity[repl]: Wards				
	Entity[notrepl]: Doctor_1	Bel: bel	Entity[repl]: Doctors				
	Entity[notrepl]: Doctor_2	Bel: bel	Entity[repl]: Emergency doctors				
p1	Entity[notrepl]: Hospital 123	Rel: admits	Entity[repl]: Patients		X	X	
p2	Entity[repl]: Doctors_2	Rel: are specialized in	Entity[notrepl]: Complaint		X	X	
p3	Entity[notrepl]: Patient_1	Rel: suffers	Entity[notrepl]: Complaint		X	X	
p4	Entity[notrepl]: Patient_1	Rel: is assigned	Entity[notrepl]: Room		X	X	
p5	Entity[notrepl]: Ward	-Pof: has	Entity[repl]: Rooms			X	
p6	Entity[notrepl]: Ward	Rel: is assigned to	Entity[notrepl]: Complaint		X	X	
p7	Entity[notrepl]: Doctor_1	Rel: is reference physician of	Entity[notrepl]: Patient_1		X	X	

Table 8: Determination of method and technique fitness (continued)

p8	Entity[notrepl]: Doctor_2	Rel: treats in emergency dept.	Entity[notrepl]: Patient_2		X	X	
p9	Entity[notrepl]: Doctor_2	Rel: is reference physician of	Entity[notrepl]: Patient_2		X	X	
P10	Entity[notrepl]: Patient_2	Rel: is assigned	Entity[notrepl]: Room		X	X	
P11	Entity[notrepl]: Patient_2	Rel: suffers	Entity[notrepl]: Complaint		X	X	
P12-1	Constraint: no later than	Operand:	P10				
P12-2	Constraint: no later than	Operand:	Value: 3 hours				
Sum				17	22		
Fitness				.55	.71		

Table 9: Derivation table for the class diagram

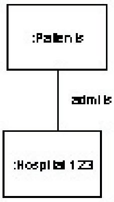

C
A
C
A
C
A
C
A
C
A
C
B
A

Element (A)	Link (B)	Element (C)	Derives to	Element (A)	Link (B)	Element (C)	Derives to:
Entity[repl]	Spec	Entity[repl]		Entity[repl]	Receives	Message	
Entity[repl]	Pof	Entity[repl]		Statespace	Pof	Entity[repl]	
Entity[repl]	Subs	Entity[repl]		Entity[repl]	Rel	Entity[repl]	

Rules:

- If an element of type Entity[notrepl] also belongs (Bel) to an element of type Entity[repl], then replace it with the latter.
- If an element of type Entity[repl] is a subset of (Subs) an element of type Entity[repl], then replace it with the latter.
- If an element Entity[notrepl] satisfies neither rule (2) nor (3), consider whether it can be considered coherently as Entity[repl].

Table 10: Derivation of a fragment of the SCM from the proposition Entity[notrepl]: Hospital 123 Rel: admits Entity[repl]: Patients. Note that the class diagram Derivation Rule (3) has been applied, considering Entity[notrepl]: Hospital 123 as Entity[repl]: Hospital 123

Element (A)	Link (B)	Element (C)	Derives to:
Entity[notrepl]: Hospital 123	Rel: admits	Entity[repl]: Patients	

FUTURE TRENDS

The proposed method paves the way for achieving a range of results. The most immediate is unquestionably the application of POAM to method and technique selection and adaptation to specific development projects. As indicated, this is possible because POAM provides a quantitative assessment of the fitness of the conceptual models and methods and techniques. However, the proposed fitness metric is far from perfect. On the contrary, this metric only refers to quasi-syntactic aspects of the model (how much information about the problem the model represents), and not to other quality criteria like functionality, maintainability, portability, reliability, efficiency and usability (ISO, 1999). Further investigation of conceptual model quality and their correspondence with particular problems is, therefore, required.

Additionally, given POAM's capability for deriving the conceptual models used by the methods and techniques, a second line of research would be to integrate POAM into the software development process as a previous step to the use of methods and techniques (Dieste et al., 2003). This line of research is particularly interesting because the use of POAM in the early stages of the development process separates analysis from later design, permitting greater freedom of choice of methods and techniques and, even, thanks to the ease with which the conceptual models are derived, changes of method or technique during the software development process.

CONCLUSIONS

Most software development techniques need to be adapted before they can be used in a particular software development project. This is because the methods and techniques can be applied to an indeterminate series of paradigmatic problems, but, as each problem moves further away from the ideal, their effectiveness falls.

Although method and technique adaptation is a recurrent theme in the literature, no papers that propose any sort of criterion or metric to assess method and technique fitness for a given problem have been published. Therefore, in this chapter, we have proposed a method that can be used to calculate method and technique fitness for specific problems

To calculate method and technique fitness, it is necessary to identify what makes a method and technique specifically oriented to a given class of problems. One of the reasons for this bias is the use, by the methods and techniques, of conceptual models. Because of their particular representation capabilities, conceptual models limit the problem domain aspects that a method and technique is capable of considering. The result is method and technique specificity for given problem types, particularly for the ones that have characteristics that can be adequately represented in the conceptual models proper to each method and technique. As it is the conceptual models that produce method and technique specificity, it is sufficient to determine the fitness of one or several conceptual models of a problem to determine the fitness of the methods and techniques.

The method proposed (POAM) can calculate the fitness of the conceptual models for particular problems. For this purpose, it uses a set of representation formalisms, called together generic conceptual model, which can represent the problem domain information without a previous categorization of the information on the basis of the standard concepts of conceptual models, like classes, objects, relationships process or states.

The generic conceptual model can, therefore, record the same information as a variety of conceptual models. This means that, instead of comparing conceptual models and problems to get a measure of fitness, it is possible to compare conceptual models with a representation of the problem recorded in the generic conceptual model. This comparison is made using a series of procedures that yield:

- A measure, termed fitness, which determines how suited each conceptual model is to the problem under analysis.
- If appropriate, the conceptual models required to pursue the remainder of the software development process.

The proposed method suggests a line of research that could lead to promising results. Specifically, a fitness measure of the conceptual models can improve method and technique adaptation procedures. Also, as POAM is capable of deriving the conceptual models used by the methods and techniques, the proposed method can be used as a prior step in the application of methods and techniques in software development projects.

REFERENCES

Brinkkemper, S., & Joosten, S. (1996). Special Issue on Method Engineering and Meta-Modeling. *Information and Software Technology*, *38*(4).

Bubenko, J.A. (1986). Information system methodologies - a research view. In T.W. Olle, H.G. Sol & A.A. Verrijn Stuart (Eds.). *Information System Design Methodologies: Improving the practice* (pp. 289-318). North-Holland.

Ceri, S. (Ed.). (1983). *Methodology and tools for database design*. North Holland.

Davis, A.M. (1993). *Software requirements: Objects, functions and states*. Prentice-Hall International.

Davis, A.M., Jordan, K., & Nakajima, T. (1997). Elements underlying the specification of requirements. *Annals of Software Engineering, 3*, 63-100.

Dieste, O. (2003). *POAM: Un método de análisis orientado a la necesidad*. Unpublished doctoral dissertation. Departamento de Informática, Universidad de Castilla-La Mancha.

Dieste, O., Genero, M., Juristo, N., Maté, J.L., & Moreno, A.M. (2003). A conceptual model completely independent of the implementation paradigm. *Journal of Systems and Software* (in press).

Dieste, O., Lopez, M., & Moreno, A.M. (1999). On the capability of analysis techniques in requirements engineering. *4th CAiSE/IFIP8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*. Heidelberg, Germany.

Firth, R., Wood, W., Pethia, R., Roberts Gold, L., Mosley, V., & Dolce, T. (1987). *A classification scheme for software development methods*. (CMU/SEI-87-TR-041). Software Engineering Institute, Carnegie Mellon University, Pennsylvania, USA.

Fowler, M., & Kobryn, C. (2002). Customizing UML for fun and profit. *Software Development*, *10*(7), July.

Fowler, M., & Scott, K. (1999). *UML distilled: a Brief guide to the standard object modeling language (2nd Edition)*. Addison-Wesley.

Glass, R.L., & Vessey, I. (1995). Contemporary application-domain taxonomies. *IEEE Software 12*(4), 63-76.

Hardy, C., Thompson, J., & Edwards, H. (1995). The use, limitations and customization of structured systems development methods in the United Kingdom. *Information and Software Technology*, *37*(9), 467-477.

ter Hofstede, A.H.M., & Verhoef, T.F. (1997). On the feasibility of situational method engineering. *Information Systems*, *22*(6/7), 401-422.

ISO. (1999). ISO/IEC 9126 - *Information Technology - Software product evaluation – Quality characteristics and guidelines for their use*.

Kirikova, M., & Bubenko, J.A. (1994). Enterprise modelling: Improving the quality of requirements specification. *Information systems research seminar in Scandinavia*, IRIS-17, Oulu, Finland.

Mayr, H.C., & Kop, C. (1998). Conceptual predesign - Bridging the gap between requirements and conceptual design. *Proceedings of the International Conference on Requirements Engineering, IEEE Computer Society Press,* (pp. 90-98).

Mylopoulos, J., Borgida, A., Jarke, M., & Koubarakis, M. (1990). TELOS: Representing knowledge about information systems. *ACM Transactions on Office Information Systems*, *8*(4), 325-362.

Nissen, H.W., Jeusfeld, M.A., Jarke, M., Zemanek, G.V., & Huber, H. (1996). Managing multiple requirements perspectives with metamodels. *IEEE Software*, *13*(2), 37-48.

Novak, D., & Gowin, D.B. (1984). *Learning to learn*. Cambridge University Press.

Russo, N., Wynekoop, J., & Walz, D. (1995). The use and adaptation of system development methodologies. *Proceedings of International Conference of IRMA (International Resources Management Association*, Atlanta, Georgia, USA.

van Lamsweerde, A., Dardenne, A., Delcourt, B., & Dubisy, F. (1991). The KAOS Project: Knowledge acquisition in automated specification of software. *Proceedings AAAI Spring Symposium Series*. University of Stanford.

Webster, D.E. (1988). Mapping the design information representation terrain. *IEEE Computer*, *21*(12), 8-23.

Woods, W. (1975). What's in a link: Foundations for semantic networks. In D.G. Bobrow & A. Collins (Eds.). *Representation and understanding: Studies in cognitive science* (pp. 35-82). Academic Press.

Yu, E. (1995). *Modelling strategic relationships for process reengineering*. PhD Dissertation, Dept. of Computer Science, Univ. of TorontoZave, P. (1990). A comparison of the major approaches to software specification and design. In R.H. Thayer & M. Dorfman (Eds.). *System and software requirements engineering* (pp. 197-199). Computer Society Press.

Chapter X

Toward an Extended Framework for Human Factors Research on Data Modeling

Heikki Topi, Bentley College, USA

V. Ramesh, Indiana University, USA

ABSTRACT

This study reviews and synthesizes over 15 years of research on human factors issues in conceptual data modeling. In addition to analyzing the variables used in earlier studies and summarizing the results of this stream of research, we propose a new framework to help with future efforts in this area. We also identify several key areas for future research and highlight the importance of building a strong theoretical foundation and using it to guide future empirical studies. It is our hope that this chapter allows both scholars and practitioners to utilize the results of existing research better and encourages continued work on conceptual data modeling.

INTRODUCTION

Conceptual data modeling continues to be an integral part of the foundation on which information systems are built. Depending on the development methodologies that are used for a particular project, the terms and methods used for conceptual data modeling vary, but in practice, a clear majority of methodologies used for systems development include a set of tools and methods for modeling data at the conceptual level. Therefore, it is not surprising that research in IS and its reference disciplines has shown a significant interest in various aspects of data modeling for the past 20 years. The focus of this chapter is on research that examines the *usability* of various conceptual data modeling approaches, that is, research that investigates human factors issues in conceptual data modeling. We review and analyze this literature and suggest several new directions for further research.

BACKGROUND

The concept of *data modeling* has been used with a variety of different meanings within various areas of study and practice. However, within the organizational context the core idea underlying all the definitions is the same: A data model is used for describing entities[1] and their relationships within a real world domain. For example, McFadden, Hoffer, and Prescott (1999) define a data model as "an abstract representation of the data about entities, events, activities, and their associations within an organization". A data model is an abstraction and a simplification of the domain it describes and thus, it always represents a limited part of reality.

The main focus of this chapter, *conceptual* data modeling, requires further clarification. Based on the ANSI/SPARC definition, a conceptual data model is any model that is independent of the underlying hardware and software. This means that using this definition, models created using formalisms ranging from the relational model to the semantically rich variants (Teorey, Yang & Fry, 1986) of Entity-Relationship modeling (Chen, 1976; Hull & King, 1987) can be considered to be at the conceptual level. A more restrictive definition of a conceptual model can be found in Batra and Davis (1992). They define a conceptual model as one that is capable of capturing the structure of the database along with the semantic constraints into a model that is easy to understand, does not contain implementation details, and can be used to communicate with users. A key criterion in the above definition is *the independence of modeling from the implementation technology*. This means that in order to be categorized as a conceptual model the representation must not be dependent on the characteristics of the database technologies available (e.g., relational, object-oriented, object-relational, network, or hierarchical).

We believe that both of the definitions presented above are, however, somewhat misleading because a true conceptual data model should capture the essential data characteristics of the *domain of interest*, and not necessarily the structure of the database. Thus, we define a conceptual data model as a set of constructs that can be used to create an abstraction of reality, that is, a representation capable of capturing the data-oriented (as opposed to process-oriented) aspects of a domain of interest in a manner that is unambiguous and easy to understand for analysts, designers, and users alike. Note that this definition does not have any references to a database structure. This is because we believe that not everything captured in a representation created using a conceptual data model will (or needs to) be reflected in a database or the eventual system being developed.

Based on the above definition of conceptual data modeling, one can synthesize at least five different uses for conceptual data models (Batra, Hoffer & Bostrom, 1990; Cambell, 1992; Juhn & Naumann, 1985; Kung & Solvberg, 1986): 1) a communication tool between analysts and users for the discovery (elicitation and representation) and validation stages of the systems analysis process, 2) a mechanism that helps analysts understand the domain of interest, 3) a formal conceptual foundation for organizational information systems at various levels (a common accepted model of reality and a communication tool between IS professionals, e.g., analysts and developers), 4) a foundation for applications developed by end users, and 5) an essential part of the system documentation for the maintenance of the system.

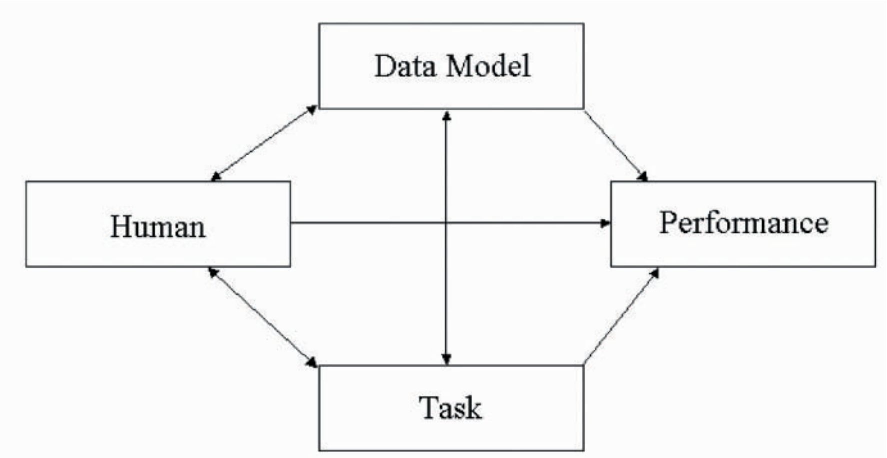
The main focus of this chapter is to examine research on the *human factors* issues in data modeling, that is, research that employs social science methods such as laboratory experiments, observations, and interviews to evaluate and improve the *usability* of the systems. Batra and Srinivasan define usability as "the ability of the user to represent a problem in a computing environment and effectively work with that representation" (1992, p. 395). Thus, two important research questions of human factors research on data modeling have traditionally been as follows: 1) How do the characteristics of the available tools affect users' ability to succeed in their tasks (i.e., what is the level of usability of the tools?), and 2) how satisfied are the users with the tools?

REVIEW OF PRIOR RESEARCH

In this section, we review the previous human factors research on data modeling. This review is based on a careful analysis of existing studies published in academic journals or in the Proceedings of the ICIS conference[2] that have *empirically* evaluated some aspect of the *usability* of conceptual data modeling tools and methods[3]. After a comprehensive search, we identified 31 articles published after (and including) Brosey and Shneiderman's early work in 1978 (Brosey & Shneiderman, 1978). A summary table of these studies is presented in Appendix A. The table includes a description of the independent variables (IV), dependent variables (DV), research tasks, and the most important results.

First, we will discuss the typical research variables used in these studies, and then, review the most important empirical findings.

Figure 1: Widely used framework for human factors research on data modeling (see, for example, Batra et al., 1990)



Variables of Interest in Empirical Studies

Research framework. Figure 1 includes a schematic representation of the research framework that has been used either explicitly (as by Batra et al., 1990) or implicitly in many of the earlier studies. *Human* refers to the individual level factors related to the characteristics of the individuals who perform the data modeling tasks, *Data Model* is used in this context to describe the differences between the data modeling formalisms, and *Task* refers to the characteristics of the tasks of interest related to data models, such as model creation, comprehension, or validation. The model indicates a reciprocal relationship between Human, Data Model, and Task, which all, in turn, have an impact on the quality of the resulting data model, that is, (human) Performance in the data modeling task. Variables in the Human, Data Model, and Task categories have been used in earlier studies as independent and control variables, as indicated in the discussion below, and Performance is a natural dependent variable in the studies.

Independent variables. The most frequently used independent variable in the earlier studies has been the data modeling approach or *data model,* as it is called by, for example, Batra and Davis (1992) and Navathe (1992) and in the research framework in Figure 1. In early research, Brosey and Shneiderman (1978) compared hierarchical and relational data models, whereas several later studies have compared different types of semantic and relational data models (Amer, 1993; Batra & Antony, 1994; Batra et al., 1990; Jarvenpaa & Machesky, 1989; Juhn & Naumann, 1985; Liao & Palvia, 2000; Sinha & Vessey, 1999) and/or two different semantic data models (Kim & March, 1995; Lee & Choi, 1998; Liao & Palvia, 2000; Nordbotten & Crosby, 1999; Palvia, Liao & To, 1992). Several of the most recent studies have compared semantic data models to object-oriented data models (Bock & Ryan, 1993; Hardgrave & Dalal, 1995; Lee & Choi, 1998; Liao & Palvia, 2000; Palvia et al., 1992; Shoval & Frumermann, 1994; Shoval & Shiran, 1997; Sinha & Vessey, 1999).

The next category of independent variables consists of *user characteristics* (*Human* in the research framework in Figure 1). The most commonly used independent variable is experience: The level of general MIS or programming experience was used as an independent variable in studies by Brosey and Shneiderman (1978) and Hoffer (1982), whereas Batra and Davis (1992), Weber (1996), and Lee and Choi (1998) analyzed the differences between subjects with various levels of data modeling experience. Ramesh and Browne (1999) differentiated between "database-knowledgeable" and "database novice" based on the subjects' understanding of basic ER concepts. Agarwal, Sinha, and Tanniru (1996) investigated the impact of the type of design experience on modelers' ability to use different formalisms for different tasks. In addition to programming expertise, Hoffer (1982) studied the effects of cognitive style, another category of individual differences. Finally, Siau, Wand, and Benbasat (1995) and Burton-Jones and Weber (1999) have explored the effects of the subjects' familiarity with the problem domain or the problem domain expertise.

A set of *task characteristics* (*Task* in the research framework in Figure 1) has also been used as an independent variable in the studies: Brosey and Shneiderman (1978) manipulated the *task type* (comprehension, problem solving, memorization), as did Batra and Antony (2001) (task's compatibility with a support tool). Hoffer (1982) varied the description of the situation on which the data model was based so that the situation was either related to a specific task or to the entire organization. *Task complexity* was used as an independent variable in Shoval and Even-Chaime (1987), Hardgrave and Dalal (1995), Weber (1996), and Liao and Palvia (2000). Jarvenpaa and Machesky (1989) investigated the effects of learning by using a within-subjects design and administering four data modeling tasks to each subject.

Dependent variables. The dependent variables can be divided into two broad categories: user performance and user attitudes. As seen earlier, the two main research questions of this area are related to modeling performance and user satisfaction, and, therefore, the widespread use of these dependent variables is understandable.

Performance has been divided into three subcategories: *model correctness* (also referred to as *procedural or skill knowledge* of the user by Jarvenpaa and Machesky (1989), measured by the characteristics of the end result of the modeling process), *time* used to create the solution, and *declarative knowledge* (understanding of the notation (Jarvenpaa & Machesky, 1989)). In most cases, the correctness of the model has been measured with the degree to which it corresponds to a predefined "correct" solution. Batra et al. (1990) were the first to refine the concept of correctness by measuring the correctness of various *facets* or structural elements of the model (entities, identifiers, descriptors, categories, and five different types of relationships: unary, binary one-to-many (1:M), binary many-to-many (M:N), ternary one-to-many-to-many (1:M:N), and ternary many-to-many-to-many (M:N:O)). The same facet structure was used later by Bock and Ryan (1993), Shoval and Shiran (1997), Lee and Choi (1998), and Liao and Palvia (2000). Kim and March (1995) divided the analysis of model correctness into *syntactic and semantic* categories: Syntactic correctness refers to users' ability to understand and use the constructs of the modeling formalism, whereas semantic correctness is the extent to which the data model corresponds to the underlying semantics of the problem domain. Another widely used measure of performance has been the time it takes to finish a modeling or model comprehension task (Hardgrave & Dalal, 1995; Jarvenpaa & Machesky, 1989; Lee & Choi, 1998; Liao & Palvia, 2000; Palvia et al., 1992; Shoval & Even-Chaime, 1987; Shoval & Shiran, 1997).

The *user attitudes* measured within this area of research are confidence (Hoffer, 1982), preference to use a certain model (Shoval & Even-Chaime, 1987; Shoval & Shiran, 1997), perceived value of the modeling formalism (Kim & March, 1995), and perceived ease-of-use (Batra et al., 1990; Hardgrave & Dalal, 1995; Kim & March, 1995).

In a study in which the dependent variable does not belong to either one of the main categories, *data model characteristics* were the main point of interest for Hoffer (1982). His study focused on the nature of the data models which the subjects created when they were able to freely choose the way to describe a structure of a database. The two characteristics of the model in his study were "image architecture" and "image size", that is, the modeling approach chosen and the number of entities.

Identified control variables. By investigating the nature of the explicitly identified control variables in previous research, it is possible to find potential independent variables of interest for future research, as well as summarize the variables that have to be controlled in future studies. *User characteristics* (*Human* in the framework in Figure 1) is the first category of specific control variables in the earlier studies. The most common individual variable in the user characteristics category is experience. The most common types of experience discussed in prior research are general work experience (Batra et al., 1990; Batra & Kirs, 1993; Jarvenpaa & Machesky, 1989; Juhn & Naumann, 1985; Liao & Palvia, 2000), general computer/IS experience (Batra et al., 1990; Batra & Kirs, 1993; Jarvenpaa & Machesky, 1989; Juhn & Naumann, 1985; Liao & Palvia, 2000), and database experience (Batra et al., 1990; Batra & Kirs, 1993; Jarvenpaa & Machesky, 1989; Juhn &

Table 1: Variables identified in human factors research on conceptual data modeling

Variable Type	Variable Category	Representative Examples
Independent variables	Data modeling formalism (Data Model)	Hierarchical vs. relational Relational vs. semantic Semantic vs. semantic Semantic vs. object-oriented
	User characteristics (Human)	General MIS experience Programming experience Data modeling experience Other modeling experience Cognitive style
	Task characteristics (Task)	Task type Task complexity
Dependent variables	Performance	Model correctness (facets, syntactic vs. semantic) Time Knowledge of the formalism
	User attitudes	Confidence Preference Perceived value Ease-of-use
Control variables	User characteristics (Human)	Work experience General IS/computer experience Database experience Age Education Intellectual ability Cognitive style
	Task characteristics	Complexity Time Structure Difficulty

Naumann, 1985; Liao & Palvia, 2000). Age (Liao & Palvia, 2000), education (Jarvenpaa & Machesky, 1989; Liao & Palvia, 2000), intellectual ability (Juhn & Naumann, 1985), and cognitive style measured with LSI (Jarvenpaa & Machesky, 1989) have been other types of individual differences which have been controlled. In most studies, user characteristics have been controlled by selecting subjects from a homogenous population and by random assignment to experimental conditions.

Controlling for *task characteristics* (*Task* in the framework in Figure 1) by keeping them the same across the treatments is a natural approach and not very interesting at the category level. Jarvenpaa and Machesky (1989) and Batra and Kirs (1993) both list specific characteristics of the task which were kept constant; these were complexity, structure, difficulty, and time, which are all related to a more general concept of difficulty. Kim and March (1995) specifically mentioned task complexity and time as task characteristics that were controlled. *Training* was also identified as a significant control variable by Batra and

Kirs (1993) and Kim and March (1995); details controlled in these experiments include trainer characteristics and instructional examples. Table 1 summarizes the variables used in prior research.

Key Findings from Prior Studies

The results from the empirical studies reviewed can be categorized as follows: a) Effects of *data modeling formalism* on *user performance and attitudes,* b) Effects of *user characteristics on user performance and attitudes,* and c) Effects of *task characteristics* on *user performance and attitudes*. Most of the studies have focused on the first category. In addition to the associations between research variables, we will review the results for various task components (facets) and the main lessons from the studies with a process focus.

Effects of data modeling formalism on user performance and attitudes. The studies that have investigated the effects of the data modeling formalism on performance and attitudes can be divided into the following subcategories: a) those comparing a semantic model to the relational model, b) those comparing two semantic models to each other, and c) those comparing a semantic model with object-oriented models.

In the first subcategory, the seven studies (Amer, 1993; Batra & Antony, 1994; Batra et al., 1990; Jarvenpaa & Machesky, 1989; Juhn & Naumann, 1985; Liao & Palvia, 2000; Sinha & Vessey, 1999) that have investigated the differences between the ER/EER and relational modeling formalisms have all found support for the positive effect of the use of the ER/EER model on one or several aspects of modeling performance. The studies provide strongest support to ER/EER's advantage in modeling binary 1:M and binary M:N relationships; four of the studies (Amer, 1993; Batra et al., 1990; Liao & Palvia, 2000; Sinha & Vessey, 1999) support this finding, whereas the other findings related to the identification of relationships and cardinalities, faster learning, understanding the notation, modeling ternary 1:M:N and unary relationships, and generalization modeling are all based on only one of the studies. For the binary relationships, these results are in line with those of Cao, Nah, and Siau's (2000) meta-analysis, which included both modeling and query writing studies; our analysis did not find the same strong support for ER/EER's advantage over relational model in modeling ternary 1:M:N relationships as theirs did. The one study (Shoval & Even-Chaime, 1987) that focused on the relationship between the relational model and a non-ER semantic model, NIAM, found the relational model to lead to better user performance and to require less time. As to the effects of the modeling formalism choice between semantic and relational models and the user attitudes, the results are scarce and inconclusive: Jarvenpaa and Machesky (1989) found that subjects perceived the ER/EER model to be easier to use than the relational model, but Shoval and Even-Chaime (1987) found that the subjects preferred the relational model over NIAM.

Six studies (Hardgrave & Dalal, 1995; Lee & Choi, 1998; Liao & Palvia, 2000; Shoval & Frumermann, 1994; Shoval & Shiran, 1997; Sinha & Vessey, 1999) have investigated the effects of the choice between object-oriented models (although not consistently the same ones) and ER/EER. The lack of consistency between the studies makes it difficult to draw any general conclusions, but the direction of the studies seems to suggest that using the ER/EER model leads to better performance in modeling tasks. The studies together indicate that the use of ER/EER has a positive effect on modeling performance in five of the modeling facets (unary 1:1, binary 1:1 and 1:M, and ternary 1:M:N, and M:N:O), but, unfortunately, the findings come from different studies that do not provide support for each other's findings. The only result related to user attitudes in these studies was made by Shoval and Shiran (1997), who found that ER/EER users' quality perceptions were higher than those of OO users.

Effects of user characteristics on performance and attitudes. Seven empirical studies have significant results regarding the effects of user characteristics on performance and attitudes, and all of them have focused on some type of task-related experience. The results do not, unfortunately, build a highly consistent image because every study has investigated a different aspect of experience. Therefore, the studies will be discussed here in chronological order. Batra and Davis (1992) confirmed that well-known process differences between novices and experts could be observed also within this domain. Siau et al. (1995) found out that domain familiarity did not have an impact on the choice between optional and mandatory relationships; subjects (experts) almost invariably chose to use an optional relationship construct. According to Agarwal et al. (1996), subjects with experience in modeling with a process focus are able to utilize this experience when they are modeling behavior but not with data structures. Weber's (1996) results in his experiment using a recall task suggest that although NIAM experts' ability to recall model elements was slightly better than that of novices, their memory structures and recall strategies were the same. Lee and Choi's (1998) results regarding the differences between experienced ER modelers and novices are somewhat difficult to interpret, but it appears that in most respects ER experience led to higher performance with the other methods, too, although experienced modelers used more time. In all cases but one (ORM), experienced ER modelers perceived the methods to be easier to use than inexperienced modelers did. According to Ramesh and Browne (1999), "database-naive" subjects were better able to express causal relationships than "database-knowledgeable" subjects, and they attribute this to the inability of commonly used modeling formalisms to support the expression of causal relationships. Finally, Burton-Jones and Weber (1999) studied the effects of domain knowledge and ontological clarity of a representation on the subjects' ability to answer problem-solving questions. Their results provide limited support to the claim that ontological clarity is particularly important in cases when domain knowledge is low.

Effects of task characteristics on user performance and attitudes. None of the studies have directly focused on the effects of task characteristics on the main dependent variables, although four of them (Hardgrave & Dalal, 1995; Liao & Palvia, 2000; Shoval & Even-Chaime, 1987; Weber, 1996) used task complexity as an independent variable and all of them found a main effect for complexity on performance (in practice, this means that the experimental manipulation worked). This is understandable because in most cases, the focus is on the moderating effects of task characteristics on the effects of other variables on performance, particularly the model formalism and user characteristics.

Differences between facets. As discussed above, most of the studies have used some version of the facet structure for analyzing user performance since Batra et al. (1990) originally presented it. Five of them have analyzed user performance in one or several of these facets with measures that are similar to each other and give us an opportunity to review users' relative performance with various facets. The performance data per facet from these studies are included in Table 2; no aggregate data is presented here because it is not in all cases clear whether or not the methods have been similar enough to justify the use of composite measures. This data does, however, lead to the following observations: 1) Identifying and modeling ternary relationships correctly is difficult for novice users, and even in the relatively simple experimental tasks users' average performance level is often below 50%. The range

Table 2: User modeling performance by facet in empirical studies

	Batra et al., 1990		Batra & Kirs, 1993		Bock & Ryan, 1993		Shoval & Shiran, 1997		Liao & Palvia, 2000		
	Rel.	ER/ EER	Rel.	ER/ EER	ER/ EER	OO	ER/ EER	OO	Rel.	ER/ EER	OO
Entity					98.0	96.0	99.0	99.0			
Identifier	72.4	73.9			96.0	80.0			62.8	69.7	77.3
Descriptor							95.0	94.0			
Category					92.0	82.0	99.0	99.0			
Unary	68.3	55.2			96.0	64.0	88.0	70.0	59.9	40.0	50.0
Binary 1:M	54.4	84.9	50.6	81.2	89.0	88.0	83.0	89.0	54.2	83.8	73.9
Binary M:N	57.1	92.9	67.5	92.5	100.0	63.0	81.0	79.0	41.2	74.4	65.3
Ternary 1:M:N	8.3	41.3	46.9	60.0	47.0	44.0	85.0	68.0			
Ternary M:N:O	33.3	45.2	40.6	45.6	79.0	72.0	94.0	76.0	35.4	57.5	47.7

of performance levels is, however, very large, varying from 8.3% for 1:M:N relationships in Batra et al. (1990) to 94% for M:N:O relationships in Shoval and Shiran (1997). 2) Results are weak (below 70%) also for unary relationships, except with a semantic formalism (ER/EER) in Bock and Ryan (1993) and Shoval and Shiran (1997). The range is large also with this facet (from 40% to 96%). 3) With semantic and object-oriented modeling formalisms, users' average performance in modeling the binary relationships is consistently at a high level (above 80%), with the exception of binary M:N relationships in Liao and Palvia (2000). 4) Modeling identifiers, a seemingly simple task, appears to cause difficulties with all modeling formalisms, with typical performance levels around 70%.

Findings related to the data modeling process. Five of the studies included in this review analyzed some aspect of the process that subjects followed while creating a data model. As discussed earlier, Jarvenpaa and Machesky (1989) investigated whether the subjects chose a top-down or a bottom-up approach when constructing data models and whether the choice of the approach was dependent on the modeling formalism. They found that users of the ER-based Logical Data Structure model were more likely to use a top-down approach than the users of the relational model. Batra and Davis (1992) studied the protocol differences between novices and experienced data modelers and found broad support for several findings from prior research regarding the differences between these two groups: Experts had richer concept vocabulary and were better able to categorize constructs and automate processes, whereas novices were more likely to make a range of modeling errors. Batra and Sein (1994) analyzed at the individual level users' ability to improve the quality of their data modeling solutions based on feedback and found out that feedback can help users avoid errors in modeling ternary relationships. Srinivasan and Te'eni (1995) focused entirely on the results of the process analysis of a specific modeling behavior. Using verbalized protocols, they analyzed the use of several heuristics at various levels of abstraction to manage the complexity of the data modeling process. The most important results reported in Srinivasan and Te'eni (1995) were that efficient data modelers use specific heuristics to reduce the complexity of the problem, test models at regular intervals, and make orderly transitions from one level of abstraction of problem representation to another. In general, the study provides an important example of a research approach that makes it possible to evaluate data modeling at a detailed level as a problem solving process. Building on an important line of research, Batra and Antony (2001) investigated the effectiveness of a consulting system that is designed to reduce data modeling errors and found out that individuals with a low initial knowledge level benefited from the consulting system.

Effects of specific characteristics of conceptual modeling grammars. Several studies have been conducted building on the foundation of Wand and Weber's theoretical work (Wand, Monarchi, Parsons & Woo, 1995; Weber, 1997) on the use of ontology as a conceptual basis for constructing and evaluating conceptual modeling grammars. These theory-testing studies have focused on specific characteristics of the grammars and their impact on user performance in specific types of tasks. Weber (1996) utilized a strong theoretical foundation in cognitive psychology and philosophy to evaluate whether or not humans tend to see entities and attributes as distinct constructs, and his conclusion based on a memory recall experiment is that these, indeed, are separate elements. In another study building on the same theoretical foundation discussed already above in the context of domain familiarity, Burton-Jones and Weber (1999) confirmed their theory-based predictions that using relationships with attributes would have a negative impact on problem-solving performance in unfamiliar domains, but they also found out that this result did not hold in familiar domains. Finally, Bodart, Patel, Sim, and Weber (2001) studied the use of optional properties (attributes and relationships) in conceptual modeling. Their three-experiment study found that models with optional properties serve well when the purpose is to help users gain a surface-level understanding, but that optional properties should not be used if the users need a deep-level understanding. Building on a different theoretical foundation, Siau, Wand, and Benbasat (1997) investigated the effects of the use of structural constraints (such as explicit cardinality constraints); their results reveal that modeling experts pay much more attention to the structural constraints than to the surface semantics conveyed in textual descriptions.

Having reviewed the results of prior usability research on conceptual data modeling, we continue by evaluating the implications of these results and suggesting several new avenues for future research.

POTENTIAL FOR FUTURE RESEARCH

Given the maturity of data modeling in practice and the results summarized above, it would be easy to conclude that further human factors research related to conceptual data modeling may not add substantially to the existing body of knowledge. In the next section we hope, however, to demonstrate that because it has focused on a relatively narrow part of conceptual data modeling, prior research has left several potentially important questions still unanswered.

Most of the empirical studies reported above that have investigated conceptual data modeling from the human factors perspective are based on the same relatively simple model: in a controlled laboratory study, subjects with only modest experience complete one or several modeling tasks in which they create a graphical representation of an organizational situation based on a narrative using one or several conceptual data modeling formalisms. The results are typically evaluated by grading the models using a solution created by the researcher as a baseline; results achieved with different formalisms are then compared to each other with standard statistical techniques. This approach has definitely improved our understanding of the factors that affect subjects' ability to represent a case situation with graphical tools, and a controlled experiment is a perfectly valid methodology for investigating specific aspects of a cognitively complex task such as conceptual data modeling. Future research should, however, utilize the opportunities created by a richer set of background theories and research methodologies.

We present two key ideas that can help with future research efforts:

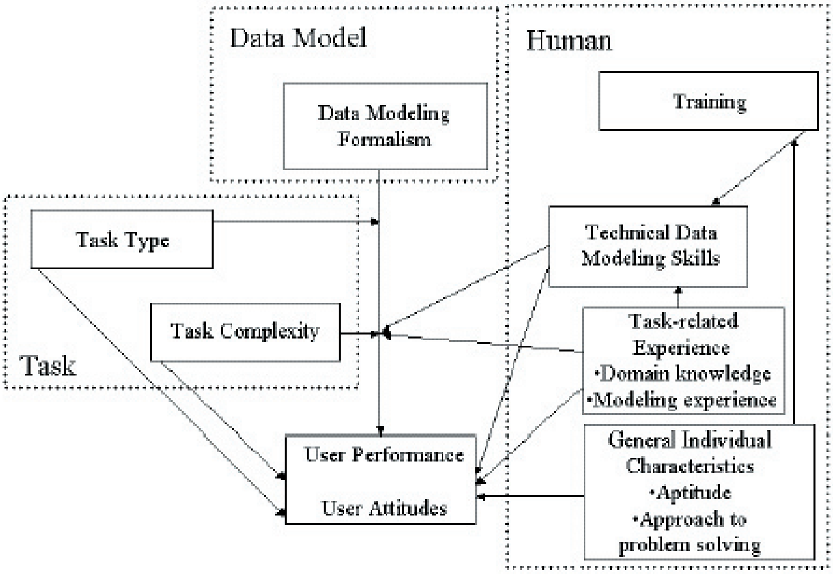
a) First, we note that because almost all of the research to date has focused on the technical characteristics of the modeling formalisms, we know very little about the effects of users' individual characteristics, task characteristics, or the interaction between the modeling formalism, user, and task. Below, we discuss a new framework that we hope will provide additional clarity to future research efforts.
b) Second, we observe that we do not have yet a good understanding of why certain formalisms work well in some situations and not in others; the mechanisms mediating the relationships between the main research variables are not clear. We provide several suggestions for research that can be used to strengthen our understanding in this area.

An Expanded Framework for Human Factors Research in Data Modeling

Our review of prior literature and additional conceptual analysis of this stream of research leads us to believe that the traditional framework that has been used to guide human factors research on data modeling (see Figure 1) can be improved and clarified. In this section, we present and justify the suggested changes, which have been incorporated into a new framework presented in Figure 2. We supplement the framework in Figure 2 by using the findings from the studies reviewed earlier as well as our theoretical understanding of the domain. It is, however, worth noting that the theoretical basis for this expanded framework as well as the Batra et al. (1990) framework lies in the classical general MIS task-technology-human research framework, which, in turn, is a derivation of Leavitt's (1965) organizational system model.

This framework was developed independently from Wand and Weber's general framework for research on conceptual modeling (Wand & Weber, 2002), and its context and intended uses are not as broad as those of Wand and Weber's framework. Our framework is specifically intended for guiding human factors/usability research on data modeling, whereas the Wand and Weber framework provides a comprehensive overall model for conceptual modeling research. For example, Wand and Weber include Social Agenda Factors as one of the critical conceptual factors; we have not included it in our framework because of our more narrow focus on usability. However, we fully acknowledge the potential importance of Social Agenda Factors as a broader contextual factor. Our Human category corresponds directly to Wand and Weber's Individual Difference Factors, and our Task category is similar to Wand and Weber's Task Factors. Most notably, Wand and Weber elegantly separate research issues related to Conceptual-Modeling Grammar and Conceptual-Modeling Method; in our framework, these are included in the Data Modeling Formalism category. We strongly encourage readers to consult Wand and Weber (2002) for a more elaborate categorization

Figure 2: Proposed framework for human factors research on data modeling



of research issues related to conceptual modeling grammars and methods. In particular, we feel that the two frameworks used in tandem can help foster productive research streams in the data modeling arena for many years.

As we have seen in the review of prior literature and summary of the results above and will discuss below, many of the relevant relationships are between specific components of the framework elements (see also Table 1). Hence, it is important to elaborate on the broad construct categories Task, Data Model, Human, and Performance. *Task Complexity* and *Task Type* should be presented as separate concepts, because these dimensions of the task are largely independent and their effects should be investigated separately from each other. For example, it is understandably possible to have various levels of complexity for comprehension, validation, and modeling tasks and both could be used separately as independent variables in the same study at the same time. As to *Human*, we can differentiate between multiple categories of individual characteristics which are independent from each other. Underlying all other aspects of an individual's performance are *general individual characteristics* such as intelligence, cognitive style, and problem-solving approach, which affect a particular individual's performance in all cognitive tasks. The only data modeling study so far that has explicitly used a variable from this category is Hoffer (1982). An individual also has *experience* in a variety of areas, many of which are potentially relevant to their performance in the task of interest (general problem-solving experience, programming experience, general modeling experience, modeling experience with specific formalism(s), etc.). This category of variables has been utilized widely in earlier research, as discussed in the review above (Agarwal et al., 1996; Batra & Srinivasan, 1992; Brosey & Shneiderman, 1978; Burton-Jones & Weber, 1999; Hoffer, 1982; Lee & Choi, 1998; Ramesh & Browne, 1999; Weber, 1996). Finally, an individual's *technical skills* in the use of a specific data modeling formalism should be conceptually separated as a factor affecting the user's performance. One of the reasons why it is essential to differentiate technical skills from other aspects is that this is the only subcategory of individual differences in this framework that can be affected by *training* (other factors that could be influenced by training include confidence, self-efficacy, task motivation, etc). Technical skills have been used as an independent variable in several studies (Batra & Antony, 2001; Weber, 1996). In general, the division of the framework elements into components forces us to specify the nature of the relationships of interest at a significantly more detailed level. This, in turn, will lead us closer to true theoretical models at least in part based on applicable theories from relevant reference disciplines, such as Anderson's ACT theory with its variants (Anderson, 1993), which was suggested as an important theoretical basis for research on information modeling (including conceptual data modeling) by Siau (1999).

Second, the framework should incorporate two different types of dependent variables to acknowledge the fact that we are not only interested in objective performance but also users' attitudes towards the tools, the tasks, and their own performance. The most often used non-performance dependent variables are ease-of-use perceptions (Batra et al., 1990; Hardgrave & Dalal, 1995; Kim & March, 1995; Lee & Choi, 1998) and modeling formalism preference (Batra & Sein, 1994; Kim & March, 1995; Shoval & Even-Chaime, 1987; Shoval & Shiran, 1997).

Third, the framework should acknowledge and explicitly incorporate the potentially complex moderating effects of other variables on the relationship between the data modeling formalism and user performance and attitudes. The direct effect of task complexity on the dependent variables, particularly performance, is seldom the main point of interest; in most cases, we are interested in the way different formalisms support users at various task complexity levels. The same is true with task type: a relevant research question is the suitability of various modeling formalisms for specific task types and thus, we should explicitly express in our research model that task type moderates the relationship between the data modeling formalism and the dependent variables. The best examples of this are the experiments by Kim and March (1995), who studied the use of two formalisms for user (validation) and analyst (modeling) tasks, and Lee and Choi (1998), who compared four different formalisms in two task types. The commonly used analysis of performance by facets (Batra et al., 1990; Bock & Ryan, 1993; Lee & Choi, 1998; Liao & Palvia, 2000; Shoval & Shiran, 1997) is, in fact, a form of analysis of the moderating effects of task type, because modeling a specific facet can be seen as a subtask. As discussed above in the summary of results, the facet being modeled often moderates the impact of a specific modeling formalism on performance.

Finally, the research framework should explicitly acknowledge that various individual characteristics have differential effects on user performance and attitudes and that many of the effects of individual differences moderate the relationship between the data modeling formalism and the dependent variables. In addition, some of the relationships between the categories of individual characteristics affect each other in a significant way: Task-related experience affects an individual's technical data modeling skills (in addition to training), and the general individual differences (such as intelligence) moderate the relationship between the training an individual receives and the individual's skills.

We believe that the use of the framework in Figure 2 and any extensions of it (particularly when used together with Wand and Weber's (2002) broader framework) would provide future human factors research on conceptual data modeling a stronger foundation and give the researchers an incentive to specify the relationships between the variables of interest at a more detailed level and present them better in relation to other, potentially significant variables.

New Areas of Focus

Finally, we would like to propose two additional foci for conceptual data modeling research: a) basic research on concept formulation, categorization, and usage, and b) applied research on data modeling processes.

First, as Wand and Weber (2002) point out, we need a better understanding of the psychological processes in data modeling and the ways the tools affect these processes. This will enable us to find a firm theoretical basis for human factors research on data modeling. Researchers in this area should be interested not only in the characteristics of the current models, but the reasons underlying the potential performance differences between various approaches to data modeling. Batra's (1993) framework of error behaviors and the introduction of the GEMS model to this domain by Batra and Antony (2001) are excellent steps in the right direction. As Siau (1999) points out, cognitive science is potentially a very useful reference discipline, especially the research in cognitive science that has its roots in cognitive psychology or in artificial intelligence (Batra, 1993; Henderson & Peterson, 1992; Rosch & Mervis, 1975; Rosch, Mervis, Gray, Johnson & Boyes-Braem, 1976; Smith & Medlin, 1981). Applied research in this field has been done, for example, in marketing and organizational behavior (for representative examples see Day & Lord, 1992; Fiol & Huff, 1992; Ozanne, Brucks & Grewal, 1992).

It is essential to point out here the very significant general theoretical foundation work Wand and Weber have done on several dimensions of conceptual modeling, particularly in demonstrating how ontology can be used as a basis for conceptual analysis. This research has been published in journal articles (Wand et al., 1995; Wand, Storey & Weber, 1999; Wand & Weber, 1993, 1995, 2002) and as a monograph (Weber, 1997). We believe that their work is an invaluable foundation for future conceptual and empirical work in this area, including the work on usability.

The essence of all modeling is in the identification of concepts and categorization of them (Booch, 1994, Chapters 1-4; Coad & Yourdon, 1991, Chapter 1). The links between theoretical research on categorization and data modeling are still somewhat weakly defined, although Parsons and Wand's (Parsons, 2003; Parsons & Wand, 1997, 2000) work is a very important contribution and an excellent example of the type of research that is needed in this area. An additional important contribution would be a conceptual analysis of the characteristics of various data modeling techniques compared with categorization theories (see Henderson and Peterson (1992) for a concise introduction) and an empirical verification of the results of this research. The central focus of this research should be on the relationships between individual abilities, individuals' histories, situation characteristics, perceptions of reality, and categorization behavior. On the other hand, it is very important to note that data models are not (or at least should not be) created in a social vacuum; a data model describes a collective cognitive view about an organization. If reality is socially constructed (Berger & Luckmann, 1967) and information processing is greatly affected by social structures and forces (Salancik & Pfeffer, 1978; Weick, 1979), a closer analysis of the impact of social forces on data modeling (Ram & Ramesh, 1998) is warranted, as was also suggested in Wand and Weber (2002).

Second, and in addition to research focusing on fundamental psychological and social psychological processes, rigorous applied empirical research and theory development is also needed. It is important that this work collectively covers all uses of conceptual data modeling (see, for example, the Background section of this chapter); much of prior research has focused on issues most closely associated with the communication between analysts and designers. In applied research, two important characteristics of the real world modeling tasks have to be taken into account. First, the process of model building, validation, and implementation is almost always iterative. Models are not built in a very limited amount of time and accepted without conceptual and empirical testing, or if they are, at least the implementation (and the implicit, but not the documented, data model) will be changed if modeling errors lead to application errors. Second, the elicitation, representation, and validation phases of the modeling process are normally closely integrated, and the separation of them in research environments is often artificial.

In addition to broader tasks, a richer set of methodologies is also needed. A quantitative analysis of results obtained in a laboratory environment is not enough. In addition, qualitative techniques and field data are needed. For example, Batra and Davis (1992) and Srinivasan and Te'eni (1995) used protocol analysis (Ericsson & Simon, 1993) to gain a deeper understanding of the modeling process. In-depth case studies, observations, and other methods that can be applied in field environments—for exploratory and later for theory testing purposes—are also necessary to analyze the real effects of data modeling in organizational environments. It is also important to continue research that studies how conceptual data modeling is, in practice, used in the broader context of systems development (see, for example, Batra and Marakas, 1995; Hitchman, 1995).

CONCLUSIONS

Conceptual data modeling forms an important foundation for systems development. In this chapter, we have reviewed the existing human factors research on conceptual data modeling. In addition, we proposed an extended framework and described avenues for further work in this area. Also, we emphasized the importance of continuing to build a stronger theoretical foundation based on the work in cognitive science and other relevant reference disciplines.

ACKNOWLEDGMENTS

Earlier versions of this chapter were published in the Proceedings of the Sixth CAiSE/IFIP8.1 Intl. Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'01) and in the Journal of Database Management (Topi & Ramesh, 2002). We gratefully acknowledge the highly valuable comments by the EMMSAD'01 participants and the reviewers and editors of all versions of this chapter.

ENDNOTES

[1] The concept of "entity" refers in this context not only to static objects but also to relevant activities and events within the domain of interest.

[2] We acknowledge that our sample may not include some relevant papers published in the proceedings of specialized conferences.

[3] Only those studies on object-oriented modeling have been included that have data modeling as their primary focus.

REFERENCES

Agarwal, R., Sinha, A.P., & Tanniru, M. (1996). The role of prior experience and task characteristics in object-oriented modeling: An empirical study. *International Journal Of Human-Computer Studies, 45*, 639-667.

Amer, T. (1993). Entity-relationship and relational database modeling representations for the audit review of accounting applications: An experimental examination of effectiveness. *Journal of Information Systems, 7*(1), 1-15.

Anderson, J.R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Batra, D. (1993). A framework for studying human error behavior in conceptual database modeling. *Information and Management, 25*(3), 121-131.

Batra, D., & Antony, S.R. (1994). Effects of data model and task characteristics on designer performance - a laboratory study. *International Journal Of Human-Computer Studies, 41*(4), 481-508.

Batra, D., & Antony, S.R. (2001). Consulting support during conceptual database design in the presence of redundancy in requirements specifications: an empirical study. *International Journal Of Human-Computer Studies, 54*(1), 25-51.

Batra, D., & Davis, J.G. (1992). Conceptual data modeling in database design - Similarities and differences between expert and novice designers. *International Journal Of Man-Machine Studies, 37*(1), 83-101.

Batra, D., & Kirs, P.J. (1993). The quality of data representations developed by nonexpert designers: An experimental study. *Journal of Database Management, 4*(4), 17-29.

Batra, D., & Marakas, G.M. (1995). Conceptual data modelling in theory and practice. *European Journal of Information Systems, 4*(3), 185-193.

Batra, D., & Sein, M.K. (1994). Improving conceptual database design through feedback. *International Journal Of Human-Computer Studies, 40*(4), 653-676.

Batra, D., & Srinivasan, A. (1992). A review and analysis of the usability of data management environments. *International Journal Of Man-Machine Studies, 36*(3), 395-417.

Batra, D., Hoffer, J.A., & Bostrom, R.P. (1990). Comparing representations with relational and EER models. *Communications of the ACM, 33*(2), 126-139.

Berger, P., & Luckmann, T. (1967). *The social construction of reality*. New York: Doubleday.

Bock, D.B., & Ryan, T. (1993). Accuracy in modeling with extended entity relationship and object oriented data models. *Journal of Database Management, 4*(4), 30-39.

Bodart, F., Patel, A., Sim, M., & Weber, R. (2001). Should optional properties be used in conceptual modelling? A theory and three empirical tests. *Information Systems Research, 12*(4), 384-405.

Booch, G. (1994). *Object oriented analysis and design with applications* (2nd ed.). Redwood City, CA: Benjamin/Cummings.

Brosey, M., & Shneiderman, B. (1978). Two experimental comparisons of relational and hierarchical database models. *International Journal Of Man-Machine Studies, 10*, 625-637.

Burton-Jones, A., & Weber, R. (1999). Understanding relationships with attributes in entity-relationship diagrams. *Proceedings of Twentieth International Conference on Information Systems*, Charlotte, NC.

Cambell, D. (1992). Entity-relationship modeling: One style suits all. *Data Base, 23*(5), 12-18.

Cao, Q., Nah, F., & Siau, K. (2000). A meta-analysis of relationship modeling accuracy: Comparing relational and semantic models. *Proceedings of the 6th Americas Conference on Information Systems*, Long Beach, CA.

Chen, P. (1976). The entity-relationship model - Toward the unified view of data. *ACM Transactions On Database Systems, 1*(1), 9-36.

Coad, P., & Yourdon, E. (1991). *Object-oriented analysis* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Day, D.V., & Lord, R.G. (1992). Expertise and problem categorization: The role of expert processing in organizational sense-making. *Journal of Management Studies, 29*(1), 35-47.

Ericsson, K.A., & Simon, H.A. (1993). *Protocol analysis. Verbal reports as data*. Cambridge, MA: The MIT Press.

Fiol, C.M., & Huff, A.S. (1992). Maps for managers: Where are we? Where do we go from here? *Journal of Management Studies, 29*(3), 267-285.

Hardgrave, B.C., & Dalal, N.P. (1995). Comparing object-oriented and extended-entity-relationship data models. *Journal of Database Management, 6*(3), 15-21.

Henderson, P.W., & Peterson, R.A. (1992). Mental accounting and categorization. *Organizational Behavior and Human Decision Processes, 51*(1), 92-117.

Hitchman, S. (1995). Practitioner perceptions on the use of some semantic concepts in the entity-relationship model. *European Journal of Information Systems, 4*(1), 31-40.

Hoffer, J.A. (1982). An empirical investigation into individual differences in database models. *Proceedings of the Third International Conference of Information Systems*, Ann Arbor, MI.

Hull, R., & King, R. (1987). Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys, 19*(3), 201-260.

Jarvenpaa, S.L., & Machesky, J.J. (1989). Data analysis and learning: An experimental study of data modeling tools. *International Journal Of Man-Machine Studies, 31*, 367-391.

Juhn, S., & Naumann, J.D. (1985). The effectiveness of data representation characteristics on user validation. *Proceedings of the Sixth International Conference on Information Systems*, Indianapolis, IN.

Kim, Y.G., & March, S.T. (1995). Comparing data modeling formalisms. *Communications of the ACM, 38*(6), 103-115.

Kung, C.H., & Solvberg, A. (1986). Activity modeling and behaviour modeling. In T. W. Olle, H. G. Sol & A. A. Verrijn-Stuart (Eds.), *Information system design methodologies: Improving the practice* (pp. 145-171). Amsterdam, the Netherlands: North-Holland.

Leavitt, H.J. (1965). Applied organizational change in industry: Structural, technological, and humanistic approaches. In J.G. March (Ed.). *Handbook of organizations* (pp. 1144-1140). Chicago: Rand McNally.

Lee, H., & Choi, B.G. (1998). A comparative study of conceptual data modeling techniques. *Journal of Database Management, 9*, 26-35.

Liao, C.C., & Palvia, P.C. (2000). The impact of data models and task complexity on end-user performance: An experimental investigation. *International Journal Of Human-Computer Studies, 52*(5), 831-845.

McFadden, F.R., Hoffer, J.A., & Prescott, M. (1999). *Modern database management*. Reading, MA: Addison-Wesley.

Navathe, S.B. (1992). Evolution of data modeling for databases. *Communications of the ACM, 35*(9), 112-123.

Nordbotten, J.C., & Crosby, M.E. (1999). The effect of graphic style on data model interpretation. *Information System Journal, 9*(2), 139-155.

Ozanne, J.L., Brucks, M., & Grewal, D. (1992). A study of information search behavior during the categorization of new products. *Journal of Consumer Research, 18*(4), 452-463.

Palvia, P.C., Liao, C.C., & To, P.-L. (1992). The impact of conceptual data models on end-user performance. *Journal of Database Management, 3*(4), 4-15.

Parsons, J. (2003). Effects of local versus global schema diagrams on verification and communication in conceptual data modeling. *Journal of Management Information Systems, 19*(3), 155-183.

Parsons, J., & Wand, Y. (1997). Choosing classes in conceptual modeling. *Communications of the ACM, 40*(6), 63-69.

Parsons, J., & Wand, Y. (2000). Emancipating instances from the tyranny of classes in information modeling. *ACM Transactions On Database Systems, 25*(2), 228-268.

Ram, S., & Ramesh, V. (1998). Collaborative conceptual schema design: A process model and prototype system. *ACM Transactions On Information Systems, 16*(4), 347-371.

Ramesh, V., & Browne, G.J. (1999). Expressing causal relationships in conceptual database schemas. *Journal of Systems and Software, 45*(3), 225-232.

Rosch, E., & Mervis, C. (1975). Family resemblances: Studies in the internal structure of categories. *Cognitive Psychology, 7*, 573-605.

Rosch, E., Mervis, C., Gray, W., Johnson, D., & Boyes-Braem, P. (1976). Basic objects in natural categories. *Cognitive Psychology, 8*, 382-439.

Salancik, G.R., & Pfeffer, J. (1978). A social information processing approach to job attitudes and task design. *Administrative Science Quarterly, 23*, 427-456.

Shoval, P., & Even-Chaime, M. (1987). Data base schema design: An experimental comparison between normalization and information analysis. *Data Base, 18*(3), 30-39.

Shoval, P., & Frumermann, I. (1994). OO and EER conceptual schemas: A comparison of user comprehension. *Journal of Database Management, 5*(4), 28-38.

Shoval, P., & Shiran, S. (1997). Entity-relationship and object-oriented data modeling - An experimental comparison of design quality. *Data & Knowledge Engineering, 21*(3), 297-315.

Siau, K. (1999). Information modeling and method engineering. *Journal of Database Management, 10*(4), 44-50.

Siau, K., Wand, Y., & Benbasat, I. (1995). A psychological study on the use of relationship concept -- Some preliminary findings. In J. Iivari, K. Lyytinen & M. Rossi (Eds.), *Lecture notes in computer science - Advanced information systems engineering, Vol. 932* (pp. 341-354). Springer-Verlag.

Siau, K., Wand, Y., & Benbasat, I. (1997). The relative importance of structural constraints and surface semantics in information modeling. *Information Systems, 22*(2-3), 155-170.

Sinha, A.P., & Vessey, I. (1999). An empirical investigation of entity-based and object-oriented data modeling: A development life cycle approach. *Proceedings of the Twentieth International Conference on Information Systems*, Charlotte, NC.

Smith, E.R., & Medlin, D.L. (1981). *Categories and concepts*. Cambridge, MA: Harvard University Press.

Srinivasan, A., & Te'eni, D. (1995). Modeling as constrained problem solving: An empirical study of the data modeling process. *Management Science, 41*(3), 419-434.

Teorey, T.J., Yang, D., & Fry, J.P. (1986). A logical design methodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys, 18*(2), 197-222.

Topi, H., & Ramesh, V. (2002). Human factors research on data modeling: A review of prior research, an extended framework and future research directions. *Journal of Database Management, 13*(2), 3-19.

Wand, Y., Monarchi, D.E., Parsons, J., & Woo, C.C. (1995). Theoretical foundations for conceptual modeling in information systems development. *Decision Support Systems, 15*, 285-304.

Wand, Y., & Weber, R. (1993). On the ontological expressiveness of information systems analysis and design grammars. *Journal of Information Systems, 3*, 217-237.

Wand, Y., & Weber, R. (1995). On the deep structure of information systems. *Information Systems Journal, 5*, 203-223.

Wand, Y., & Weber, R. (2002). Research commentary: Information systems and conceptual modeling - A research agenda. *Information Systems Research, 13*(4), 363-376.

Wand, Y., Storey, V., & Weber, R. (1999). An ontological analysis of the relationship construct in conceptual modeling. *ACM Transactions On Database Systems, 24*(4), 494-528.

Weber, R. (1996). Are attributes entities? A study of database designers' memory structures. *Information Systems Research, 7*(2), 137-162.

Weber, R. (1997). *Ontological foundations of information systems*. Melbourne, Australia: Coopers & Lybrand.

Weick, K.E. (1979). Cognitive processes in organizations. *Research in Organizational Behavior, 1*, 41-47.

APPENDIX A: EMPIRICAL HUMAN FACTORS RESEARCH ON CONCEPTUAL DATA MODELING

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Brosey & Shneiderman	1978	a) Data modeling formalism (Hierarchical vs. relational); b) Programming experience	Model comprehension	1) Comprehension based on the subjects' ability to answer questions. 2) Recall of a database schema from memory.	Hierarchical schemas appeared to be easier to recall than the relational schema. The hierarchical schema was easier to understand (particularly for beginning users) than the relational schema.
Hoffer	1982	a) Programming experience; b) Cognitive style; c) Task focus	Data model characteristics	Development of data models based on four different case descriptions: 1) order processing/inquiry; 2) customer and salesperson tracking; 3) product sales and inventory tracking; and 4) task not specified.	Data flow models were by far the most common way of modeling data. Decreased situation focus leads to decreasing confidence.
Juhn & Naumann	1985	Data modeling formalism (ER vs. relational (RDM) vs. LDS vs. DAD)	a) Three aspects of comprehension and b) Ability to identify database elements needed for a search	To answer seventeen questions based on different representations of the same data model. Models not given.	The subjects using semantic models (ER, LDS) identified relationships and their cardinalities better than those using relational models (RDM, DAD).

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Shoval & Even-Chaime	1987	a) Data modeling formalism (Normalization vs. Information Analysis (NIAM)); b) Task complexity (Simple vs. complex)	a) Modeling performance measured with model quality; b) Time required to finish the modeling task; and c) Preference over a modeling formalism	To develop two logical data models (one with NIAM, the other with normalization) based on DFDs and related narratives. No specific info on cases.	The subjects performed better in both simple and complex tasks and needed less time when using normalization; in addition, the subjects preferred normalization over NIAM.
Jarvenpaa & Machesky	1989	Data modeling formalism (EER vs. relational)	a) Data model accuracy, b) Data modeling time, c) Understanding of notation, and d) Top-down vs. bottom-up approach	To develop a logical (according to the authors) data model based on a narrative. Only one of the four cases included.	EER modelers learned faster, perceived the task to be easier, understood the notation better, and used the top-down approach more often.
Batra, Hoffer, & Bostrom	1990	Data modeling formalism (EER vs. relational)	a) Performance measured with modeling correctness for nine different facets; b) Perceived ease-of-use	To develop a representation based on a narrative using either EER or relational method. Case: Projects, Inc., adapted from Teorey et al., 1986.	Modeling performance of EER users was better than that of relational model users with binary 1:M, binary N:M, and ternary 1:M:M relationships.
Palvia, Liao, & To	1992	Data modeling formalism (Data Structure Diagrams (DSD) vs. ER vs. OO/Kroenke)	a) Performance measured with comprehension of the meaning of the database (authors' terminology); b) Time	To answer questions based on a "version of the database" corresponding to the database model.	Comprehension performance was significantly better and comprehension time faster with an OO database than with the other two.

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Batra & Davis	1992	Experience (Novice vs. expert)	Protocol analysis; no clearly defined DV.	To prepare an ER model based on a narrative and verbalize their thoughts when preparing the model. Case: Far-Eastern Repair Center.	No clear quantitative results; support obtained for five findings from prior research regarding the difference between novices and experts (different process models, experts have richer vocabulary and better ability to categorize; experts' ability to automate; novices' tendency to make mistakes).
Amer	1993	Data modeling formalism (ER vs. relational)	Performance measured with the number of errors in an error identification task	To identify errors in a "conceptual database" (author's description) model based on a narrative description of an accounting transaction processing cycle.	The performance of the users of the ER model was significantly better in identifying all binary 1:M, binary M:N, unary, and class-subclass relationships (not ternary).

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Batra & Kirs	1993	Data modeling formalism (EER vs. Data aggregation)	Performance measured with the correctness of a relational representation of the conceptual data models	To develop a conceptual representation (either EER or DA) based on a narrative and converting that into a logical (relational) design. Case: Adapted from Teorey et al. (1986).	There was no difference between the relational representations (they were equally poor). EER-representations were, in general, good, and the degradation of quality from EER to relational was clear. With DA users, already the quality of the DA models seemed to be poor (this was not formally evaluated).
Bock & Ryan	1993	Data modeling formalism (EER vs. OO/Kroenke)	Modeling correctness measured for eight different facets	To develop a conceptual representation (either EER or OO - Kroenke) based on a narrative. Case: Batra et al. (1990).	EER modelers performed significantly better in three of the eight facets: identifier attribute, unary 1:1 and binary M:N.
Batra & Sein	1994	No IV: examination of response to feedback	Reduction in the number of errors after feedback was provided	To develop a relational representation based on a narrative (intermediate conceptual representation was ok but not required). Case: Golden Panther Rare Animal Zoo.	Through feedback, the subjects were able to correct 19 of initial 53 errors. Twelve of these were in the categories of ternary and unary association relationships.

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Batra & Antony	1994	Data modeling formalism (ER vs. relational)	Performance (measured with the correctness of the solution) in a user view modeling task	To convert user views (forms and reports) supplemented with a narrative into either an ER or a relational model. Case: Best Furniture Company (McFadden, Hoffer & Srinivasan).	The overall performance of ER modelers was better. Increased number of nesting levels had a negative impact on performance.
Shoval & Frumermann	1994	Data modeling formalism (EER vs OO (generic))	User comprehension measured with a true-false instrument (5 facets)	To answer a set of 48 True-False statements based on a conceptual data model. Case adapted from Batra et al. (1990).	Subjects using EER were able to interpret ternary relationships more correctly. OO users were better with a vague “other facts” category.
Hardgrave & Dalal	1995	a) Data modeling formalism (EER vs OMT); b) Task complexity (Simple vs. complex)	a) Ability to understand a conceptual data model measured with a multiple choice instrument (one item per facet); b) Time to understand; c) Perceived ease-of-use	To answer 5 (simple condition) or 10 (complex condition) multiple-choice questions based on a conceptual data model. No information given about the case.	Subjects evaluating OMT models were significantly (10-20%) faster than subjects evaluating EER models.

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Kim & March	1995	Data modeling formalism (EER vs. NIAM)	Semantic (entities, attributes, relationships, dependencies, and identifiers) and syntactic task performance, Perceived difficulty of formalism, Perceived value of formalism.	User experiment: To answer a list of questions based on the conceptual model and to identify discrepancies between a narrative and a conceptual model; Analyst experiment: To develop a conceptual data model based on a narrative. Case: Two operations management cases: YBCL and Air King.	EER users performed significantly better in all aspects of semantic correctness, and perceived EER to be less difficult and more valuable than NIAM.
Siau, Wand, & Benbasat	1995	Domain familiarity	Interpretation regarding the nature of a relationship (optional vs. mandatory); confidence	To evaluate a set of relationships and determine whether they are optional or mandatory	Subjects chose overwhelmingly the interpretation that the relationships are optional regardless of domain. Subjects' confidence level was significantly higher with a familiar domain.
Srinivasan & Te'eni	1995	No manipulation; an analysis of heuristics used by the modeler.	Modeling correctness (but main focus was on process/heuristic analysis).	To develop a logical data model based on a narrative and perform queries against it (task description vague).	A set of findings regarding effective heuristics and transitions between them.

APPENDIX A (CONTINUED)

Authors	**Year**	**Independent Variable(s)**	**Dependent Variable(s)**	**Research Task(s)**	**Results**
Agarwal, Sinha, & Tanniru	1996	a) Prior experience (whether or not the user has process-oriented design experience); b) Task type (process-oriented vs. object-oriented).	Performance (measured with the number of correct elements) in modeling structure, behavior, and a combination of the two above	To develop object-oriented models based on narrative descriptions of business problems (accounts payable system (process-oriented) and employee benefits system (object-oriented)).	Subjects with experience in process-oriented modeling performed better in modeling behavior but not in modeling structure.
Weber	1996	a) Experience (NIAM expert vs. novice); b) Task complexity (Simple vs. complex)	Ability to recall items in a NIAM diagram; recall sequence	To draw NIAM diagrams from memory in a five-trial sequence.	The results suggest the existence of entities as separate structures from attributes as a mechanism to support long-term recall.
Shoval & Shiran	1997	Data modeling formalism (EER vs. OO(O2/ODE))	a) Modeling performance measured with model quality (9 facets); b) Time required to finish the modeling task; and c) Preference over a modeling formalism	To develop two conceptual data models (one with EER, the other with OO) based on a narrative. Cases: Extended Batra et al. (1990) and a hospital case.	The subjects performed better in modeling unary 1:1 relationships and ternary relationships when using EER; the subjects needed less time with EER; the subjects preferred EER over OO.
Siau, Wand, & Benbasat	1997	The existence of a conflict between structural constraints and surface semantics in ER diagrams	Interpretation regarding the nature of the relationship (optional/mandatory), confidence, and perceived familiarity with domain	To evaluate a set of relationships and determine whether they are optional or mandatory and to indicate domain familiarity	Subjects structural constraints instead of surface semantics. Conflicting structural constraints increased confidence and perceived domain familiarity

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Lee & Choi	1998	a) Data modeling formalism (EER vs. SOM vs. ORM vs. OMT), b) Experience (Training in ER), and c) Task type (language vs. forms)	a) Modeling performance measured with correctness of eight different facets; b) Time required to finish the modeling time; and c) Perceived ease-of-use	To develop a conceptual data model based on either a narrative or a set of forms.	Novice subjects performed better with EER and OMT than with SOM and ORM; Experts performed faster and perceived higher ease-of-use with EER and OMT than with SOM and ORM.
Ramesh & Browne	1999	Database-knowledgeable vs. Database-novice (knowledge of ER)	Ability to model/identify causal relationships	To sketch a graphical description of a case situation based on a narrative (no formalism specified). Case: Mountain View Community Hospital.	Database-novice subjects identified causal relationships more frequently than database-knowledgeable.
Norbotten & Crosby	1999	Data modeling formalism (IDEF1X vs. SSM vs. NIAM vs. OODM (Cattell))	a) Model comprehension measured by an ability to recognize a modeling construct based on a model; b) Attention paid to constructs; c) Reading strategies	To identify modeling constructs in a data model presented to the user on a computer screen.	Familiar notation improves comprehension. Identifying relationships is more difficult with highly graphical models (specifically NIAM).
Burton-Jones & Weber	1999	a) Ontological clarity (relationships with vs. without attributes), and b) Domain knowledge (high vs. low)	a) Problem-solving performance based on the model; b) Ability to understand the model	To answer comprehension questions and to solve problems based on the data model.	In unfamiliar domains, using relationships with attributes affects performance harmfully.

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Sinha & Vessey	1999	a) Conceptual (EER and OOD/Coad & Yourdon) vs. logical (RDM and OOT); b) EER vs. OOD; c) RDM vs. OOT; d) EER --> RDM vs. OOD --> OOT	Modeling performance measured with accuracy of modeling entities/classes and attributes, association relationships, and generalization relationships.	To develop four conceptual data models (RDM, EER, OOT, OOD) based on a case representing a university.	The subjects performed better with conceptual than logical data modeling with binary 1:M and M:N relationships; OOD was better than EER for representing entities/classes and attributes; OOT was better than RDM for representing generalization; OOD --> OOT was better for conceptual --> logical mapping than EER --> RDM.
Liao & Palvia	2000	a) Data modeling formalism (Relational vs. EERM vs. OOM (Kim)); b) Task complexity	a) Modeling performance measured with model quality (8 facets); b) Time	a) To develop a data model based on a narrative; b) to convert a conceptual data model to a logical data model.	The subject using relational and OOM performed better than those using EERM in modeling unary one-to-one relationships; EERM users performed better than relational users for modeling binary one-to-many and many-to-many relationships. EERM users required more time than relational and OOM users.

APPENDIX A (CONTINUED)

Authors	Year	Independent Variable(s)	Dependent Variable(s)	Research Task(s)	Results
Batra & Antony	2001	a) Consulting system vs. no consulting system; b) Knowledge level; c) Task type (relationship types)	Modeling performance measured with model quality	To develop a conceptual data model based on a narrative.	The consulting system improved performance of the low knowledge-level subjects.
Bodart, Patel, Sim, and Weber	2001	E1: a) Type of representation (optional properties vs. subtyping); b) Domain complexity; c) Trial E2: a) Type of representation; b) Trial E3: a) Type of representation	E1: Recall accuracy; E2: Accuracy and time E3: Problem-solving performance	E1: Free recall test (review diagram and draw it from memory); E2: Comprehension test (yes/no questions); E3: Problem-solving test with open-ended questions based on an ER-model.	Optional attributes and relationships should be used only when the goal is to obtain surface-level understanding of the domain and should not be used when a deep-level understanding is needed.
Parsons	2003	Form of representation: a single global schema vs. two local schemas	Ability to interpret information from a conceptual schema correctly	Answer questions based on a schema/schemas	Local schemas better for local verification; Global schema better if necessary to understand a complex classification structure with complementary views; Local schemas better if conflict between schemas

SECTION III:

DATABASE DESIGN AND DEVELOPMENT: APPLICATIONS

Chapter XI

Using DEMO and ORM in Concert:

A Case Study

Jan L.G. Dietz, Delft University of Technology, The Netherlands

Terry Halpin, Northface University, USA

ABSTRACT

The Demo Engineering Methodology for Organizations (DEMO) enables business processes of organizations to be modeled at a conceptual level, independent of how the processes are implemented. DEMO focuses on the communication acts that take place between human actors in the organization. The Object-Role Modeling (ORM) approach enables business information to be modeled conceptually, in terms of fact types as well as the business rules that constrain how the fact types may be populated for any given state of the information system and how derived facts may be inferred from other facts. ORM also includes procedures to map conceptual data models to physical database schemas. Both DEMO and ORM treat fact types as fundamental, and require that their models be expressible in natural language sentences. This suggests that the approaches may be synthesized in a natural way, resulting in a more powerful method for business modeling. This chapter discusses an exploratory case study in which both methods were used in concert, and identifies some lessons learned.

INTRODUCTION

Demo Engineering Methodology for Organizations (DEMO) is a method for organization engineering, an emerging discipline concerning the design and implementation of organizations (Dietz, 1994, 1999, 2003a, 2003b; Van Reijswoud, Mulder & Dietz, 1999). Traditional organization science is based on a teleological system definition, which is concerned with the function and the behavior of a system in its environment. The corresponding dominant paradigm for studying organizations is the IPO-paradigm (Input-Process-Output). The matching model type is the black-box-model. *Organization engineering* is based on an *ontological* system definition, which is concerned with the construction and operation of a system. Its dominant paradigm for studying organizations is the *PSI-paradigm* (Performance in Social Interaction). The matching model type is the *white-box-model*.

Organization science and organization engineering are complementary fields. The former is particularly useful for *managing* organizations (strategic, tactic and operational management), while the latter is especially useful for *changing* organizations (redesign/re-engineering of business processes, forming networks of organizations, etc.).

The PSI-paradigm states that an organization consists of people who, while communicating, enter into and comply with commitments (social interaction) about the things they bring about in reality (performance). This reality therefore is to a large extent an inter-subjective reality. Put differently, in their social interaction people engage in obligations about actions to take, and reach agreement about the results of those actions. The PSI-paradigm is made more specific and operational in DEMO as described later. DEMO belongs to a group of modeling approaches that are all based on the Language/Action Perspective (e.g., Goldkuhl, 1996; Medina-Mora, Winograd, Flores & Flores, 1992). Van Reijswoud and Dietz (1999) provide a detailed description of DEMO.

Object-Role Modeling (ORM) is a fact-oriented approach for modeling information at a conceptual level. An overview of ORM is given in Halpin (1998a), and a detailed treatment in Halpin (2001a). ORM includes a family of closely related variants, including Natural Information Analysis Method (NIAM) (Wintraecken, 1990), Natural Object Relationship Method (NORM) (De Troyer & Meersman, 1995), Predicator Set Model (PSM) (ter Hofstede, Proper & van der Weide, 1993), and Fully Communication Oriented Information Modeling (FCO-IM) (Bakema, Zwart & van der Lek, 1994). Unlike Entity-Relationship (ER) modeling (Chen, 1976) and the class diagram technique of the Unified Modeling Language (UML) (OMG UML RTF, 2003), ORM makes no use of attributes as a base construct, instead expressing all fact types as relationships. This attribute-free approach leads to greater semantic stability in conceptual models and conceptual queries (Bloesch & Halpin, 1997; Halpin, 1998b) and enables ORM fact structures to be directly verbalized and populated using natural language sentences.

ORM supports mixfix predicates of any arity (unary, binary, ternary, etc.), so its constraints and derivation rules can also be directly verbalized in sentential form. For details on business fact and rule verbalization in ORM, see the series of articles initiated by Halpin (2003). Moreover, ORM's graphic constraint notation is far more expressive than that of UML class diagrams or industrial ER versions. ORM is now supported by a number of modeling tools, which can automatically transform ORM schemas into physical database schemas (e.g., see Halpin, Evans, Hallock & MacLean, 2003). For such reasons, ORM is being increasingly used for conceptual analysis of information, as well as ontology specification (Spyns, Meersman & Jarrar, 2002), and is currently being considered as a candidate for a standard business rule modeling language within the Object Management Group.

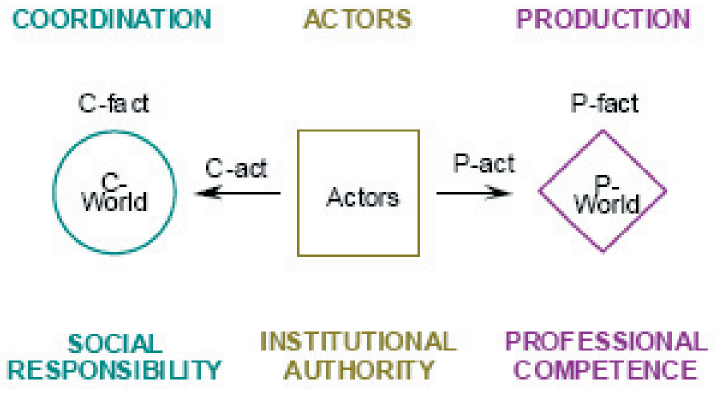
Both DEMO and ORM treat fact types as fundamental, and require that their models be expressible in natural language sentences. This suggests that the approaches may be synthesized in a natural way, resulting in a more powerful method for business modeling. This chapter discusses the first attempts to explore the feasibility of this synthesis, and identifies some lessons learned, using a running example of a library application to illustrate the main ideas.

The following section summarizes the essential concepts and model types underlying the DEMO approach, and discusses how the library application is modeled using DEMO. Next, the chapter explains the main concepts and notations of ORM, and shows how the library application may be modeled in ORM. Then, the chapter identifies some ways in which ORM supplements DEMO by providing additional constructs and techniques for modeling the information. The result of successfully performing a P-act is a *production fact* or P-fact. P-facts in our library example include "membership M has started to exist" and "the late return fine for loan L is paid". The variables M and L denote an instance of membership and loan, respectively. All realization issues are fully abstracted out. Only the facts as such are relevant, not how they are achieved. Examples of C-acts are requesting and promising a P-fact (e.g., requesting to become a member of the library).

The result of successfully performing a C-act is a *coordination fact* or C-fact (e.g., being requested of the production fact "membership #387 has started to exist"). Again, all realization issues are ignored (e.g., whether the request is made by a letter or e-mail or via a web site). Just as we distinguish between P-acts and C-acts, we also distinguish the two worlds in which these kinds of acts have effect: the *production world* or P-world and the *coordination world* or C-world, respectively. Both the P-world and the C-world are at any moment in a particular state. A state is simply defined as a set of facts. So, a state of the P-world is a set of P-facts and a state of the C-world is a set of C-facts. State changes, also called *transitions*, take place instantaneously. The occurrence of a transition at a particular point in time is called an *event*. An example of an event is the creation of the P-fact "membership #387 has started to exist". Events occur at discrete points in time, and the number of events in any finite time interval is finite.

P-acts and their related C-acts appear to occur in generic recurrent patterns, called *transactions*. A transaction has three phases: the order phase, the execution phase, and the result phase. It is carried out by two actors, who alternately perform acts. The actor who starts the transaction and eventually completes it is called the *initiator*. The other, who actually performs the production act, is called the *executor*. The order phase is a conversation that starts with a request by the initiator and ends (if successful) with a promise by the executor. The result phase is a conversation that starts with a statement by the executor and ends (if successful) with an acceptance by the initiator. In between these two conversations, there is the execution phase, in which the executor performs the P-act. The process of a transaction can be more complicated but its complexity is always limited (Dietz, 2003b).

Figure 1: The white-box model of an organization



Transactions are the molecules of business processes (Dietz, 2003a), the C-acts and P-acts being the atoms. A *business process* is defined as a (arbitrarily large) structure of causally linked transactions. A transaction T02 is causally linked to a transaction T01 if and only if T02 is initiated during the course of T01 by either the initiator or the executor of T01. Usually, T01 has to wait for the completion of T02 before proceeding.

Concerning production acts, and hence actors, three levels of abstraction are distinguished (see Figure 2). These levels may be understood as 'glasses' for viewing an organization. Looking through the es*sential* glasses, one observes the core business actors, who perform production acts that result in original (non-derivable) facts, and who directly contribute to the organization's function (e.g., approving a membership application, or diagnosing a patient's medical problems). These essential acts and facts are collectively called *B-things* (from Business). Looking through the *informational* glasses, one observes intellectual actors, who execute informational acts like collecting, providing, recalling and computing knowledge about business acts and their results. Informational acts and facts are collectively called *I-things* (from Information and Intellect). Looking through the *documental* glasses, one observes documental actors, who execute documental acts like gathering, distributing, storing, copying, and destroying documents containing the aforementioned knowledge. Documental acts and facts are collectively called *D-things* (from Documents and Data).

The three kinds of actors are called B-actors, I-actors and D-actors. They are elements of three corresponding aspect systems of an organization: the B-system, the I-system, and the D-system. The starting point and emphasis in DEMO is the B-system. Only in the B-system may new original facts be created to contribute to fulfilling the organization's mission. The corresponding I-system and D-system are part of the realization of the B-system, and so can be designed only after the B-system is designed. Information and communication technology can be applied without any risk or harm to the I-system and the D-system. However, one must be cautious in applying it to the B-system, to prevent machines from taking over the responsibility of B-actors. One can only mimic or simulate B-systems. The triangular shape of the levels in Figure 3 shows that there is nothing 'above' the B-system, and that generally the amount of D-things in an organization is much more than the amount of I-things, and that the amount of I-things is much more than the amount of B-things.

Figure 2: The three levels of abstraction

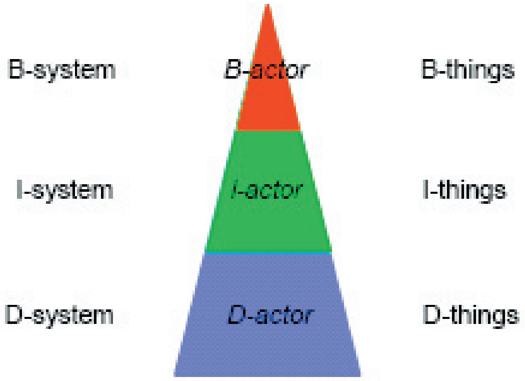
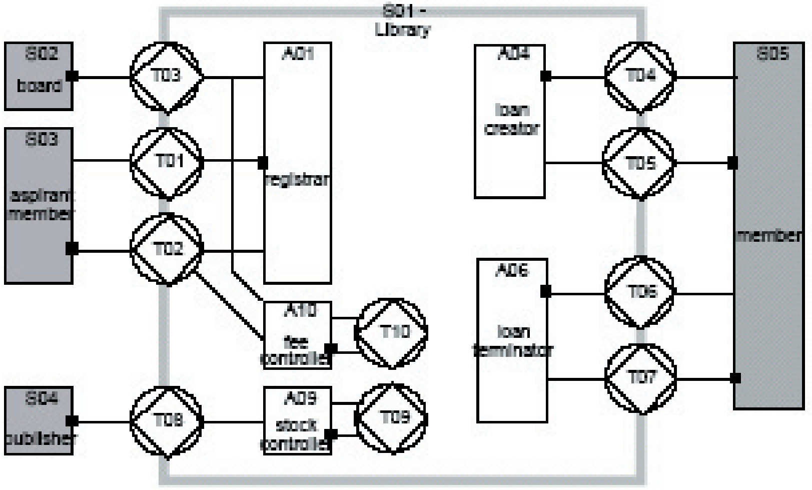


Figure 3: DEMO Construction Model (CM) of the library



transaction type		resulting P-event type	
T01	membership_registration	PE01	*membership M has started to exist*
T02	membership_fee_payment	PE02	*the fee for membership M in year Y has been paid*
T03	reduced_fee_approval	PE03	*the reduced fee for membership M in year Y has been approved*
T04	loan_start	PE04	*loan L has started to exist*
T05	book_return	PE05	*book copy C has been returned*
T06	loan_end	PE06	*loan L has ended to exist*
T07	return_fine_payment	PE07	*the late return fine for loan L has been paid*
T08	book_shipment	PE08	*shipment S has been performed*
T09	stock_control	PE09	*the stock_control for period P has been done*
T10	annual_fee_control	PE10	*the annual_fee_control for year Y has been done*

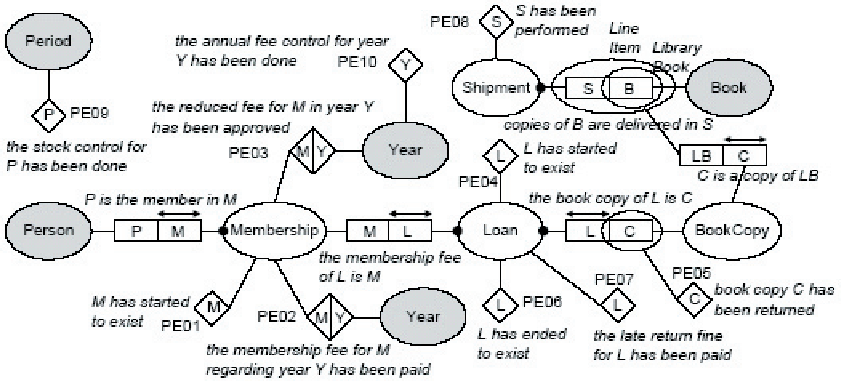
The complete model of the B-system of an organization in DEMO is called the *essential model* of the organization. It consists of an integrated set of four aspect models: the *Construction Model* (CM), the *Process Model* (PM), the *State Model* (SM), and the *Action Model* (AM). The CM shows the actor roles and the transaction types in which they play (as initiator and/or executor). The AM specifies the action rules that the actors apply in carrying out their transactions. Based on the AM, the PM shows how the transaction types are causally and conditionally related, and the SM models the fact types that are created and/or used in carrying out the transactions. Only the CM and the SM are elaborated in this chapter.

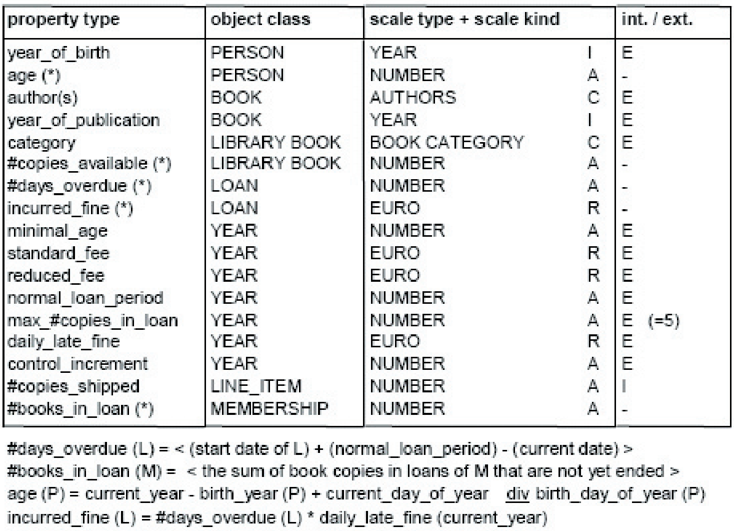
Figure 3 shows the CM of the library case. The diagram (an Actor Transaction Diagram) shows the actor roles, transaction types, and the relationships between them (i.e., which actor roles are initiator and/or executor of which transaction types). An actor role is represented by a box; the transaction symbol is a diamond (production) in a disk (coordination). The small black box denotes which actor role is the executor of a transaction type. The boundary of the considered part of the library is represented by the gray-lined open box. Actor roles inside the boundary are elementary actor roles—they execute exactly one transaction type. Actor roles outside the boundary are (by definition) non-elementary, so-called system actor roles; they are colored gray. Actually, what is inside the boundary is the ‘uncovering’ of the system actor role S01 (Library).

The table below the diagram (called a Transaction Result Table) lists all transaction types and specifies for each the resulting P-event type. Actor roles A09 and A10 are self-activating actors: they are both initiator and executor of the same transaction. This is how DEMO models periodic activities.

Figure 4 shows the SM corresponding to the CM of Figure 4. The diagram (an Object Fact Diagram), plus the table below it (an Object Property Table) may be viewed as a variant of the ORM model discussed in this chapter. They specify all object types and fact types occurring in the action rules of the AM (of the B-system). The SM of (a part of) an organization is an *ontological* conceptual schema—it describes the types of things and facts (relationships) that can be observed, as well as the laws that appear to hold for the co-

Figure 4: DEMO State Model (SM) of the library





property type	object class	scale type + scale kind		int. / ext.
year_of_birth	PERSON	YEAR	I	E
age (*)	PERSON	NUMBER	A	-
author(s)	BOOK	AUTHORS	C	E
year_of_publication	BOOK	YEAR	I	E
category	LIBRARY BOOK	BOOK CATEGORY	C	E
#copies_available (*)	LIBRARY BOOK	NUMBER	A	-
#days_overdue (*)	LOAN	NUMBER	A	-
incurred_fine (*)	LOAN	EURO	R	-
minimal_age	YEAR	NUMBER	A	E
standard_fee	YEAR	EURO	R	E
reduced_fee	YEAR	EURO	R	E
normal_loan_period	YEAR	NUMBER	A	E
max_#copies_in_loan	YEAR	NUMBER	A	E (=5)
daily_late_fine	YEAR	EURO	R	E
control_increment	YEAR	NUMBER	A	E
#copies_shipped	LINE_ITEM	NUMBER	A	I
#books_in_loan (*)	MEMBERSHIP	NUMBER	A	-

#days_overdue (L) = < (start date of L) + (normal_loan_period) - (current date) >
#books_in_loan (M) = < the sum of book copies in loans of M that are not yet ended >
age (P) = current_year - birth_year (P) + current_day_of_year div birth_day_of_year (P)
incurred_fine (L) = #days_overdue (L) * daily_late_fine (current_year)

existence of these things and facts. The gray-colored boxes depict external object classes. They contain objects that play a role in the business processes, but their existence is determined by transactions other than those in the CM. The white-colored boxes depict internal object classes. The objects in these classes are created in the mentioned transactions. For the classes Membership, Loan, and Shipment, this is obvious. For BookCopy, these are the books delivered in shipments to the library.

The diamond shaped fact types are the production fact types that also appear in the Transaction Result Table of Figure 3. These fact types link the conceptual schema of the production world to the transactions that change the state of the production world. Consider the creation and termination of loans. There are two 'normal' fact types: "the membership of L is M" and, "the book copy of L is C". A uniqueness constraint holds for the role of the loan in both fact types: a loan always relates to at most one membership and one book copy. A mandatory constraint also holds for Loan in both fact types. Hence a loan always relates to exactly one membership and one book copy. Therefore, the fact types "the membership of L is M" and "the book copy of L is C" are *existentially dependent* on Loan.

A new loan can be conceived of (and in a simulation game be generated), but that doesn't mean that it actually exists yet. In order to come into being, an event of type PE04 is needed. This event has a time stamp (the point in time at which it occurs). By definition this is the point in time at which the transaction T04 concerning L has successfully been completed (Dietz, 2003a). The loan ends its existence by an event of type PE06. During the lifetime of the loan, an event of type PE07 may occur (late return fine payment).

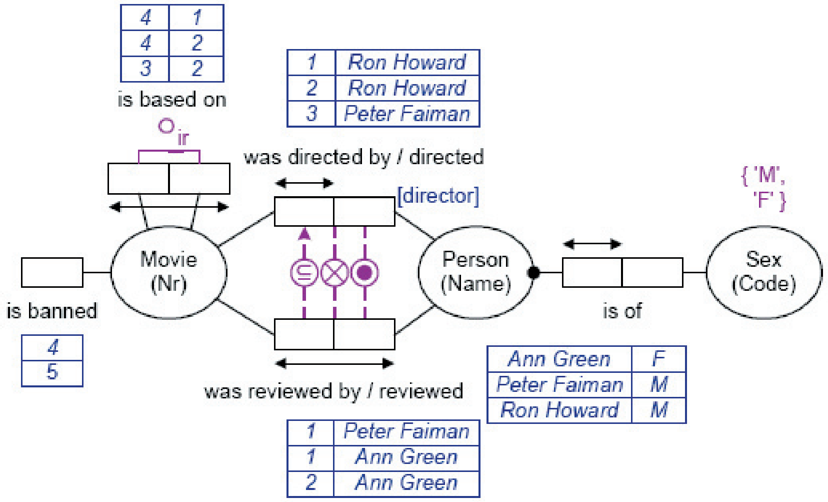
ORM

This section briefly explains the basic ORM graphical symbols, and then provides an ORM model for the library application. Object-Role Modeling is so-called because it views the universe of discourse (application domain) as a set of *objects* (non-lexical entities or lexical values) that play *roles* (parts in relationships). ORM stores all data in simple *fact types*, catering for unary, binary, and longer relationships, and allowing all fact structures to be easily populated with sample data to help validate business rules. Unlike ER and UML, ORM makes no use of attributes.

Graphically, object types are depicted as named ellipses (solid for entity types, and dotted for value types). As in logic, a *predicate* is a proposition with object-holes in it. In ORM, a predicate is treated as an ordered set of one or more roles, each of which is depicted as a box, which may optionally be named. A fact type is formed by applying a predicate to the object types that play its roles. Fact types in ORM must be given one or more readings. The *arity* of a predicate is its number of roles. For discussion purposes, each fact type may be populated by entries in a sample fact table that includes one column for each role of the fact type.

The ORM model in Figure 5 includes three object types (Movie, Person and Sex) and five fact types: Movie is banned; Movie is based on Movie; Movie was directed by Person; Movie was reviewed by Person; Person is of Sex. Inverse readings are supplied for two associations: Person directed Movie; Person reviewed Movie. One role is named ("director"). Simple identification schemes may be abbreviated in parentheses. For example, Movie(Nr) abbreviates the injective (1:1 into) association Movie has MovieNr. For simplicity, we assume that persons in this domain may be identified by name. In this

Figure 5: An ORM model including an ORM schema and sample fact populations



example, all fact types are unary or binary. We could add Movie was released in Country in Year as a ternary fact type.

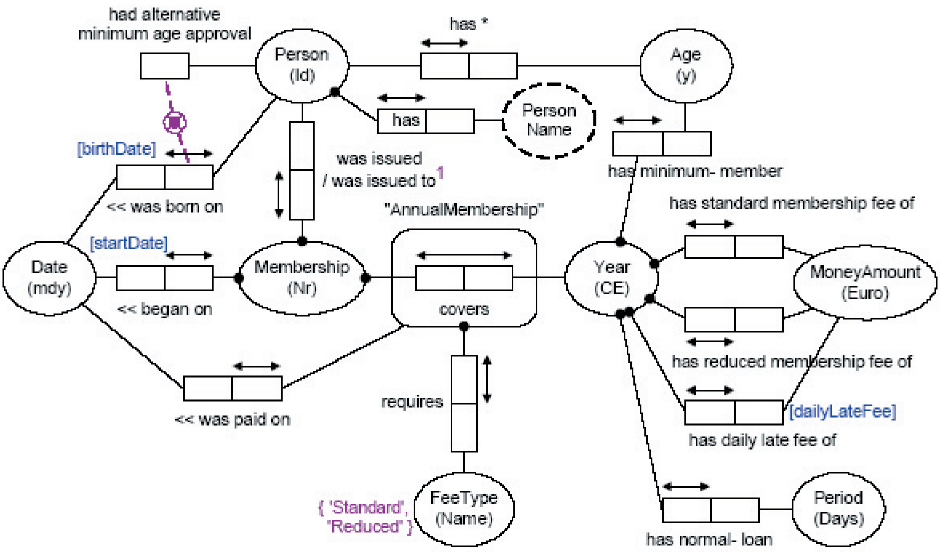
ORM classifies business rules into constraints and derivation rules. The ORM model in Figure 5 includes constraints but no derivations. The *value constraint* {'M', 'F'} indicates the possible sex codes. Arrow-tipped lines across one or more roles denote *uniqueness constraints*, indicating that instantiations of that role sequence must be unique. For example, the uniqueness constraint on the first role of Person is of Sex indicates that entries in the fact column for that role must be unique. The English version of ORM's formal textual language verbalizes this constraint as: **each** Person is of **at most one** Sex.

A solid dot (possibly circled) connected to a set of one or more roles denotes a *mandatory constraint* over that role set. For example, the mandatory dot connected to the first role of Person is of Sex indicates that each Person is of **some** Sex. The mandatory dot connected to the other two roles played by Person depicts an *inclusive-or constraint*: each Person directed **some** Movie **or** reviewed **some** Movie (possibly both).

The ᴼir symbol connected to the roles of the fact type Movie is based on Movie denotes the irreflexive *ring constraint*: **no** Movie is based on **itself**. The circled subset symbol "⊆" connected by an arrow from the first role of Movie was reviewed by Person to the first role of Movie was directed by Person denotes a *subset constraint*, indicating that the population of the first role must always be a subset of the population of the second role. In English: each Movie **that** was reviewed by **some** Persons **also** was directed by **some** Person.

A subset constraint is one kind of set-comparison constraint. In general a set-comparison constraint applies across sequences of compatible role sequences (of one or more roles). Other varieties of set-comparison constraints are exclusion and equality constraints. For example, the circled "X" in Figure 5 denotes an *eXclusion constraint* between the role-pairs that comprise the direction and review predicates. In English: **no** Movie was directed by **and** reviewed by **the same** Person.

Figure 6: ORM subschema for library membership



We now turn to the library application. For convenience, we divide the ORM schema into four subject areas: membership, loan, book, and book shipment. Figure 7 shows the main aspects of the membership subschema. The reference mode "Id" for Person indicates that each person has a *value-based identifier*, called PersonId, used in human communication. Each person also has a name, not necessarily unique. A library year is a calendar year, at some time during which the library was in operation. The *reference mode* "CE" denotes "Common Era", indicating calendar years are based on the Gregorian calendar.

The association Membership covers Year is *objectified* as the entity type AnnualMembership. Its association with FeeType indicates whether or not a given member has been granted a reduced membership fee for a given year. If desired, a derived fact type may be added to infer the fee paid for a given annual membership, based on the fee type and the membership fee of that type for the given year. For simplicity we assume that a member pays the full annual fee regardless of when he/she began or renewed the annual membership. In practice, it would be more commom to apply a pro-rated fee or extend the membership to a year after the date paid.

By default, predicates are read left to right and top to bottom. A reversed reading direction is indicated by a back arrow "<<". The first role of the fact type Person was born on Date is optional. This means it is optional whether we record a person's birth date (even though in the real world each person has a birth date). An ORM model reflects the *universe of discourse* (i.e., those aspects of the application world that we wish to discuss, and the rules that we wish to enforce), so the model need not agree in every respect with the real world. In this aspect, ORM differs from DEMO, where birth date is mandatory simply because each person in the real world has a birth date.

The life-buoy symbol (combination of inclusive-or and exclusion symbols) denotes an *exclusive-or constraint*: **each** Person was born on **a** Date **or** had alternative minimum age approval, **but not both**. Here the unary fact type caters for the case where a person does not supply his/her birth date, (e.g., he/she may not wish to divulge it, or might not know it) but can have the minimum age requirement approved by authorized library staff (e.g., visual inspection of a person who is obviously old).

Notice the use of *hyphens* in the fact types Year has minimum-member Age and Year has normal-loan Period. This causes the hyphenated and any subsequent words before the following term for the object type to be bound to that term for verbalization purposes. For example, the uniqueness constraint on the first of these fact types verbalizes as, “**each** Year has **at most one** minimum member Age” instead of, “**each** Yeas has minimum member **at most one** Age”.

As discussed later, *role names* displayed in square brackets are used to provide function names for derivation rules that make use of attribute-style notation. The second role of predicates with the reading “has” is assumed to have the name of the second object type, with the first letter in lower-case, unless an explicit role name overrides this. For example, the second role of Person has PersonName is named “personName”. For binary predicates with a reading comprised of “has” followed by a hyphenated phrase, the second role has a default name obtained by prepending the hyphenated phrase to the right-hand object type term. For example, the second role of the fact type Year has normal-loan Period is “normalLoanPeriod”.

The superscript “^{1}” on the fact type Membership was issued to Person indicates the existence of a *textual constraint* on this fact type. The asterisk “*” on the fact type Person has Age indicates that this fact type is *derived*. In a complete ORM model, all constraints that cannot be expressed in graphical notation as well as all *derivation rules* (to indicate how derived fact types are derived from other fact types) should be specified in a formal, textual language. For example, the derivation rule for Person has Age may be specified in attribute-style as shown below. Here, dayOfYearNr denotes the sequential position of the day in its year (e.g., 2003 September 14 has dayOfYearNr 257).

Person.age = today.year – Person.birthdate.year if today.dayOfYearNr >= Person.birthdate.dayOfYearNr else = today.year – Person.birthdate.year + 1

This formulation makes use of various operations (e.g., date subtraction) and functions (e.g., year) that are predefined for Date. Figure 8 summarizes some of the main underlying semantics from an ORM perspective. Each circled “u” depicts an *external uniqueness constraint*, indicating that each Year, DayOfYearNr combination and each Year, MonthNr, DayNr combination refers to only one Date. While the mdy (month-day-year) format for dates is used for communication purposes, internally dates may be implemented otherwise (e.g., as Julian dates). Fundamentally, ORM uses relational-style, over which an attribute-style may be defined. The nullary function “today” is defined as the result of the query !Date is today (using “!” to prepend each desired projection). The role names “dayOfYearNr”, “year”, “monthNr”, “dayNr” on the right-hand roles of the derived predicates may be used as function names in attribute-style rules.

As a small extension to the current age rule shown earlier, a derivation rule may also be specified for the derived fact type Person on Date had Age. Using this fact type, the function “age of … on …” may now be specified over the parameter list (Person, Date). The textual constraint indicated by the subscript “1” in Figure 7 may now be specified as follows:

Figure 7: Some predefined semantics underlying Date

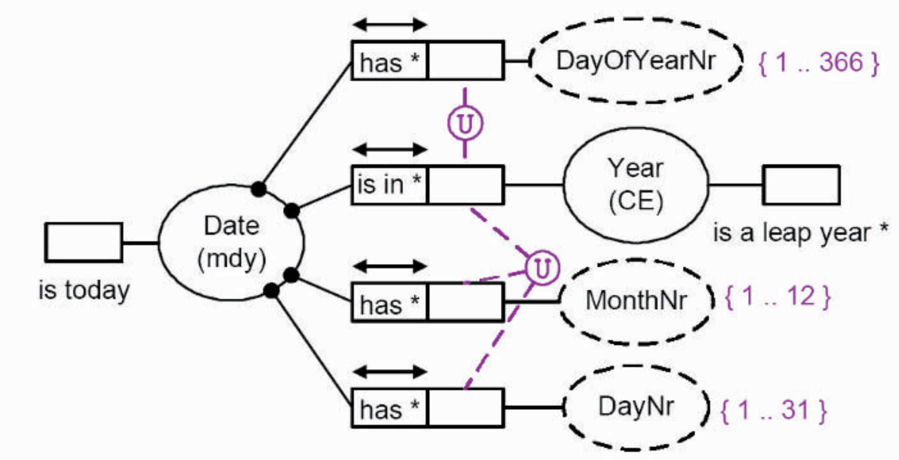
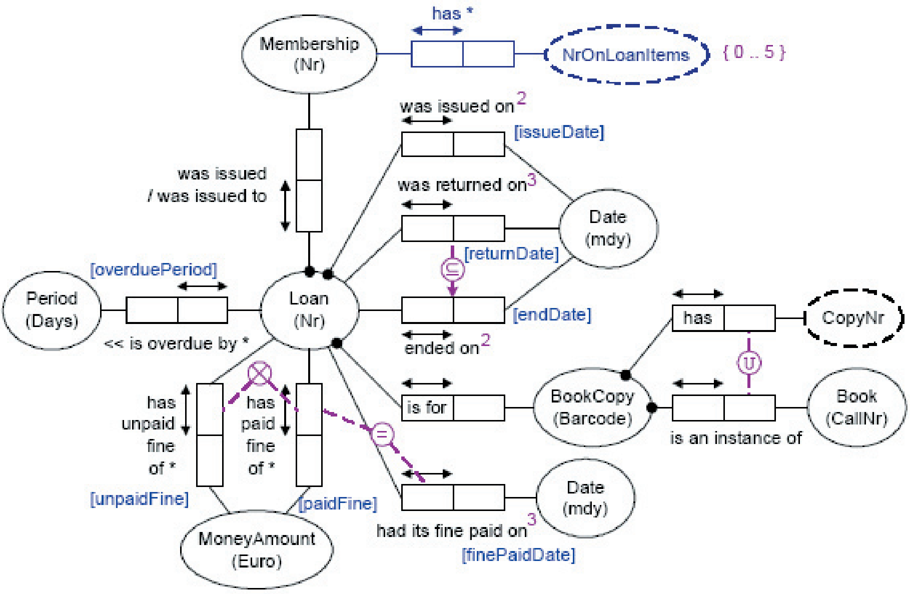


Figure 8: ORM subschema for library loans



[2] **For each** Loan, endDate >= issueDate
[3] **For each** Loan, finePaidDate >= returnDate

* Membership.nrOnLoanItems = **count each** Loan that was issued to Membership
and was returned on **no** Date
* Loan is overdue by Period **iff** Loan ended on **no** Date
and Period = **today** - Loan.issueDate - Loan.issueDate.year.normalLoanPeriod
and Period > 0
* Loan.unpaidFine = Loan.overduePeriod.**nrUnits** * Loan.issueDate.year.dailyLateFee
* Loan.paidFine = (Loan.returnDate - Loan.issueDate - Loan.returnDate.year.normalLoanPeriod).**nrUnits**
* Loan.returnDate.year.dailyLateFee

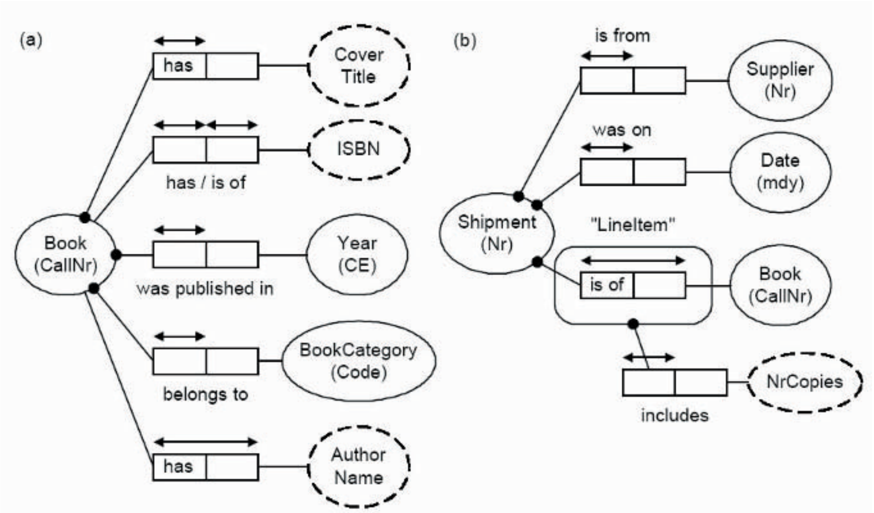
Membership.person.[age of Person on Membership.startDate] >= Membership.startDate.year.minimumMemberAge

If a person's birth date is not recorded, the age function returns null, and the whole expression evaluates to unknown. As in SQL, the constraint is violated if and only if it evaluates to false.

Figure 8 shows an ORM subschema for the main details about library loans. The circled "u" depicts an external uniqueness constraint, indicating that a particular copy (physical instance) of a book can be identified by combining the call number for the book with the copy number. As well as this composite identification scheme, a book copy also has a simple identification scheme (its barcode). The circled "=" depicts an *equality constraint* (a loan has a paid fine if and only if it had its fine paid on some date).

Each loan is for exactly one book copy. The subset constraint between the loan-return and loan-end associations declares that each loan that was returned on a date also ended on the same date. The superscripts "^{2}" and "^{3}" on fact types indicate that a textual constraint applies to them. In this case, the textual constraints are listed below the diagram. For each derived fact type (asterisked), a formal derivation rule declares how instances of the fact type may be derived from other facts. This example includes four derivation rules displayed below the diagram. ORM rules and queries (Bloesch & Halpin, 1997) may be formally specified in relational style and/or attribute-style (using role names and/or defined functions). The first derivation rule is expressed in relational style, the second rule in a combination of relational and attribute styles, and the last two rules in attribute style. The derivation rule for unpaid fines determines the fine currently accrued for an overdue loan—this amount may vary over time. The derivation rule for paid fines enables the system to compute the fine amount actually paid. The predefined nrUnits function converts a unit-based amount (e.g., three days) into a pure number (e.g., 3). This function may apply to any expression that returns a unit-based type, and enhances semantic stability by protecting rules against changes to choice of units.

Figure 9: ORM subschemas for details about Books and Shipments



Usually, constraints on derived fact types are themselves derivable. However, further constraints can be explicitly added to them (e.g., the value constraint on NrOnLoanItems). This provides a convenient and powerful way to declare various business rules that are awkward to express on base fact types.

Figure 9 shows basic subschemas for the book and shipment areas. If a book has an International Standard Book Number (ISBN), the library records this as well. These subschemas are straightforward, so should need no further explanation.

POSSIBLE BENEFITS OF ORM FOR DEMO

As explained earlier, the DEMO approach uses a state model to declare the "essential" fact types and rules pertaining to the real world objects in the application domain. A state model is specified using an object-fact diagram supplemented by an object property table. Figure 4 shows the state model for the library application. Collectively, Figure 6, Figure 8 and Figure 9 provide an ORM model for the library application. A comparison between these two models reveals some important differences.

An ORM model is intended to capture all the fact types that are of interest in the application domain, as well as all static business rules (constraints and derivation rules that apply to each individual state of the information system) that need to be enforced. ORM models are also formal, so that they can be automatically transformed into implementation models. For these reasons, ORM models tend to be more complete and precise than corresponding DEMO state models.

The first major addition provided by ORM models is their *inclusion of at least one identification scheme for each entity type*. For example, in Figure 6 we see that each loan is identified by a loan number, and each book copy is identified by a barcode. In addition, we see that each book copy can be identified by combining the call number of its book with a copy number. Any reference scheme that is to be used in the application is considered relevant. Apart from being needed for the operation of the information system, such identification schemes enable the modeler to use real examples when populating fact types for validation purposes (as shown in Figure 5). This makes it much easier to decide whether the model accurately reflects the application domain. As DEMO considers the choice of any identification scheme as non-essential, this kind of information is ignored.

The second major difference is that ORM models typically *capture more constraints*. For example, the DEMO-SM ignores any dependency between the unary fact types PE05 (BookCopy has been returned) and PE04 (Loan has ended to exist) because this is captured in the OM (and consequently in the PM). To enforce the dependency, the ORM model includes a subset constraint between the loan-return and loan-end fact types to ensure that each returned loan is classified as ended. In general, ORM's constraint language is more powerful (e.g., see Halpin, 2002b).

A third addition provided by ORM models is that *all temporal aspects are declared explicitly*. For example, consider the DEMO unary fact type PE04: Loan has started to exist. Like any other DEMO fact type, this has an implicit time stamp. In ORM, this is explicitly modeled using the fact type: Loan was issued on Date. This goes beyond the DEMO representation by including the *granularity* of the time stamp—in this case, day, rather than, for example, minute or second. This granularity choice is uncovered by inspection of sample requirements or by discussion with the domain expert. One of the design heuristics in ORM is to consider each fact type, and ask whether it needs to be treated in a snapshot or historical way. For example, consider the fact type: BookCopy has CopyNr. Although in the real world, instances of this fact type come into being at a given time (e.g., when assigned by the librarian), the recording of time stamp information for this fact type is not of interest to the users of the library application (as confirmed in interview sessions). Hence the ORM model excludes any temporal information about this fact type. In contrast, DEMO's ontological approach includes time stamps for all production events.

A fourth difference is that ORM provides *formal derivation rules* for relevant derived fact types. This makes it possible to automatically generate application code to enforce the rules. For example, consider the four derived properties listed at the bottom of the DEMO state model in Figure 4. Although precise, they are not expressed in a formal language, so are not executable. Although a derivation rule for computing a person's age is included in the ORM model, this applies only to those members who supply their birth dates. The library decided not to require all applicants to provide their birth date (this is left optional), allowing other ways to establish age (e.g., by visual inspection of the applicant). In contrast, the DEMO model assumes that birth dates are always known, and its derivation rule is based on this assumption.

Unlike a person's age, the determination of fines for overdue loans is always considered to be of interest to the system, as is the recording that such fines were paid. Unlike the DEMO model's single derivation rule for incurred fines, the ORM model includes two derivation rules, one to allow the computation at any instant for unpaid fines, and one to record fines that were actually paid (see Figure 8). The ORM model captures explicitly all decisions about what history to record in the information system.

In addition to enabling the formal capture of more information than DEMO state models, ORM provides *modeling procedures* and *formal transformation theorems* to assist modelers to create conceptual models and map them to implementation code. Details on ORM's conceptual schema design procedure and transformation theorems may be found in Halpin (2001a). Of particular interest in this regard is ORM's use of data use cases (samples of required information) to seed the model. For example, concrete instances of data required from an as-is or to-be library system can be extremely helpful for specifying an initial model. But this practice requires the use of value-based identification schemes (at least tentative ones) for the entities involved, an aspect ignored by DEMO.

For the above reasons, ORM appears to provide a useful supplement to DEMO, offering ways to flesh out state models to complete, executable data models, and providing further procedures to help in the modeling process itself.

POSSIBLE BENEFITS OF DEMO FOR ORM

ORM is a method for information modeling, in particular for developing conceptual database schemas. Although ORM can be used to model manual and/or automated information systems, it is especially useful for specifying an executable schema for a fully automated information system (AIS). Because of its data-oriented focus, ORM covers only part of the scope of a business system (BS). This section investigates what DEMO can add to ORM in this respect.

The first addition provided by DEMO is the distinction between a BS and an AIS, which DEMO treats as an automated realization of the I-system discussed earlier (see

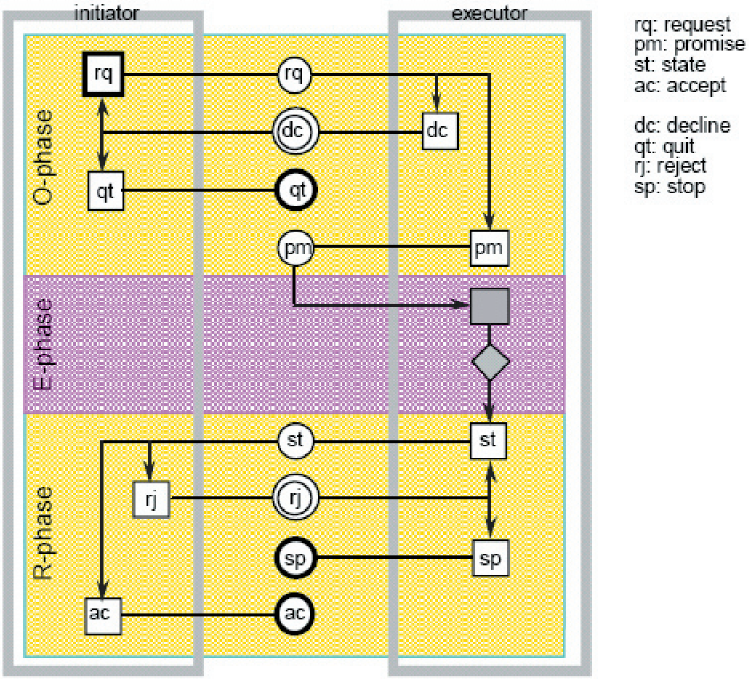
Figure 2). This I-system supports the B-system, which represents the abstracted essence of the organization. The kinds of support are purely informational: collecting, providing, recalling and computing knowledge about business acts and their results. The AIS and the BS can each be modeled as a discrete dynamic system (or discrete event system) (Hee, van Houben & Dietz, 1989), but of a different category—a BS is a social system, whereas an AIS is a rational system.

An AIS is a software system, so the only support it can offer is to provide information to the BS that is modeled in the AIS. Only in the BS may original facts be created (which can then be entered in the AIS). For example, the replenishment orders generated by an automated stock control system are just (computed) output information as far as the AIS is concerned. At the I-system level they are not business orders. Only by virtue of the declaration by the B-system do these information items count as replenishment orders.

The second contribution offered by DEMO to the design of an IS is a full account of the possible actions to be supported. The operating principle of a BS is the ability of human beings (in their role of social individuals) to enter into and comply with commitments and agreements. This was called coordination in this chapter. The standard pattern of C-acts and resulting C-facts of a transaction is shown in Figure 10. An open or white box represents a C-act type and an open or white disk represents a C-fact type. A gray box represents a P-act type and a gray diamond a P-fact type.

The initial C-act is drawn with a bold line, as is every terminal C-fact. The gray colored frames denoted by “initiator” and “executor” represent the responsibilities of the two partaking actor roles. The steps in the transaction process are ideal candidates for the functions (use cases) of the AIS. These are the atomic components of business processes;

Figure 10: The standard pattern of a transaction



there is nothing more to support. Using ORM, one (only) has to decide which actions will be supported and how, and which will not.

As a third contribution to ORM, DEMO distinguishes between the dynamics of the BS and AIS. Every C-fact may serve as an agendum (singular of agenda) to be dealt with by an actor. Typically, an actor disposes of a set of agendums, or agenda. In dealing with an agendum, one or more new agendums may be generated. This constitutes the dynamics of a BS. The dynamics of the AIS are basically *asynchronous* with respect to the dynamics of the BS. They coincide only when products of the AIS are declared to count as acts in the BS (like we have seen for an automated stock control system). In all other cases, the C-acts must be made known to the AIS.

Consider for example the borrowing of a library book. This transaction of type T04 starts with a member request. The resulting C-fact "requested" is entered in the AIS. At the same time, a new instance of Loan is created by the AIS, including the facts that existentially depend on it ("the membership of L is M" and "the book copy of L is C"). Ideally, the recorded time stamp of the C-fact is the real time (valid time) at which it was created in the BS. It is, however, common practice to take the time of entering into the AIS (transaction time) as the time stamp. This usually causes no problems since the order in which the steps of Figure 10 are entered in the AIS is easily controlled by the AIS. For example, a promise fact is rejected by the AIS if there is no corresponding request fact. If the transaction succeeds, the terminal state is the C-fact "ac" (accepted). At the time of establishing this fact, the production fact becomes existent. From that time on, the loan really exists in the BS. This definition is easily implemented in the AIS: as soon as the accepted fact is entered, the loan exists (there is only the unavoidable time delay with the BS).

CONCLUSIONS

This chapter outlined the essential features of the DEMO and ORM approaches to conceptual modeling, then explored various potential benefits of synthesizing both methods to achieve a more complete and productive approach to business and information system modeling. As both methods treat fact types as fundamental, it seemed judicious to use their fact models as a basis for integration. With this in mind, a basic library application was modeled in both DEMO and ORM, and then commonalities and differences between these models were examined.

As regards the benefits of supplementing DEMO with ORM, it seems clear that ORM offers several advantages for fleshing out DEMO state models into more comprehensive, formal data models that can be automatically transformed into application code. In particular, ORM models can extend DEMO state models by providing identification schemes, additional constraints, explicit and granular coverage of relevant temporal aspects, and formal derivation rules, as well as focusing on those features of actual interest to the automated information system. In addition, various ORM modeling procedures may provide additional assistance in the task of constructing models.

On the other hand, using DEMO in conjunction with ORM provides a more comprehensive modeling approach that goes beyond ORM's data-oriented perspective. In particular, DEMO provides a clean integration of static and dynamic aspects of business modeling, offering high level, implementation-independent ways of modeling the essential business processes in terms of the communication acts being performed by the business actors. Because communication acts may be modeled in terms of propositions (facts) and associated illocutionary forces, a clean integration with ORM's fact-based approach becomes feasible.

While our initial findings indicate positive benefits for synthesizing the DEMO and ORM approaches, a number of research problems require further analysis. In particular, the role of identification schemes in modeling needs further study. ORM mandates the use of such reference schemes early in the modeling process, while DEMO deliberately avoids them. The pragmatic consequences of this difference needs closer examination, as does the decision process involved in specifying automation boundaries to scope those aspects of the business that are to be implemented in an automated information system.

REFERENCES

Bakema, G., Zwart, J., & van der Lek, H. (1994). Fully communication oriented NIAM. In G. M. Nijssen & J. Sharp (Eds.), *NIAM-ISDM 1994 Conf. Working Papers* (pp. L1-35). Albuquerque, NM.

Bloesch, A.C., & Halpin, T.A. (1997). Conceptual queries using ConQuer-II. *Proceedings of the 16th International Conference on Conceptual Modeling ER'97* (pp. 113-126). Los Angeles: Springer LNCS 1331.

Chen, P. P. (1976). The entity-relationship model—Towards a unified view of data. *ACM Transactions on Database Systems*, *1*(1), 9-36.

De Troyer, O., & Meersman, R. (1995). A logic framework for a semantics of object oriented data modeling. *OOER'95, Proceedings of the 14th International ER Conference* (pp. 238-249). Gold Coast, Australia: Springer LNCS 1021.

Dietz, J.L.G. (1994). Modeling business processes for the purpose of redesign. *Proceedings of the IFIP TC8 Open Conference on BPR*. Amsterdam: North-Holland.

Dietz, J.L.G. (1999). Understanding and modeling business processes with DEMO. *Proceedings of the 18th International Conference on Conceptual Modeling ER'99*. Paris: Springer LNCS.

Dietz, J.L.G. (2003a). The atoms, molecules and fibers of organizations. *Data & Knowledge Engineering*.

Dietz, J.L.G. (2003b). Generic recurrent patterns in business processes. In W. van der Aalst, A. ter Hofstede & M. Weske. (Eds.), *Business Process Management*, LNCS 2678. Springer-Verlag.

Goldkuhl, G. (1996). Generic business frameworks and action modelling. In F. Dignum, J. Dietz, E. Verharen & H. Weigand (Eds.), *Communication modeling - The language/action perspective. Proceedings of the First International Workshop on Communication Modeling*. Electronic Workshops in Computing Springer. [Online] Available: http://www.springer.co.uk/ewic/workshops/CM96/

Halpin, T.A. (1998a). ORM/NIAM Object-Role Modeling. In P. Bernus, K. Mertins & G. Schmidt (Eds.), *Handbook on information systems architectures* (pp. 81-101). Berlin: Springer-Verlag.

Halpin, T.A. (1998b). Conceptual queries. *Database Newsletter*, *26*(2). Boston, MA: Database Research Group.

Halpin, T.A. (2001a). *Information modeling and relational databases*. San Francisco: Morgan Kaufmann.

Halpin, T.A. (2001b). Supplementing UML with concepts from ORM. In K. Siau & T. A. Halpin (Eds.), *Unified Modeling Language: Systems analysis, design, and development issues*. Hershey, PA: Idea Group Publishing.

Halpin, T.A. (2002a). Metaschemas for ER, ORM and UML: A comparison. *Journal of Database Management*, 4-13. Hershey, PA: Idea Group Publishing.

Halpin, T.A. (2002b). Join constraints. In T. Halpin, J. Krogstie & K. Siau (Eds.). *Proceedings of the Seventh CAiSE/IFIP-WG8.1 International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design* (pp. 121-131). Toronto, Canada.

Halpin, T.A. (2003). Verbalizing business rules: Part 1. *Business Rules Journal*, *4*(4). [Online]. Available: http://www.BRCommunity.com/a2003/b138.html

Halpin, T.A., Evans, K., Hallock, P., & MacLean, B. (2003). *Database modeling with Microsoft Visio for enterprise architects*. San Francisco: Morgan Kaufmann.

Hee, K.M. van, Houben, G.-J., & Dietz, J.L.G. (1989). Modelling of discrete dynamic systems; framework and examples. *Information Systems*, *14*.

Hofstede, A.H.M. ter, Proper, H.A., & Weide, th. P. van der (1993). Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, *18*(7), 489-523.

Medina-Mora, R., Winograd, T., Flores, R., & Flores, F. (1992). The action workflow approach to workflow management technology. In J. Turner & R. Kraut (Eds.). *Proceedings of the 4th Conference on Computer Supported Cooperative Work*. New York: ACM Press.

OMG UML RTF. (2003). *Unified Modeling Language (UML), Version 2.0.* [Online]. Available: www.omg.org/uml

Spyns, P., Meersman, R., & Jarrar, M. (2002). Data modeling versus Ontology engineering. *ACM SIGMOD Record*, *31*(4), 12-17.

Van Reijswoud, V.E, & Dietz, J.L.G. (1999). *The DEMO modeling handbook*. [Online]. Available: http://www.demo.nl

Van Reijswoud, V.E., Mulder, J.B.F., & Dietz, J.L.G. (1999). Speech act based business process and information modeling with DEMO. *Information Systems Journal*.

Wintraecken, J. (1990). *The NIAM information analysis method: Theory and practice*. Deventer, The Netherlands: Kluwer.

APPENDIX: BASIC DESCRIPTION OF THE LIBRARY CASE

The library described hereafter is one of the branches of the public library of Delftown. In the building in which it is located is a desk for lending books (the out-desk) and a desk for returning books (the in-desk). The in-desk is occupied by Louise and the out-desk by Tim and Kevin on turn. There is a third desk, called the information desk, which is occupied by Lisa. The books that may be borrowed are put on shelves, sorted on the category of the title. Every (copy of a) book is identified by a bar code.

At the information desk one can get information such as opening hours, loan rules, and membership fees, and of course about the books. There is a binder on Lisa's desk, which contains the complete library catalog, sorted in several ways (on author, on category and on title). One can freely browse through the binder to find the book one is looking for. Next to that, one can ask Lisa about the books in the catalog.

The information desk also serves as the registration desk. Anyone who wants to be registered as a member of the library has to apply with Lisa. She writes the data needed on a registration form. These forms are collected daily by someone from the central office. Within a few days, the new member receives a letter welcoming him/her as a new member and informing him/her about the library rules. The letter also contains the fee to be paid, and the message that the membership card can be collected at the branch office. By default, this fee is the standard annual fee as determined by the library board. Exceptions may be made for people without means. In that case, Lisa applies in writing to the library board for the reduced fee. Of course, she has to wait for the board's decision, which she also gets in writing, before the membership can be registered. One gets the membership card after cash payment of the fee. The membership card has a bar code on it representing the membership number.

If one wants to borrow a book, one has to take (a copy of) the book from the shelves and take it to the out-desk. Tim or Kevin will then scan the bar code on the membership card, as well as the bar code on the book. These data are automatically entered into the library information system (LIS). The book is now considered to be lent to the member. No more than five books may be lent simultaneously to the same member.

When one returns a book, one goes to the in-desk and hands the book to Louise. She scans the book code, which is automatically entered into LIS. On the screen of her computer, she sees whether the loan period is exceeded or not. If it is, she also sees the fine that has to be paid. The person who returns the book has to pay the fine right away and in cash. After payment, Louise marks the book in her computer as returned. If the loan period is not exceeded, she only enters that the book has been returned. Returned books are piled on a table next to Louise. About every hour Lisa collects the pile and puts the books back on the shelves.

Every month, the librarian (Maria) decides which titles should be added and how many copies per title have to be ordered. She does so on the basis of the announcements of new books she knows of (by means of flyers of publishers but also by surfing on the web).

At the start of a new calendar year, Lisa sends out invoices to all current members for the annual membership fee. Fees have to be paid in cash at the branch office. If applicable, she also sends renewal requests for the reduced fee to the library boards.

Chapter XII

Revisiting Workflow Modeling with Statecharts

Wai Yin Mok, University of Alabama in Huntsville, USA

David Paper, Utah State University, USA

ABSTRACT

In this chapter, we model business workflows using Harel's statecharts. We demonstrate that mapping to statecharts allows one to systematically identify potential workflow problems. Moreover, it also allows one to investigate specific properties inherent in actual business workflows. Our research focuses on three desirable properties of active database systems — termination, confluence, and observable determinism. As a theoretical lens for termination and confluence, we develop algorithms linking desirable active database system properties to workflow management systems problems. Preliminary validation of our algorithms is accomplished by mapping business workflows from a case study. Our research thus generates preliminary theory by developing a systematic method for identifying workflow problems.

INTRODUCTION

Business workflows can be well defined, predictable, and frequently executed. We thereby refer to these as structured business workflows. Such workflows can be automated by machines to reduce clerical tasks and potential human intervention errors. Workflow management systems (WMS) are a tested vehicle to facilitate automation of structured business workflows. WMS, which are new generations of computerized systems, are designed to manage automated parts of business workflows (Brunwin, 1994). By separating workflow definitions from application software, WMS provide process and knowledge independence, much like data independence provided by database management systems.

In this research, we use Harel's statecharts to model structured business workflows (Harel, 1987) for three reasons. First, Harel's statecharts are used in the Unified Modeling Language (UML) as a means for modeling behavior (Object Management Group, 1999).

Since the UML is the standard modeling language of the Object Management Group[1], Harel's statecharts will soon become common. Second, statecharts are easy to understand and they do not have the problem of exponential growth of states that plague ordinary state transition diagrams (Harel, 1988). We shall elaborate on this point in the Related Work section. Third, their semantics are rigorous enough for formal analysis on various aspects of structured business workflows (Harel & Naamad, 1996).

Within the framework of statecharts, we will show how to model workflow concepts and present algorithms that determine whether a given business workflow has certain predefined properties. We will then use a case study with Moore BCS (recently recast as Moore Wallace Incorporated) to explore the characteristics of a business workflow. The algorithms we develop in this study will become part of a software design tool that we will develop in the future.

RELATED WORK

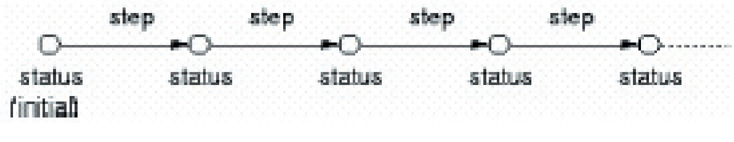
An overview of workflow management using the latest technology can be found in Georgakopoulos, Hornick, and Sheth (1995). Specification and implementation of exceptions in workflow management systems are discussed in Casati, Ceri, Paraboschi, and Pozzi (1999) and workflow evolution in Casati, Ceri, Pernici, and Pozzi (1998).

Active database systems have been studied extensively (Paton & Diaz, 1999). Active database systems and workflow management systems are related since both types of systems employ triggers to respond to external and internal events and exceptions. We are interested in three important properties of active database systems in this research, namely termination, confluence, and observable determinism, which are formally defined in Allen, Hellerstein, and Widom (1995). More discussion on active database systems, which includes several research prototypes and commercial products, can be found in Zaniolo (1997).

The statemate approach, which uses statecharts in modeling reactive systems, is described in Harel and Politi (1998) and its semantics in Harel and Naamad (1996). By far, the statemate semantics of statecharts is the most rigorous and precise execution semantics defined for statecharts and it has been in use for more than ten years (Harel & Naamad, 1996). Here we point out the most significant aspects of the execution semantics. The reader may consult Harel and Politi (1998) and Harel and Naamad (1996) for details.

The behavior of a system described in statemate semantics is a set of possible runs, each representing the responses of the system to a sequence of external stimuli generated by its environment[2]. A run consists of a series of detailed snapshots of the system's situation; such a snapshot is called a status. The first in the sequence is the initial status, and each subsequent one is obtained from its predecessor by executing a step (see Figure 1).

Figure 1: Status and step



Some of the general principles of statemate semantics are as follows:

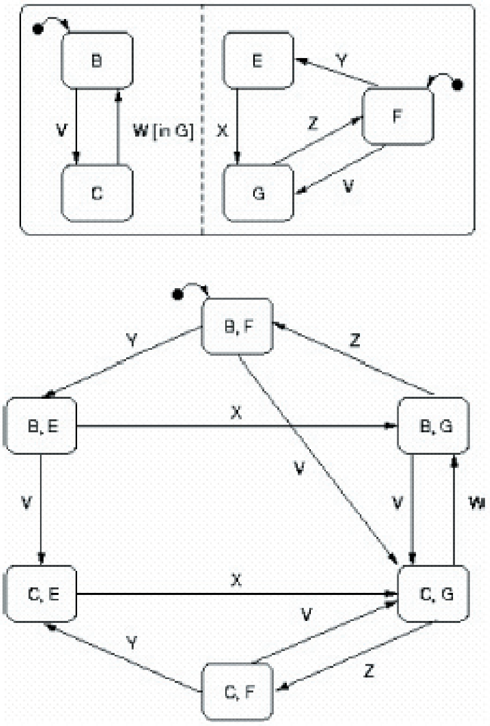
1. Reactions to external and internal events, and changes that occur in a step, can be sensed only after completion of the step.
2. Events are "live" for the duration of one step only, the one following that in which they occur, and are not "remembered" in subsequent steps.
3. Calculations in one step are based on the situation at the beginning of the step (e.g., the states the system was in, the activities that were active, and the values of conditions and data-items at that time). Updates of data items only occur at the end of a step.
4. A maximal subset of non-conflicting transitions is always executed.

Item 3 deserves more explanation. As an example, suppose there is an action:

*"X := X + 1; Y := X * 5", which is executed in a step. Further suppose that X is equal to 4 at the beginning of the step. Because of Item 3, after executing the step, X becomes 5 and Y becomes 20. Note that every computation of the action does not influence any other computation of the action. The semicolon separating the actions means, "do this too" rather than "and then do" in statemate semantics.*

Using activity diagrams to model workflows is discussed in Chapter 19 in Booch, Rumbaugh, and Jacobson (1999). Note that activity diagrams are special statecharts in which all

Figure 2: Exponential growth of states



of the state transitions are triggered by completion of activities in the source states. Activity diagrams are not designed to handle events. Since exceptions may happen during the execution of a workflow instance and exceptions are best modeled as events, activity diagrams can only model very simple workflows. In this sense, statecharts are more appropriate for modeling realistic workflows.

Before we show how to use statecharts to model workflow concepts, we present an example, adapted from Harel (1987), that shows the problem of exponential growth of states that plague ordinary state transition diagrams. In Figure 2, a statechart and its equivalent state transition diagram are presented. Note that by making use of an and-state in the statechart, we can easily model concurrency in a system by orthogonal components in the and-state. On the other hand, to perform the same modeling in the equivalent state transition diagram, we require six states. Using the same reasoning, for an and-state with two orthogonal components with a thousand states in each of them, the equivalent state transition diagram would require a million states. It is easy to see that it is difficult, if not impossible, to model concurrency in ordinary state transition diagrams because of the problem of exponential growth of states.

BASIC CONCEPTS AND TERMINOLOGY

The Workflow Management Coalition[3] (WfMC) has published numerous documents on various aspects of business workflow. We now introduce some basic concepts and terminology defined by WfMC. A *business workflow*, or simply a workflow, is a set of activities which collectively realize a business objective. An insurance claims process is an example. A workflow is defined in a *workflow definition* that consists of a network of activities. Usually a workflow definition is a formal representation of a business workflow. An activity is a logical step within a workflow. As such, it is usually the smallest unit of work within a workflow. Further, an activity can be manual or automated. A *workflow management system* is used to manage automated activities, but not manual activities. A *workflow instance* is the representation of a single execution of a workflow. It has its own workflow instance data and is capable of independent control as it progresses towards completion. The processing of an insurance claim for a particular customer is thus an example of a workflow instance of the insurance claims process.

Similarly, an *activity instance* is the representation of a single invocation of an activity within a workflow instance. Several activity instances may be associated with a workflow instance, but one activity instance cannot be associated with more than one workflow instance.

MODELING WORKFLOW CONCEPTS

A business workflow can be formally represented by a statechart. Each workflow instance has its own copy of the statechart. An activity of a workflow, whether it is manual or automated, is represented by a state in the statechart. An activity is being carried out only if the system resides in the state that corresponds to that activity.

Transitions between activities are thus modeled as transitions between states in the statechart, which are triggered by *events* and guarded by *conditions*. Events can be external (generated by elements outside the statechart) or internal (generated by elements inside the statechart). A transition between a source state and a target state will take place if and only if the system currently resides in the source state, and the event of the transition occurs, and the conditions that guard the transition are true. In other words, the system must be in the source state, the event must occur and the conditions must be true for the transition to take place.

The four possible types of routing in workflows are sequential, parallel, conditional, and iterative (van der Aalst, 1998). In the following subsections, we show how to model these four types of routing in statecharts, and illustrate several statechart concepts that are relevant in modeling workflows.

Sequential Routing

Activities are executed one after the other in sequential routing. In Figure 3, E1 is the event, C1 is the condition, and A1 is the action of the transition between state A and state B. In Figure 3, the system may still reside in state A unless at the instant E1 occurs, C1 is true. Events are instantaneous. A transition between a source state and a target state will take place if and only if the system currently resides in the source state, and the event of the transition occurs, and the conditions that guard the transition are true. In other words, the system must be in the source state, the event must occur and the conditions must be true for the transition to take place.

An action can be sending an event, but the event can be lost if the system is not in the proper state. As an example, suppose the transition between state B and state C fires and A2 is the action "sending event E1". Since the system is not in state A, E1 is simply lost.

Parallel Routing

In contrast to sequential routing, activities can be executed concurrently in parallel routing. This is exactly why several activity instances may associate with a workflow instance. We do not specify conditions in Figure 4. By default, they are assumed to be the completion of the activities represented by the source states. If additional conditions are given for a transition, then the actual guarding condition of the transition is the conjunction of the additional conditions and the completion of the activity represented by the source state, unless otherwise stated. As an example, in Figure 3, the actual guarding condition of the transition from state A to state B is the completion of the activity represented by state A and the additional condition C1. Events and actions are also omitted. If the event of a transition is omitted, then the system will check the condition continuously. Thus, whether

Figure 3: Sequential routing

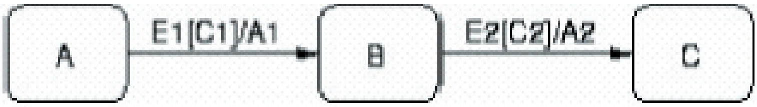
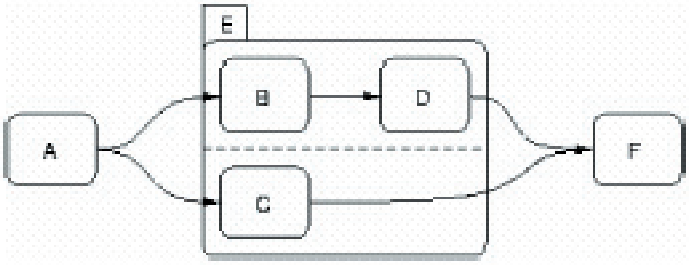


Figure 4: Parallel routing



the transition will take place may depend solely on the condition of the transition (Harel & Politi, 1998). Hence, transitions do not have to depend on any particular events and actions do not have to be performed during transitions. In Figure 4, E is an *and-state*, which has two orthogonal components. Being in E means being in these two components simultaneously. The *fork construct* specifies that when the system exits state A, it will enter states B and C simultaneously. The *merge construct* specifies that the system will leave state E only if it resides in states D and C simultaneously. However, because of the default conditions, this transition will only take place if activities D and C are both completed. Thus, synchronization occurs at merge constructs. Fork and merge constructs can be used to model the and-split and and-join defined by WfMC (Workflow Management Coalition, 1999).

Conditional Routing

The system will choose one activity among several target activities to execute in conditional routing. The decision depends on the truth or the falsity of the conditions of the transitions. In Figure 5, C1 and C2 are two mutually exclusive conditions and the system can only enter either state B or state C but not both. Conditions in statecharts can be used to model the or-split and or-join defined by WfMC (Workflow Management Coalition, 1999).

Figure 5: Conditional routing

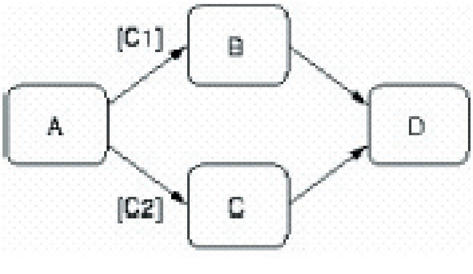
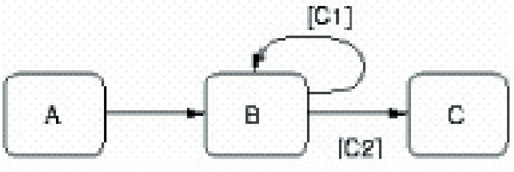


Figure 6: Iterative routing



Iterative Routing

Iterative routing is similar to conditional routing. Again, C1 and C2 are two mutually exclusive conditions. Whether or not the system stays in state B in Figure 6 depends on the truth or falsity of the iterative condition C1.

DESIRABLE PROPERTIES

Workflow management systems and active database systems both employ triggers to respond to exceptions and events. Thus desirable properties of active database systems are also applicable to workflow management systems. We chose to examine three salient desirable properties of active database systems, namely, termination, confluence, and observable determinism. Given a statechart of a business workflow, we present several procedures to determine whether the given statechart has these properties.

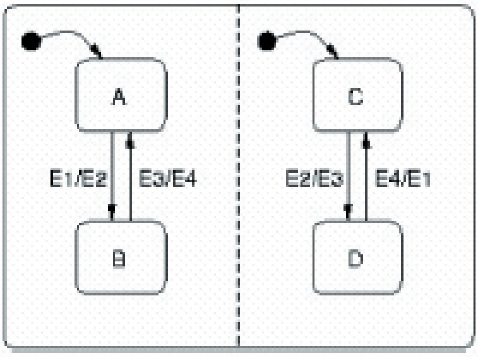
Termination

As discussed in the Modeling Workflow Concepts section, external events are generated by elements outside the given statechart. Internal events, on the other hand, are generated by elements inside the statechart. Sometimes events can be generated both externally and internally. For example, consider the statechart of a machine. The event “power off” can be generated externally by an operator when he shuts down the machine or the event may be generated internally by the machine itself when it is overheated. Generation of events may lead to infinite execution of a statechart. In Figure 7, which is adapted from Figure 47 in Harel (1987), once the event E1 occurs externally, events E2, E3, E4, and E1 will be generated in this order internally, forever, meaning that the statechart will never terminate. A workflow design tool should be able to detect cycles of this sort before the actual deployment of the system. In this way, termination problems can be detected and corrected during modeling rather than in production.

There are certain distinguishing features in Figure 7. First, it leads to infinite execution and second, it contains cycles. In fact, a statechart that leads to infinite execution always has a cycle even though having a cycle in a statechart does not mean that the statechart will always lead to infinite execution. Third, in Figure 7, the transitions on the cycles are triggered by internally generated events or conditions that will never run out or never be false.

Note that in the third point above, the condition that the “internally generated events or conditions that will never run out or never be false” is important since internally generated events may eventually run out or conditions of transitions may eventually become false, as shown in the following example.

Figure 7: A statechart that will not terminate



In Figure 8, if *x* is equal to 5 initially, we will only go through the loop 5 times. However, detecting transitions of this kind requires complicated analysis of the actions of the transitions. In the literature, Baralis, Ceri, and Paraboschi (1996) contain complicated algorithms for this kind of analysis, which may lead to a long execution time. Our techniques, on the other hand, are only based on reading the values of and writing values to data items. Admittedly, our techniques do not have the precision of those in Baralis et al. (1996).

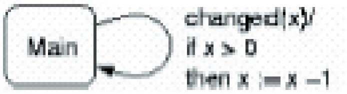
We now introduce Algorithm 1. Algorithm 1 provides a mathematical procedure that determines whether or not a given statechart terminates. The proof of Algorithm 1 theoretically validates the viability of termination as a critical property of business workflows.

Algorithm 1

Input: A statechart.

Output: Yes or no. (Yes means the statechart may have non-termination problems. Note that our analysis is very conservative in the sense that if our algorithm says "yes", the statechart may still terminate because events may run out or conditions may become false on a transition. However, as we have just mentioned, detecting situations of this kind must be done by careful analysis of the actual computations of the transitions, which we do not perform here. As an example, in Figure 8, after examining the computation of the transition, we can conclude that the loop will eventually terminate.

Figure 8: A statechart that will terminate



Nevertheless, for much more complicated transitions which may have hundreds or thousands of lines of codes, analysis could be hard, if not impossible, to perform.)

1. If there is no direct cycle constructed from states and transitions in the statechart, the statechart will terminate and we may stop and say “no”; otherwise let S be the set of such cycles. For each cycle s in S, if s contains a transition whose event can only be generated externally or whose condition can only be set to true externally, s will not cause non-termination and we may remove s from S.
2. If there is no cycle constructed from internally generated events or conditions in the statechart, the statechart will terminate and we may stop and say “no”; otherwise let E be the set of such cycles.
3. If there exists an element s in S and an element e in E such that the events that take the system from one state to the other in s is a subsequence of e, then the statechart will never terminate and we say “yes”; otherwise the statechart will terminate and we say “no”.

Theorem 1. Algorithm 1 specifies sufficient conditions for a statechart to terminate.

Proof: In Step 1, if there is no direct cycle constructed from the states and transitions in a statechart, the statechart will terminate since the activity associated with a state will terminate and each state in the statechart will only be visited once. In case there is such a cycle s, and there is a transition of s whose event can only be generated externally or whose condition can only be set to true externally, the completion of s depends on external interventions. Thus, s will eventually be stopped by external means. In Step 2, if there is no cycle constructed from internally generated events or conditions in a statechart, the events and conditions are not “self-feeding”, which means the events will eventually run out and the conditions will eventually become false. On the other hand, in Step 3, if we can find such a cycle s and a self-feeding cycle e of events and conditions, then s will never stop once s is started.

As an example, in Figure 7, S is {[A, B, A], [C, D, C]} and E is {[E1, E2, E3, E4, E1]}. Let s be [A, B, A] and e be [E1, E2, E3, E4, E1]. The events that take the system from one state to the other in s is [E1, E3], which is a subsequence of e. Thus the statechart will never terminate.

In the Statechart Analysis section, we will use Algorithm1 to demonstrate its ability to detect a non-termination problem from the actual workflow scenario illustrated in Figure 10. In the next section, we discuss confluence.

Confluence

Consider a set of non-prioritized transitions that are fired at the same time. If the final status of the system does not depend on their order of execution, then the system is *confluent*.

Whether a system is deterministic or not has a great impact on the confluence of the system. For example, in Figure 5, if C1 and C2 are not mutually exclusive, then both of them could be true at the same time and thus the system needs to non-deterministically choose either state B or state C to enter. To avoid situations like this, for each conditional routing, we require the user to prioritize the alternatives so that in case there is a tie, a tiebreaker is provided.

Two transitions are in conflict if there is some common state that would be exited if any one of them were to be taken (Harel & Naamad, 1996). Nonconfluent statecharts, or systems, are caused by non-determinism of execution and conflicting transitions in the statecharts. In other words, when a statechart encounters non-determinism (that is, when there is more than one possible execution sequence of the conflicting transitions in a step), the final database state may be different due to a different order of execution of the conflicting transitions. However, for some conflicting transitions, a different order of execution may still lead to the same final database state after the statechart becomes stabilized. This is because they do not have a read-write racing problem or a write-write racing problem, which are defined as follows:

Two transitions t_1 *and* t_2 *have a read-write racing problem if* t_1 *reads the value of a data item x and* t_2 *writes a value to x.*

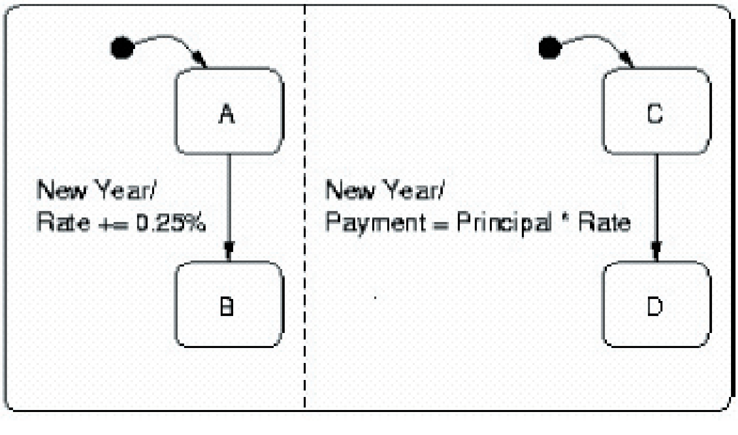
Two transitions t_1 *and* t_2 *have a write-write racing problem if both* t_1 *and* t_2 *write values to a common data item x.*

The two racing problems mentioned above are related to concurrency control problems inherent in database management systems (Bhargava, 1999). In this research, however, we adhere to the terminology used in the statechart literature. That is, we will keep using the terms read-write racing problems and write-write racing problems.

Figure 9 demonstrates a read-write racing situation. In this example, when the event "New Year" occurs, two transitions take place at the same time. Whether the new payment will be based on the new interest rate or the old interest rate depends on which transition is executed first. In this case, the statechart is not confluent.

We now introduce Algorithm 2, whose purpose is to identify a set of concurrently executed transitions that may lead to non-confluence.

Figure 9: A read-write racing problem



Algorithm 2

Input: A set T of concurrently executed transitions.
Output: A partition of the transitions in T such that each partition class may lead to non-confluence.

(Note that once again our analysis is very conservative in the sense that theset N outputted by Algorithm 2 may not lead to non-confluence. As in Algorithm 1, to conclude that there is non-confluence, we must carefully study the actual computations of the transitions, which we do not perform here.)

1. We first create a graph G with each transition in T as a vertex in G. However, G has no edge in this stage.
2. If two distinct transitions t1 and t2 have a read-write racing problem or a write-write racing problem, we add an edge to the two corresponding vertices in G. We continue this step until no more edges can be added to G. Note that at most n(n-1)/2 edges are added to G if n is the number of transitions in T.
3. The transitions in each connected component of G with at least two transitions may lead to non-confluence.

Theorem 2. Algorithm 2 correctly identifies sets of transitions that potentially lead to non-confluence.

Proof: Note that Algorithm 2 partitions the set T into disjoint subsets of T. Consider a partition class C where C is a connected component in G. If C has at least two transitions, then a transition in C has either a read-write racing problem or a write-write racing problem with another transition in C. Switching the order of execution of these two transitions will cause different final states of the system.

We may repeat Algorithm 2 until the set T becomes empty. At that time, there are no more transitions to be removed from T. Note that all the sets of transitions outputted by Algorithm 2 that may potentially lead to non-confluence have at least two transitions.

For any such set N and for any transition in N, there is another transition in N such that they have either a read-write racing problem or a write-write racing problem. In this way, Algorithm 2 points out the set of transitions that may potentially lead to non-confluence to the analyst and the analyst may consult with the client to devise a solution to the problem. The next theorem is interesting in the sense that the statemate semantics of statecharts avoid certain problems.

Theorem 3. If a statechart S implements the statemate semantics, then read-write racing problems will not cause non-confluence.

Proof: Since each transition is prioritized and calculations in one step are based on the situation at the beginning of the step, and updates of data values only occur at the end of a step, the data values read during the execution of a step are all produced in the previous step. Thus any data values produced during the execution of a step will not be read by any transitions executed in the same step. Therefore, read-write racing problems will not cause non-confluence.

As an example, if the statechart in Figure 9 implements the statemate semantics, then the calculation of the payment will be based on the old interest rate rather than on the new interest rate. The calculation of the new interest rate, of course, is also based on the old interest rate. However, the new interest rate is not available to the other calculations that occur in the same step.

In the Statechart Analysis section, we will use Algorithm 2 to identify the set of transitions in Figure 10 that may lead to non-confluence.

Observable Determinism

A transition is *observable* if its action is visible to the environment. A good example would be "print Profit" where Profit is a variable. Consider a set of non-prioritized and observable transitions that are fired at the same time. If the order of the output of the system does not depend on their order of execution, then the set is *observably deterministic*.

Theorem 4. If a statechart S implements the statemate semantics, then S is observably deterministic.

Proof: In the statemate semantics of statecharts, all actual updates of data items (or variables) are done at the end of a step (Harel & Naamad, 1996). Thus, any values that are displayed or printed out during the execution of a step are updated at the end of the previous step. Hence, displaying values to the environment cannot interleave with updating of values within the system. Therefore, if a system implements the statemate semantics, then it is observably deterministic.

The next theorem shows the relationship between the properties Observable Determinism and Confluence.

Theorem 5. If a statechart S is observably deterministic, then S is confluent.

Proof: For each transition in S, we add the action "show the entire current status of S". Thus if S is observably deterministic, then the printout of the status of S will be deterministic, which means S is confluent.

By Theorem 5, if a statechart is not confluent and its outputs are visible, then it is not observably deterministic. In the Statechart Analysis section, we will identify the set of transitions in Figure 10 that may lead to non-observable determinism.

CASE STUDY

We introduce a case study to theoretically validate our algorithms within a real-life context. Benbasat, Goldstein and Mead (1987) and Yin (1994) endorse the use of case studies to capture knowledge from practice. Our study generates theory with the assistance of algorithms. These algorithms prove that statecharts are valuable in determining termination, confluence, and observable determinism in workflows. Therefore, the results of the algorithms provide validated theory related to workflow properties. We also extend our theory by testing these properties in a real-life context (the case study). The case study approach offers a vehicle to construct applied theory from scholarly theory.

The case study was with Moore Business Communication Services (BCS), located in Logan, UT. Since the case study was administered in 1999-2000, Moore BCS has been recast as Moore Wallace Incorporated (MWI). From this point forward, we will use the MWI name when we refer to the case. MWI is a large company with approximately 2.32 billion dollars in 2003 revenue. MWI helps large corporations increase their competitiveness by improving the effectiveness of important business-to-customer communications. It provides consulting, project management, reengineering and distribution of high volume, customized communications to its clients. MWI delivers personalized, easy-to-read documents that facilitate a positive impression on an organization's customers. Its reengineering and redesign services help to ensure that the client organization's business communications have high quality and clarity.

By outsourcing with MWI, clients can divert their internal resources to other priorities because the dedicated production facilities can be trusted to help ensure faster cycle times, ultimately reducing overall costs. Equipped with the latest print and digital technologies, MWI has become a market leader in managing critical business communications.

MWI offers products and services that include statement/billing, cards (e.g., phone cards, credit cards, etc.), government noticing, policyholder and plan member communication, and database marketing. The technology environment at MWI paces, and in many areas leads, the marketplace in its industry.

Case Study Methodology

In the spring of 1999, we embarked on a case study of the card recovery system at MWI. The goal of the research was to map the existing state of the card recovery system process. Once mapped, we were charged with redesigning the process to remove redundancies and improve the overall effectiveness of the system. However, we were not responsible for implementing suggested changes. Our job was to examine the overall process of the system and devise a set of recommendations for management. The study began in January 1999 and was completed in December 2000. We were able to speak with several BCS employees, but our main contacts were Ferris Jorgensen, Phone Card Project Manager, Dennis Elwood, National Manufacturing Systems Project Manager, and Harvey Black, Project Manager.

Our last meeting with Dennis was on December 4, 2000 to discuss future research and refine our theoretical assumptions. We have built a solid relationship with MWI over the past several years. As a result, we have a trusting relationship and are able to gather additional data when needed.

Analysis of the card recovery process was conducted in four distinct phases. Phase one consisted of the problem definition. The problem was within the context of a problem statement. The problem statement was agreed upon by all parties involved and signed on February 24, 1999. The problem statement reads as follows: "MWI has a phone card division. During production, cards may become damaged or lost. The company has a need for a system that will track missing and replacement cards through the production cycle." A phone card recovery system existed prior to the research, but was not fully automated. During a meeting on March 3, 1999 with Ferris Jorgensen, it was decided that an updated system was needed to track missing and replacement cards through the production cycle because it was becoming an unacceptable cost to the organization. We worked closely with a small team of systems analysts and programmers to develop an accurate map of the existing system. Once the map was refined by us and the other team members, we redesigned the system as a working prototype. MWI can use this working prototype to integrate into their existing information systems infrastructure. We developed a prototype system because we didn't have the charge to implement a new system in accordance with existing systems. Phase two consisted of studying the current physical system. This involved building entity relationship diagrams, data flow diagrams, statechart diagrams, and completing a feasibility study. Phase three consisted of defining end-user requirements. End-user requirements are the functional and technical needs of the logical new system. Phase four consisted of clearly defining the possible alternatives and selecting a feasible solution. After careful analysis and several meetings with our team and key managers, it was decided that the prototype would be built using MS Access, MS Excel, and MS Visual Basic. This choice allowed us to develop a fully functional prototype without having to build the necessary error-checking routines. In addition, our choice of platform can be integrated with existing BCS systems.

A meeting on March 3, 1999 revealed the specifics of the existing process. An excerpt from the meeting follows: "When a card is found missing or damaged, the operator fills out a missing slip form and turns it in to a central processor, who then enters the information into a spreadsheet and forwards the request for replacement cards to the programmers. A replacement card is then produced and inserted into the proper bundle to be shipped." As can be seen from this narrative, the process is not very well automated because it requires several people to communicate process changes on a continuous basis. As such, process accuracy is suspect because of the tremendous potential for human error, and efficiency is low because of the large amount of human observation needed to continuously monitor the process. The meeting also revealed the specific purpose of the redesigned system: "The purpose of the new application is to automate the card recovery process in an attempt to increase efficiency and accuracy. The new system should reduce the need for entering the original data several times. It should also make the entire process nearly 'paperless' by eliminating several iterations of the same forms and information."

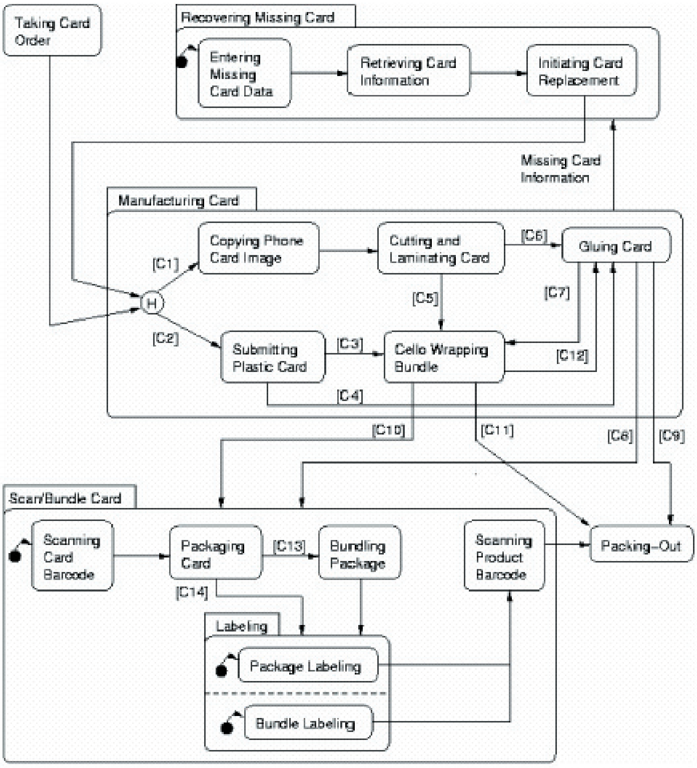
A meeting on March 23, 1999 revealed the system requirements — system inputs and functions, general system requirements, attributes to track, system outputs, and reports. Detailed system requirements are too vast to mention here, but we thought it prudent to include a few to give the reader a sense of the project's scope. Some of the system inputs and functions included start number, end number, and enter missing number. Some of the general system requirements included implementing a virtually 'paperless' process, eliminating forms, and building capability to determine where problems occur most frequently. Some of the attributes to track included programmer ID, project manager, client number, and workstation. System outputs included missing card 'slip'. System reports included error occurrence and orders sent to programmers for replacement cards.

The design phase included acquisition and design of the newly mapped system. A request for proposal (RFP) was written to communicate to vendors the desired features and requirements. The primary intent of the RFP was to solicit specific configurations, prices, maintenance agreements, conditions regarding changes made by buyers, and servicing. The RFP also conveys proposals for evaluating criteria, closure, postmark dates, and constraints. Meetings were held in early April to refine the RFP. The design specifications were agreed upon during April 1999. The design specifications explained the physical system requirements and the proposed prototype of the new system. The document included design of computer outputs, database and computer files, computer inputs, terminal dialogues and user interfaces, and methods and procedures.

The implementation phase included construction of the new system (prototype) and delivery of the new system. Meetings during April and May 1999 were conducted to facilitate this phase of the project. Construction of the prototype included building, testing, recording data, and developing integrated databases for the network of connected computers. Construction of the prototype also included installation and testing of new software packages, and writing and testing new programs. Delivery included developing conversion plans including database installations, end-user training, and physical conversion. Delivery also included the writing and delivery of the User Manual.

To provide a sense of how the process actually works, we now briefly describe the basic steps involved in the MWI workflow. A graphic of the workflow is depicted in Figure 10. Manufacturing (Manufacturing Card) receives an order for cards. At this point, an operator is responsible for checking whether the cards are in raw form or already laminated. If the cards are not laminated, the operator creates an image through a copy process. From this copied image, the cards can now be accurately cut and laminated. The cards are then either sent for gluing (the image must be glued to a rigid backing for stability) or sent to be bundled in a larger package and then glued. If the cards are already laminated, they are either sent for gluing or sent to be bundled in a larger package and then glued. Once bundled by another team of operators, cards can be sent directly to the Packing-Out stage for final delivery if the order requests this action. The same is true at the Gluing Card stage. Once glued, the cards can be sent directly to Packing-Out. However, this is not the norm because cards sent directly to Packing-Out are never scanned and therefore are not tracked properly in the system. As a result, we recommend that all bundled and glued cards be sent to the

Figure 10: MWI workflow



Scan/Bundle Card stage prior to Packing-Out. Unfortunately the existing process does not ‘force’ this recommendation, resulting in potential losses in efficiency. An operator is then responsible for checking for misplaced cards and defects at the end of the Manufacturing Card stage (if the normal process flow is followed). As such, any cards suspected of being misplaced or damaged are replaced. Data on misplaced and damaged cards are sent to the Recovering

Missing Card stage so that the data can be analyzed and appropriate action taken. From this point, all glued cards (if the normal process flow is followed) are sent to the Scan/Bundle Card stage. Barcodes for cards are then scanned so that this data can be properly stored in the database. Cards can be in either a single package or a bundle, depending on the order request. Packages or bundles are then labeled and scanned. Scanning is done twice because bundles and packages have distinct barcodes from an individual card. Finally, bundles and packages are sent to the Package-Out stage for shipment to customers.

Statechart Analysis

This research focuses on the statechart we generated from our analysis of the card recovery process. We analyzed the statechart in terms of the three properties defined earlier. With the aid of algorithms, we examined the workflow within the context of statecharts to determine potential problems with the workflow. From our analysis and algorithmic generation, we were able to build preliminary applied theory that we believe can assist systems designers in their attempt to design effective and accurate workflows.

In addition to the default conditions, additional conditions are shown in Figure 10. According to Harvey Black, those 14 extra conditions in Figure 10 (C1 – C14) are provided by the user. However, these conditions must satisfy the following rules:

1. (C1 xor C2) = True,
2. (C3 xor C4) = True,
3. (C5 xor C6) = True,
4. (C7 xor C8 xor C9) = True,
5. (C10 xor C11 xor C12) = True,
6. (C13 xor C14) = True.

These rules specify that for the conditions in each rule, one and only one can be true at any given time. For example, in Rule 4, at any moment in time, there are only three possibilities, namely, either C7 = True, C8 = False, and C9 = False; or C7 = False, C8 = True, and C9 = False; or C7 = False, C8 = False, and C9 = True.

Analysis of the statechart we developed revealed some potential problematic structures in the same. Note that there are two cycles. One is from the high-level state Manufacturing Card to the high-level state Recovering Missing Card and back to Manufacturing Card. The other one is from the basic state Cello Wrapping Bundle to the basic state Gluing Card and back to Cello Wrapping Bundle. However, the event on the transition from Manufacturing Card to Recovering Missing Card is external. Therefore, the transition from Manufacturing Card to Recovering Missing Card will only take place when the event Missing Card information happens. We do not believe this event will happen all the time and thus the first cycle does not cause any critical problem. For the second one, since the transitions do not depend on external events and it is possible that C7 and C12 are both true, it is possible that non-termination may occur. We therefore advise the user that either C7 or C12 must eventually become false in the specification. Otherwise, there is a fundamental flaw in the workflow; that is, the cycle may never terminate.

Algorithm 1 provides a means to detect and correct non-terminating cycles. However, it provides much more. It forces us to analyze the statechart in a systematic manner. Our first analytical action after we finished design of the MWI workflow statechart was to identify cycles in the workflow. Once all of the cycles were identified, we then began to look closely at each cycle for possible non-termination. Without Algorithm 1, we would never detect or even suspect non-termination problems. Thus, Algorithm 1 acts as a high-level analytical tool to systematically identify and correct non-termination problems within a given statechart. System designers can identify, discuss, and correct potential cycle non-termination problems during design rather than attempt to correct problems in production. Of course, system designers can also correct non-termination problems for existing systems in the manner discussed in the MWI case.

Another potential workflow problem is confluence. *Algorithm 2 provides a means to identify a set of transitions that may lead to non-confluence*. In Figure 10, we identified an and-state in the sub-state Labeling contained in the high-level state Scan/Bundle card. According to Harvey Black, sometimes package labels are put on bundles or bundle labels are put on packages. Thus, the and-state Labeling may have a write-write racing problem.

Notice that we spoke with Mr. Black once we noticed the and-state. After Mr. Black was made aware of the and-state, he was able to better understand where the workflow problems were occurring. Of course, we didn't explain the details of statecharts to Mr. Black. We instead explained to him the situation in business language. As a result of our intervention, MWI is attempting to rethink the labeling process.

Hence, Algorithm 2 provided a systematic basis for redesign. By using the principles developed in Algorithm 2, we were able to flag the and-state structure as a source of potential problems and inform the user about it.

Another potential workflow problem is observable determinism. As Theorem 5 indicates, if a statechart is not confluent and its outputs are visible, then it is not observably deterministic. Since the and-state Labeling has a write-write racing problem and its outputs are visible to the environment, the outputs of the and-state Labeling are not observably deterministic. After explaining to Mr. Black the concept of observable determinism, he was able to identify a potential workflow problem. During label printing, there is a real danger that package labels and bundle labels can be switched. Although it is easy to distinguish by eye the difference between a bundle (a set of packages) and a package, it is very possible that an operator will accidentally place a bundle label on a package and vice-versa. Keep in mind that the labels are both plain white and the bar codes are not easy to see with the naked eye. As a result of this analysis, Mr. Black has suggested to management that bundle and package labels be made different colors. Although color printing is more expensive, the reduction in errors should more than justify the investment.

Algorithms 1 and 2 offer a systematic means to identify problems in complex business workflows. By using the principles developed in this study, one can scan any statechart quickly and efficiently to flag potential workflow problems. Process improvement and redesign has tended to focus on correcting, streamlining, and/or completely rethinking existing business workflows to reap vast improvements in performance and significant costs savings. However, this study pioneers the use of statechart analysis to identify workflow problems during redesign.

FUTURE WORK AND CHALLENGES

We are in the process of developing a workflow design tool that incorporates the ideas we developed in this research with the notion of automating statechart analysis. The idea of developing this tool is to allow the user to develop sophisticated workflow models without having to be concerned with the underlying formalisms and algorithms we developed. In theory, the tool will automatically flag potential workflow problems for the user to aid in workflow redesign efforts. Here, we point out some of the challenges that we might face and obstacles that we must overcome to realize such a goal.

Workflow verification and validation are important topics in academia and industry alike. Given a workflow design and a specification, it is important to see if the workflow design fulfills the requirements in the specification. Similarly, the output of a workflow design needs to be validated for the workflow to be ready for production. Considering the complexity of today's business workflows, it would be extremely useful if the process of verifying and validating workflow designs could be automated, or less-ambitiously, semi-automated. However, the computer has certain limitations, particularly with algorithms, which must be understood. The Church-Turing Thesis states that Turing machines precisely capture the intuitive notions of algorithms. Turing machines, or any equivalent forms of computation, have limitations, however. A well-known problem that does not have an algorithmic solution is the halting problem of a Turing machine, which can be stated as follows: Given a Turing machine and an input text string, it is not algorithmic to determine if the Turing machine will halt on that input string. This important result in the theory of computation has serious consequences. One of the consequences of the halting problem is that it does not have a computer-based solution to determine if an algorithm possesses certain properties. Thus, in general, it is hopeless to develop an automated software to accept a workflow design as input and determine if the workflow design possesses certain nontrivial properties. However, if we put certain constraints on the given workflow, then it would be possible to develop an automated solution. Therefore, it is our job, as researchers, to determine the constraints, or the bounds, that we must impose on the workflows for such an automated solution to be feasible. This research, therefore, is a step in such a direction.

Of course we are speculating about the potential of our design tool until we can empirically validate it in the field. As such, we intend to conduct an extensive case study of MWI once our workflow design tool has been prototyped. It is hoped that additional case study iteration will reveal the tool's capabilities in a more granular manner.

We intend to further explore business workflows at MWI and other organizations to validate and extend our findings. We recently visited (May 2001) an executive at MWI who was not part of this study. The purpose of the visit was to initially verify the findings that we obtained from this study and discuss future work possibilities with MWI. The executive we interviewed was very positive about our current findings and has agreed to participate in an extension of this study.

Our next study will focus on developing the prototype and further testing our theory on a new workflow mutually agreed upon by us and the MWI contact. If we can replicate the findings we obtained in this study, it will greatly enrich context and theoretical validity. As such, we hope to build a cumulative tradition over time.

Of course we realize that in-depth case studies tend to uncover many ideas, constructs, and concepts that are unanticipated. Therefore, we will try to keep our study somewhat within the scope of theory we have already generated to enable rigorous replication.

ENDNOTES

[1] Available: http://www.omg.org

[2] See page 298 in Harel and Naamad (1996).

[3] Available: http://www.aiim.org/wfmc/mainframe.htm

REFERENCES

Aiken, A., Hellerstein, J.M., & Widom, J. (1995). Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems*, *20*(1), 3-41.

Baralis, E., Ceri, S., & Paraboschi, S. (1996). Modularization techniques for active rules design. *ACM Transactions on Database Systems*, *21*(1), 1-29.

Benbasat, I., Goldstein, D.K., & Mead, M. (1987). The case study research strategy in studies of information systems. *MIS Quarterly*, *11*(3), 368-386.

Bhargava, B. (1999). Concurrency control in database systems. *IEEE Transactions on Knowledge and Data Engineering*, *11*(1), 3-16.

Booch, G., Rumbaugh, J., & Jacobson, I. (1999). *The Unified Modeling Language: User guide*. Reading, Massachusetts: Addison-Wesley.

Brunwin, V. (1994). A survivor's guide to workflow. *Management Development Review*, *7*(4), 27-29.

Casati, F., Ceri, S., Paraboschi, S., & Pozzi, G. (1999). Specification and implementation of exceptions in workflow management systems. *ACM Transactions on Database Systems*, *24*(3), 405-451.

Casati, F., Ceri, S., Pernici, B., & Pozzi, G. (1998). Workflow evolution. *Data & Knowledge Engineering*, *24*, 211-238.

Georgakopoulos, D., Hornick, M., & Sheth, A. (1995). An overview of workflow management: From process modeling to workflow automation infrastructure. *Distributed and Parallel Databases*, *3*, 119-153.

Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, *8*, 231-274.

Harel, D. (1988). On visual formalisms. *Communications of the ACM*, *31*(5), 514-530.

Harel, D., & Naamad, A. (1996). The statemate semantics of statecharts. *ACM Transactions on Software Engineering and Methodology*, *5*(4), 293-333.

Harel, D., & Politi, M. (1998). *Modeling reactive systems with Statecharts: The STATEMATE approach*. New York: McGraw-Hill.

Object Management Group, Inc. (1999). *OMG Unified Modeling Language specification*. Version 1.3.

Paton, N.W., & Diaz, O. (1999). Active database systems. *ACM Computing Surveys*, *31*(1), 63-103.

Van der Aalst, W.M.P. (1998). The application of petri nets to workflow management. *The Journal of Circuits Systems and Computers*, *8*(1), 21-66.

Workflow Management Coalition. (1999). *Workflow management coalition – Terminology & glossary*. Document Number WFMC-TC-1101, Document Status – Issue 3.0. Available: http://www.aiim.org/wfmc/mainframe.htm

Yin, R.K. (1994). *Case study research* (2nd editon). London: Sage Publications.

Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., & Zicari, R. (1997). *Advanced database systems*. San Francisco: Morgan Kaufmann Publishers, Inc.

Chapter XIII

Framework for the Rapid Development of Modeling Environments

Akos Ledeczi, Vanderbilt University, USA

Miklos Maroti, Vanderbilt University, USA

Peter Volgyesi, Hungarian Academy of Sciences, Hungary

ABSTRACT

This chapter introduces the concepts and techniques required for developing graphical, domain-specific modeling and program synthesis environments. It argues that a fully functional modeling environment can be quickly developed for a wide variety of engineering domains using a configurable and extensible toolset with a limited set of generic concepts. The configuration is accomplished through metamodels specifying the modeling language and methodology containing all syntactic, semantic and presentation information of the domain. The authors applied this approach to several real-world systems.

INTRODUCTION

Graphical modeling environments for system development are integrated sets of modeling, model analysis, simulation and code generation tools that aid the design of systems in a particular, well-defined engineering field. These toolsets capture specifications in the form of domain models, support the design process by automated systems analysis and simulation and automatically generate, configure or integrate components of the target applications. Examples for such domain-specific environments are Rational Rose for object-oriented software development, Matlab/Simulink for signal processing and LabView for instrumentation.

Advantages of such environments are the result of their domain specificity. Domain-specific modeling methodologies enable the concise representation of essential design views, the formal expression and automated enforcement of integrity constraints and model composition that is synergistic with the design process in the domain.

While these benefits of domain-specific development environments are well understood and documented, their high cost represents a significant roadblock against their wider application. Consequently, domain specific toolsets are available commercially only for domains with large markets, where the significant initial investment is offset by high volume. For the rest of the application areas, one solution is to create configurable tools that readily provide the generic functionality of graphical development environments (creating and managing design projects, editing and combining diagrams, translating information into output formats), and let them easily be tailored to use the concepts of a given domain. Such tools can approach, albeit never fully reach, the features of an environment directly developed for a given domain. Their key advantage is that effort needed for customizing them for the domain is orders of magnitude less than developing a custom-made toolset.

Furthermore, these tools ease the development and evaluation of new or modified modeling methodologies. As we will show, the development of a fully functional modeling environment using a configurable toolset can take from hours to days, depending on the complexity of the given modeling methodology. On the other hand, the development of a custom environment from scratch is measured in man-years.

The Generic Modeling Environment (GME) (Ledeczi et al., 2001), developed at the Institute for Software Integrated Systems at Vanderbilt University (freely available at http://www.isis.vanderbilt.edu/projects/gme), is one of the more prominent configurable modeling environments. Its configuration is accomplished through metamodels specifying the modeling paradigm (modeling language, modeling methodology) of the application domain. The modeling paradigm contains all the syntactic, semantic, and presentation information regarding the domain — which concepts will be used to construct models, what relationships may exist among those concepts, how the concepts may be organized and viewed by the modeler, and rules governing the construction of models. The modeling paradigm defines the family of models that can be created using the resultant modeling environment.

Metamodeling is the primary method for specializing a GME instance. The metamodeling language is based on the UML class diagram notation. Metamodels also contain OCL constraints specifying the static semantics of the modeling language. These constraints are automatically enforced in the target GME instance. Additional methods for customizing GME include decorators, interpreters, and add-ons. Decorators are simple software components that can be attached to a GME instance. They are used for domain-specific visualization of the models. Interpreters and add-ons are external software components that interface with GME and provide additional domain-specific functionality including, but not limited to, code generation.

The rest of this chapter is organized as follows. In the next several sections the different methods for providing native support for different modeling methodologies are described in detail. Then examples are presented that illustrate these techniques. Finally, we compare two other well-known configurable modeling environments to our approach and present our conclusions.

METAMODELING

The metamodeling language is not used for defining domain models, but rather for defining domain-modeling languages. Thus, "sentences" in the meta-language define specific domain languages, while "sentences" of the domain language define specific systems. GME follows the standard four-layer modeling architecture.

The GME metamodeling methodology is implemented with GME itself. The metamodeling language is just another domain language. The metaspecifications that configure the GME are generated by the metamodeling interpreter from the metamodels. The metamodeling language itself is generated by the same interpreter when translating the meta-metamodels.

At the metamodeling level, GME provides generic modeling primitives that assist an environment designer in the specification of new modeling environments. These concepts are directly supported by the framework as stereotypes of the specific classes. Elementary types that do not contain other objects are defined as *atoms*, while *models* are composite classes. Associations between these classes are modeled using the *connection* primitive that is visualized by the modeling tool as a line between the objects. Connections can only express relationships between objects at the same hierarchy level or one level deeper with the help of ports on composite models. *References* help to overcome this limitation by enabling the user to associate objects in different model hierarchies. A reference always refers to exactly one object, which can be of any kind except connection; this establishes a relationship between the model that contains the reference and the referred object. Connections and references model relationships between at most two objects. *Sets* can be used to specify the relationship among a group of objects. Atoms, models, connections, references and sets are the first class objects (*FCO*) of the modeling framework.

The language designer can assign different attributes to first class objects. Attributes are values of predefined simple types, such as integer, string, boolean and enumeration. The meta-attribute defines the name, the value type and the default value of the attribute. The attribute value of an instantiated object is user-changeable at the modeling level.

The framework provides various techniques for managing the complexity of large-scale models; the most notable concept is the introduction of aspects enabling the domain users to focus on selected parts of a design. At the metamodeling level, a set of aspects are assigned to every composite type and the visibility of each contained class can be defined for a given aspect. This powerful construct assisting the modeler in separating the concerns of multi-perspectives is similar to views in the world of rational databases.

The modeling concepts above can be used only if the environment designer precisely defined them in the metamodel of the paradigm. In our experience, it is often necessary to associate information chunks without real semantic meaning or strict syntax to objects. This includes, for example, the specification of visualization information, such as color, style and icon for an object. Therefore we have added an extensible storage to every FCO, called the registry. The registry is a tree data structure containing the auxiliary data in the nodes of the tree. The shape and node names of the tree are not fixed, in order to provide extensibility for external tools.

The metamodeling environment provides a powerful set of inheritance operators to describe specialization and to support metamodel composition. The common inheritance operator implements the semantics of the standard UML specialization; thus the specialized child type inherits all of the parent's attributes and can participate in any association the parent can participate in. Two additional operators are available to provide finer-grained control over the inheritance relation. These were specifically designed to support metamodel composition. Implementation inheritance propagates all of the parent's attributes, but only the containment association — where the parent functions as the container — to the child type. No other associations are inherited in this case. Interface inheritance allows no attribute inheritance, but does allow full association inheritance, with one exception: containment relations where the parent functions as the container are not inherited. Note that the union of the two special inheritance operators gives the common inheritance, and their intersection is null.

Just as the reusability of domain models from application to application is essential, the reusability of meta-models from domain to domain is also an important consideration. In GME, a library of meta-models of important sub-domains is made available to the meta-modeler, who then can pick and choose from them, extend and compose them together to specify new domain languages. The extension and composition mechanisms must not modify the original meta-models for two reasons. First, changes in the meta-model libraries, reflecting a better understanding of the given domain, for example, should propagate to the meta-models that utilize them. Second, by precisely specifying the extension and composition rules, using inheritance and equivalence operators, for instance, models specified in the original domain language can be automatically translated to comply with the new, extended and composed, modeling language. This is a simple and elegant solution to the well-known model migration problem. (For more detail on metamodel composition please see Ledeczi, Nordstrom, Karsai, Volgyesi & Maroti, 2001).

TYPES AND INSTANCES

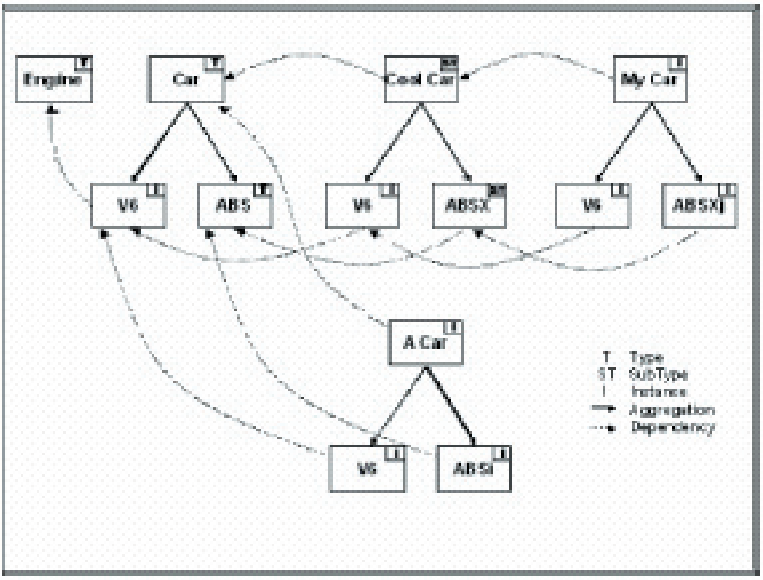
Model reuse and tools for information maintenance between similar models is a natural requirement in large-scale models or where model composition is heavily used. The provided solutions in GME — types and instances — resemble those of object-oriented programming languages. The only significant difference is that in GME, model types are similar in appearance to model instances; they too are graphical, have attributes and contain parts.

By default, a model created from scratch — based on a meta-type — is a type. A subtype or an instance of a model can be created with a simple operation, and both will depend on the type they are created from. There is one significant rule that differentiates subtypes from instances. New parts are allowed in a subtype, but not in an instance. Otherwise, contained children can be renamed, set membership can be changed and references can be redirected in both subtypes and instances. However, objects in the containment hierarchy cannot be removed in either subtypes or instances.

The advantage of using types is clear: any modification in a type model propagates down the inheritance hierarchy. For example, if a part is deleted in a type, the same part will be automatically removed in all of its instances and subtypes — even in instances of the subtypes — all the way down the inheritance tree.

Types can contain other types as well as instances as parts. The mixture of aggregation and type inheritance introduces another kind of relationship between objects. This is best illustrated through an example. In Figure 1, there are two root type models: the *Engine* and the *Car*. The car contains an instance of an engine, *V6*, and an *ABS* type model. V6 is an instance of the Engine; this relationship is indicated by the dashed line. Aggregation is indicated by solid lines.

Figure 1: Model dependency chains



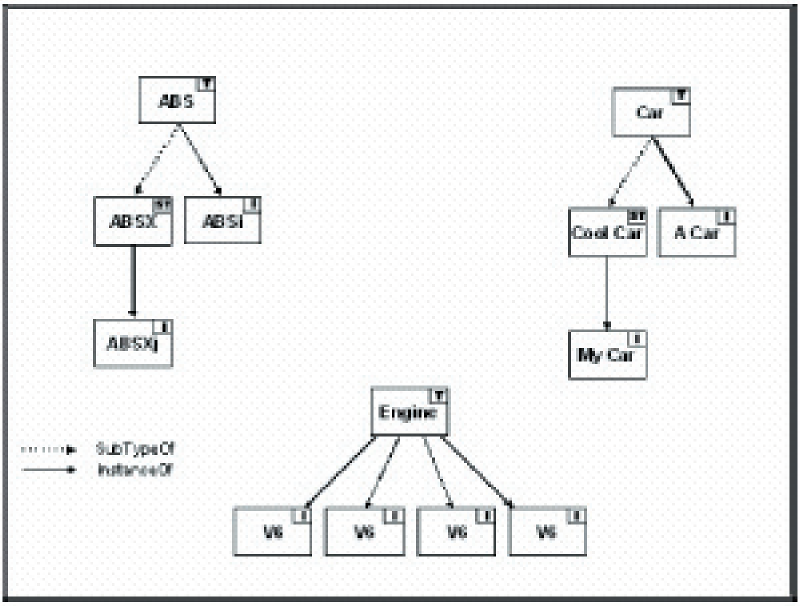
When a subtype of the Car is created, e.g., *Cool Car* above, we indirectly create another instance of the Engine (V6) and a subtype of the ABS type. This is the expected behavior, as a subtype without any modification should look exactly like its base type. Notice the arrow that points from V6 in Cool Car to V6 in Car. Both of these are instances, but there is dependency between the two objects. If we modify V6 in Car, V6 in Cool Car should also be modified automatically for the same reason: If we do not modify Cool Car it should always look like Car itself. The same logic applies if we create an instance of Cool Car – *My Car* in Figure 1. It introduces a dependency between V6 in My Car and V6 in Cool Car. As the figure shows, this forms a dependency chain from V6 in My Car through V6 in Cool Car and V6 in Car all the way to the Engine type model.

An interesting situation arises if we modify V6 in Cool Car by changing an attribute. The question is whether an attribute change in V6 in Car should propagate down to V6 in Cool Car and below. Since the attribute has been overridden, the dependency chain is broken up with respect to that attribute. However, if the same attribute is changed in V6 in Cool Car, that should propagate down to V6 in My Car unless it has already been overridden there. Figure 2 shows the same set of models, but only from the pure type inheritance perspective.

The real strength of types and instances can be exploited with the use of model libraries. Based on predefined and verified models residing in these libraries, the modeler is able to create new instances in his or her project without losing the connection to the prototype model; thus further enhancements and corrections in the original model can be easily propagated to all of its subtypes and instances automatically.

The type and instance feature of GME is sometimes confused with the inheritance relation in the meta-modeling environment. While the meta-inheritance implements the

Figure 2: Type inheritance hierarchy



semantic of the UML specialization — and is handled by the meta-interpreter, types and instances are supported by the modeling engine, and their primary goal is to support model reuse independently of the meta-model.

CONSTRAINT MANAGEMENT

One can consider the UML class diagram-based meta-model as syntax specifications. It determines what concepts are used in the modeling language and specifies relations and attributes. It does not say much about what constitutes a correct model. We use the Object Constraint Language (OCL) for the specification of the static semantics of the modeling language. Constraints are attached to the meta-models specifying well-formedness rules.

In addition to the OCL expression, a GME constraint has a priority attached to it specifying the action the built-in constraint manager should take upon its violation. The highest priority results in an error message and the abortion of the current transaction. Lower priority violations only cause warning messages.

Constraints can be attached to editing events specifying when they must be checked. For example, the "on connect" event should be specified for a constraint that restricts the kind or number of connections a given model can be attached to. Furthermore, all constraints can be checked on-demand at any time.

Certain pieces of information captured in the meta-model cannot be compiled directly into the paradigm configuration because of the limitations of that format. In such cases, such as the multiplicity information of containment, membership and connection cardinality definitions, the meta-interpreter automatically generates OCL constraints.

Complex and reusable constraints can be defined in constraint functions that can be called from constraints or other functions. They even support recursion. Function parameters enable the constraint developer to formalize reoccurring definitions in a generic form.

One of the most powerful facilities of the constraint system is the browser that provides an interactive window to the constraint database. The browser displays the definition, state and other attributes of each available constraint. Selected constraints can be evaluated on demand or can be disabled temporarily by the user. In addition to the constraints defined in the meta-model, the model builder is able to add and remove custom constraints at the modeling level.

The constraint debugger assists the modeler in discovering erroneous constraint definitions. The stack of evaluated expressions, along with the current values of all variables, are displayed in the debugger. The evaluation tracking facility can be turned off when the designer is confident in the correctness of the constraint definitions.

VISUALIZATION

The modeling framework provides different kinds of graphical interfaces to the model database. The primary display area represents models as separate windows showing contained objects as icons and lines. The physical position of these objects can be arranged arbitrarily and independently in each aspect of the model. The connection paths are controlled by a powerful real-time autorouter. The object positions in different aspects can be selectively synchronized to help clean up large models.

The model browser uses a different visualization approach, displaying the model hierarchy as a tree where non-leaf (composite) nodes can be collapsed or expanded on demand. While this method provides less control and information on a specific node, it reveals the whole model hierarchy.

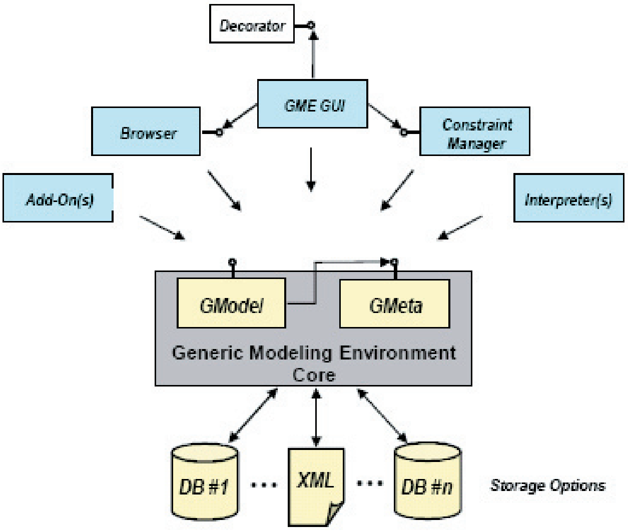
The third interface displays model information in tabular format, similar to a spreadsheet, which supports batch editing and consistency checking nicely.

Finally, it is the main editing window where most of the model creation and modification takes place. Model visualization can be customized to fit the target modeling methodology. All object drawing and mouse handling operations are assigned to a software component, called a decorator, outside of the regular user interface. Whenever these operations need to be executed the GME editor calls the decorator registered for the current modeling language through a predefined interface. GME comes with a default decorator and some samples. Any modeling paradigm can have a custom decorator implementing domain-specific visualization. Decorators have full access to the model database to be able to provide context-based visualization for the decorated objects. The graphical framework may send update requests to a specific decorator with a given frequency, if animated visualization is requested.

EXTENSIBILITY

GME was designed with extensibility as one of the most important goals. The tool has a modular component architecture, shown in Figure 3. At the bottom, different storage formats, ranging from relational databases through a fast proprietary binary file format to XML, are supported. Two key components of the GME are GMeta and GModel. The GMeta component defines the modeling paradigm, while GModel implements the GME modeling

Figure 3: GME architecture



concepts for the given paradigm. GMeta configures itself by reading the meta-specifications (generated from the metamodels), while GModel uses the services of GMeta for self-configuration. The GModel component exposes its services through a set of public interfaces as well. The architecture is based on Microsoft COM technology.

The user interacts with the components at the top of the architecture: the GME User Interface, the Model Browser, the Constraint Manager, Interpreters and Add-ons. Add-ons are event-driven interpreters. The GModel component exposes a set of events, such as "Object Deleted" or "Attribute Changed", etc. External components can register to receive some or all of these events. They are automatically invoked by GModel when the events occur. Add-ons are extremely useful for extending the capabilities of the GME User Interface, for example. When a particular domain calls for some special operations, these can be supported by add-ons without modifying any GME components.

Since the event dispatching mechanism is a vital part of the architecture, its performance has a significant impact on the usability of the overall framework. All external components own a so-called territory that keeps track of all objects that the component may be interested in. This repository is automatically maintained based on the object references that GModel ever handed over to the component. A specific event will be propagated to the component only if the affected object is in the component's territory. This technique reduces the number of redundant event messages dramatically.

The performance and reliability of the overall system is further improved with the help of transactions. Model operations — even read-only actions — on the GModel level must be encapsulated in transactions. Components are receiving events when a transaction begins and finishes; thus they are able to aggregate multiple changes and react to those changes at the end of the whole transaction. In the case of a read-only transaction, they may discard all notifications.

The framework keeps track of all registered external components and integrates them into the user interface. Add-ons are automatically started when a project in a supported paradigm is opened. Interpreters are integrated into the menus and toolbars of the user interface after successful registration. Information on the registered components is retained across invocations of the tool.

COM technology enables the seamless integration of additional components. Moreover, components can be implemented using any programming language that supports COM, such as C++, Visual Basic, Python, Java or C#.

The different standard technologies applied throughout the environment, such as UML, OCL, XML, and COM ensure maximum flexibility. The modeling tool also provides a C++ programming framework along with a code wizard to help in developing external components without understanding COM or working on infrastructural code. The framework, called the Builder Object Framework, is a hierarchy of classes that represents and mirrors the model database in the form of C++ objects. It enables the developer to focus on the domain-specific part of the program immediately, thus implementing simple but useful components that can take as little time as a few hours.

EXAMPLE

As a first example, consider the meta-modeling language itself. Figure 4 shows the meta-model of a simple hierarchical finite state machine modeling paradigm (HFSM) in GME configured for meta-modeling. In the lower right corner of the GME window you can see the currently active modeling language; in this case it is MetaGME. The window in the lower left corner is the partbrowser. It contains the kind of parts that can be inserted in the current aspect of the currently open model. The tabs in the bottom show all available aspects. For the meta-modeling language these are: ClassDiagram, Visualization, Constraints and Attributes. Aspects help manage model complexity by separating orthogonal concerns. Aspects are captured in the Visualization aspect of the meta-model (not shown).

Figure 4: HFSM metamodel

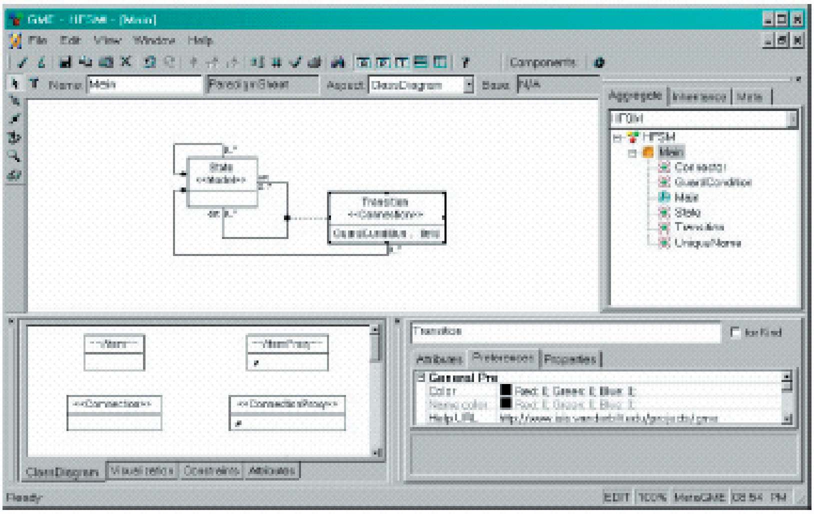
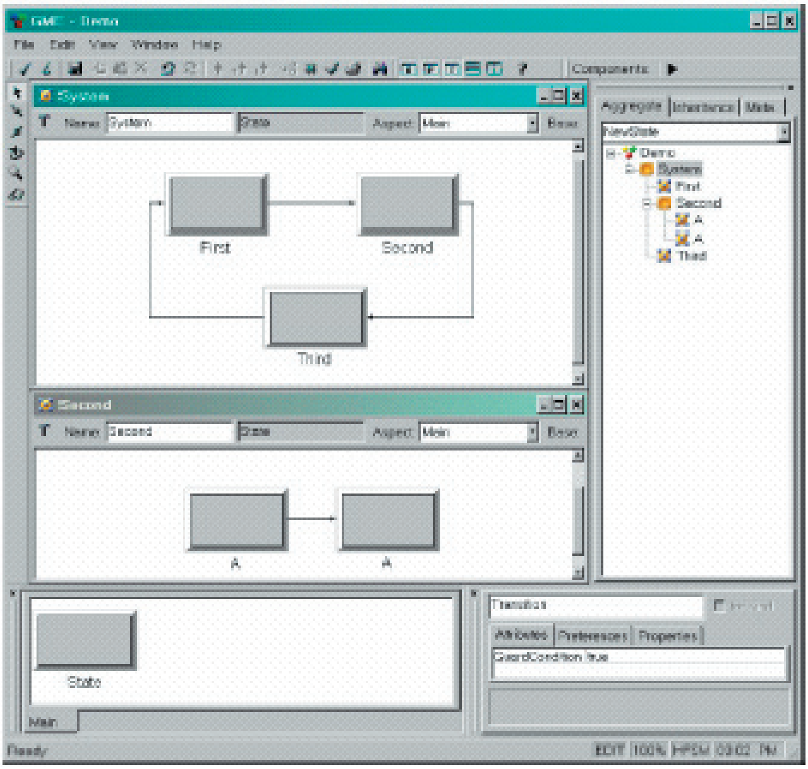


Figure 5: Example HFSM model



The window in the lower right corner shows the attributes, preferences and properties of the selected object or objects. The window on the top right is the model browser. It shows the hierarchy of the whole project. Notice that in this case it shows parts GuardCondition, Main or UniqueName, which are not shown in the main window. The reason is that those parts are shown in different aspects of the Main model. GuardCondition is an attribute of Transition captured in the Attributes aspect, Main is the single aspect in our HFSM modeling language modeled in the Visualization aspect, while UniqueName is a constraint specified in the Constraint aspect. This constraint specifies that no two substates of a state can have the same name:

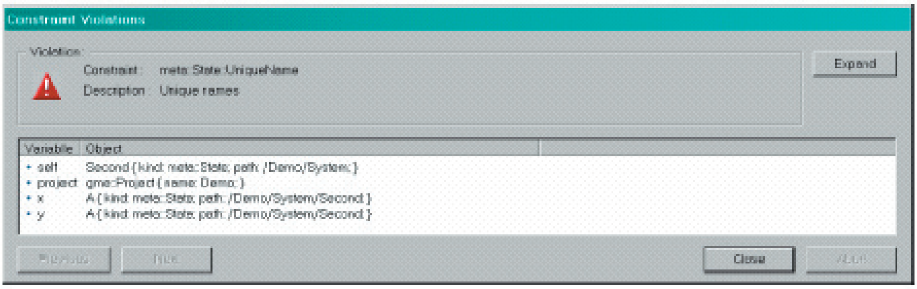
self.parts(State)->forAll(x, y | x.name = y.name implies x = y)

For this particular constraint the Close Model event seems to be the best candidate for enforcement. Whenever the parent model is closed the constraint will be checked.

The meta-modeling environment has its own decorator. It is capable of visualizing the UML class diagram notation. It displays the names, stereotypes and attributes of all classes. It shrinks or expands the box to fit the displayed text. This particular decorator is about 1500 lines of C++ code, most of which is the baseline code shared among all decorators.

The meta-modeling environment has a good example for an add-on. OCL syntax checking is a very specific functionality that is not part of the baseline GME program.

Figure 6: Constraint violation



Constraint : meta: State UniqueName
Description : Unique names

Variable	Object
self	Second { kind: meta::State; path: /Demo/System; }
project	gme::Project { name: Demo; }
x	A { kind: meta::State; path: /Demo/System/Second }
y	A { kind: meta::State; path: /Demo/System/Second }

However, its function is very important to meta-modeling; the user does not want to wait to catch syntax errors in her constraints until a constraint is checked in the target environment. Therefore, we provide an add-on to the meta-modeling environment that checks the OCL syntax of the constraint that is currently being edited. The OCL expression is a textual attribute of the constraint object. The OCL syntax checker add-on is registered to catch attribute change events. Whenever it is fired it parses the OCL expression and provides immediate feedback to the user.

As do most meaningful modeling paradigms, the meta-modeling environment has its own interpreter. It parses all the class diagrams (in the HFSM case there is only a single one) and generates an XML representation of the modeling language. GME can read this file and configure itself to support the new language. Figure 5 shows an example model captured in the resulting HFSM modeling environment.

The name of the modeling language is shown in the lower right corner again. Notice that the only part shown in the part browser is State and the only aspect is Main. The attribute window shows the GuardCondition attribute of one of the transitions. Our simple HFSM environment uses the standard, built-in decorator that is able to display boxes, icons, names, etc. Notice that the state Second has two substates with the same name "A". When trying to close the model or explicitly requesting constraint checking, the violation is caught by the constraint manager. The error message displayed is shown in Figure 6.

PRACTICAL APPLICATIONS

Three practical applications of GME are presented below:

MILAN, the Model-based Integrated simuLAtioN framework, is a GME-based extensible environment that facilitates rapid evaluation of different performance metrics, such as power, latency and throughput, at multiple levels of granularity, of a large class of embedded systems by seamlessly integrating different widely-used simulators into a unified environment (Ledeczi, Davis, Neema & Agrawal, 2003). The MILAN framework is aimed at the design of embedded high-performance computing platforms, of System-on-Chip (SoC) architectures for embedded systems, and for the hardware/software co-design of heterogeneous systems.

MILAN provides an integrated environment where existing development and analysis tools, primarily simulators, can work seamlessly together. MILAN defines an integrated data model that captures the shared semantics of all the tools integrated and a bi-directional semantic translator for each of them. This is an elegant solution to tool integration that also avoids the scalability problem associated with pair-wise translation; i.e., the integration of a new tool requires only a single new translator (and possibly modifications to existing ones) and not N, i.e., the number of integrated tools. This is a key advantage since one of the design goals of MILAN is to provide an open environment so that users can integrate additional tools on their own.

The complex modeling language allows for the specification of the desired application functionality in the form of an extended dataflow representation with strong data-typing and parametric modeling, provides the means to specify the available hardware resources and enables the user to define mapping information between the two. Finally, application requirements can also be captured by explicit constraints in the models. Instead of specifying a point solution, however, MILAN enables capturing the whole design-space of the application. At any point in the hierarchical dataflow, explicit design or implementation alternatives can be specified. For example, different algorithm choices optimized for speed, memory requirements or power consumption can be captured this way or optimized implementations can be provided for different hardware targets. Similarly, multiple hardware resource options can also be supplied. Finally, hardware/software allocation need not be fully specified. These techniques make it possible to describe a large — potentially exponential — set of solutions forming the design space of the application. Our symbolic design-space exploration and pruning technology rapidly narrows it down to the subset that satisfies all requirements of the system that are captured as constraints (Neema, 2001).

Tools currently integrated into the MILAN framework include such functional simulators as Matlab, SystemC and ActiveHDL (a VHDL simulator) and a high-level performance estimator, HiPerE (Mohatny & Prassana, 2002). These tools are not aware of which parts of the system are to be implemented in hardware and which parts in software. This makes MILAN a true system-level hardware/software co-design environment. Of course, lower-level integrated tools, such as such cycle-accurate simulators as SimpleScalar, are already tied to a specific hardware technology. Note that the different tools only communicate through the integrated system models. This eliminates the need for each tool to be interfaced directly to all others.

SSPF, a predecessor of GME was used to create the Saturn Site Production Flow (SSPF) system that monitors the car manufacturing process at GM's Saturn Corporation, providing key production measures to managers in real-time (Long, Misra & Sztipanovits, 1998). The system models describe the manufacturing processes down to the machine level, the buffers between the processes (e.g., conveyor belts), the instrumentation (i.e., PLCs), and how the information is to be presented to the user. The interpreters generate various configuration files and SQL database schema to configure the SSPF client-server application. The program gathers the production information, stores it in a real-time database and makes it available to any user in the plant.

GRATIS, a graphical development environment for TinyOS, provides an intuitive visual interface and automatic code generation capability for the development of TinyOS-based sensor network applications (Volgyesi & Ledeczi, 2002). With the original TinyOS tools (Hill, 2000) working with textual configuration files while developing non-trivial applications could quickly become an error-prone and tedious process. Function-like entities have two or more names in the final application; this characteristic is inherent in the flexible design, enabling the creation of countless different applications without touching the implementation of the individual components. However, as a side effect, it has notable impact on the maintainability of the applications. GRATIS replaces the textual representation of the interface and configuration specifications. Even with a simple application, a more expressive representation of components and interconnections between them can help design better applications and increase their readability. With more sophisticated components and especially with hierarchical composition, this becomes an absolute requirement. There are cases where components might impose additional complex restrictions on their use — like mutual exclusion or maximum fan-out — in addition to the normal rules of composition in TinyOS. These additional requirements can be easily captured by the constraint language provided by the GME modeling framework.

Since all practical applications use system components from the TinyOS distribution, GRATIS also provides a mapping from the existing large code base to the graphical environment. Therefore, the interpreter not only generates text files from graphical models, but it is also capable of parsing existing files and building the corresponding GME models from them. The main use of this parsing feature is to automatically generate the graphical equivalent of the TinyOS system components and to provide them as a library to the user in the GRATIS environment. An indisputable benefit of the parsing and model building process is an exhaustive testing, since the parser — with the help of the predefined constraints in the meta-model — builds and validates all components and applications found in the source tree. Since scripting languages are generally superior to compiled languages in the field of text processing, we have implemented GRATIS using GME and the Python language exclusively, which also demonstrates an extension alternative to our C++ interpreter framework.

Other experimental modeling languages have also been implemented to describe not only the type requirements, but temporal dependencies and the implementation details also. These languages comprise our further work to understand compatibility and composability issues better in the field of embedded systems. GME proves to be an efficient tool and approach to build such environments.

COMPARISON TO OTHER TOOLS

In terms of supported features, maturity, and the number of real-world applications, three configurable environments stand out: Dome by Honeywell Laboratories (Honeywell, 2000), MetaEdit+ by MetaCASE Consulting of Finland (MetaCase, 2000) and our own Generic Modeling Environment (GME) (Institute for Software Integrated Systems, 2002). The four key areas that enable true support for widely different modeling methodologies are meta-modeling, constraint management, visualization and extensibility.

Meta-modeling: Meta-modeling may be regarded as just another type of modeling; therefore, Dome and GME use the tools themselves to implement this functionality. MetaEdit+ has a more conservative approach; a series of dialog boxes are used to specify the meta-model in a non-graphical way. Meta-models typically evolve while being used, and modifications in the meta-model often break the validity of models. These concerns are just partially handled in Dome and MetaEdit+. GME is the only tool that demonstrates a strict discipline: meta-models are versioned, and new versions of meta-models do not affect existing models until they are explicitly upgraded to the new version. Such an upgrade implies extensive validity checking. This is somewhat cumbersome, but essential for warranting the correctness of models, especially if they are beyond the usual demo application size.

GME is the only environment that provides support for meta-model composition and meta-model libraries.

Constraint management: Dome and MetaEdit+ have some built-in support for certain types of frequently used constraints. GME, on the other hand, has a full-featured constraint manager supporting OCL, a standard constraint language. Constraints can also be associated with editing events and priorities, making constraint management interactive and flexible.

Visualization: While MetaEdit+ provides an impressive built-in symbol editor, GME and Dome provide only the choice of simple built-in symbols and bitmap files provided by the user, and rely on user-defined drawing routines for more complex visualization. While this user-defined visualization can be very powerful and flexible, their implementation obviously requires some traditional programming skills from the user.

Extensibility: The interface to Dome models is primarily through its Alter language, a Scheme variant. MetaEdit+ only supports a proprietary scripting language to access the models. On the other hand, the component-based architecture of GME makes it easily extensible. Meta and model information are all available through public COM interfaces. Events are also exposed through COM. When any component makes a change to the models, all other interested components are notified. The toolset can be extended using any programming language that supports COM (e.g., C++, VB, Python). Furthermore, GME supports XML export/import for both model and meta-information.

CONCLUSIONS

We presented GME, a framework that enables the rapid development of modeling environments. Its powerful meta-modeling capabilities make it possible to create a full-featured modeling environment in hours. For example, an experienced user could easily create the simple HFSM environment presented above as an example in well under an hour. Hence, GME supports the rapid design of modeling languages enabling immediate hands-on experience with the language. It supports an iterative design method of modeling methodologies; users can quickly evolve their language design by iteratively modifying the corresponding meta-model. When the modeling methodology becomes satisfactory then the effort to create decorators, add-ons and interpreters is much better justified. Note, however, that typical domain-specific components for GME are relatively small; hence the effort to create the additional software modules is not prohibitive, even for small projects. For example, an HFSM simulator that animates the automaton within the GME user interface was written by a graduate student in a couple of weeks.

The HFSM example presented here is very simplistic for clarity. In our experience, GME scales very well. The modeling language of MILAN, for example, has hundreds of modeling concepts. Both MILAN and GRATIS support large model databases. Other large-scale, real-world applications of the technology are presented in Ledeczi et al. (2001).

The research in configurable modeling environments, meta-modeling methodologies and model visualization continues at our institute. GME is just a reflection of the current state-of-the-art of our research. New versions of the software are regularly released multiple times a year. Ongoing work includes research on generative modeling, automatic interpreter generation using graph transformations (Agrawal & Karsai, 2003) and integrating GME into the popular Eclipse framework (Eclipse.org Consortium, 2001).

ACKNOWLEDGMENTS

The authors would like to express their gratitude to DARPA and the Boeing Company for the generous sponsorship of the research described here.

REFERENCES

Agrawal, A., & Karsai, G. (2003). A UML-based graph transformation approach for implementing domain-specific model transformations. *International Journal on Software and Systems Modeling* (Submitted).

Eclipse.org Consortium. (2001). *Eclipse official web site.* Retrieved from: http://eclipse.org

Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., & Pister, K. (2000). System architecture directions for networked sensors. *Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) 2000,* Cambridge, MA, USA.

Honeywell. (2000). *Dome official web site.* Retrieved from: http://www.htc.honeywell.com/dome

Institute for Software Integrated Systems. (2002). *GME official web site.* Retrieved from: www.isis.vanderbilt.edu/projects/gme

Ledeczi, A., Bakay, A., Maroti, M., Volgyesi, P., Nordstrom, G., Sprinkle, J., & Karsai, G. (2001). Com*posing domain-specific design Environments.* IEEE Computer, *34*(11), 44-51.

Ledeczi, A., Davis, J., Neema, S., & Agrawal, A. (2003). Modeling methodology for integrated simulation of embedded systems. *ACM Transactions on Modeling and Computer Simulation, 13*(1), 82-103.

Ledeczi, A., Nordstrom, G., Karsai, G., Volgyesi, P., & Maroti, M. (2001). On metamodel composition. *IEEE Conference on Control Applications 2001,* Mexico City, Mexico, CD-ROM.

Long, E., Misra, A., & Sztipanovits, J. (1998). Increasing productivity at Saturn. *IEEE Computer, 31*(8), 35-43.

MetaCase Consulting. (2000). *MetaEdit+ official web site.* Retrieved from: www.metacase.com

Mohatny, S., & Prassana, V.K. (2002). *HiPerE: A framework for rapid system level power and performance estimation of embedded applications on SoC/SoP Architectures.* Design, Automation, and Test in Europe.

Neema, S. (2001). *System level synthesis of adaptive computing systems.* Ph.D. Dissertation, Vanderbilt University, Department of Electrical and Computer Engineering.

Volgyesi, P., & Ledeczi, A. (2002). Component-based development of networked embedded applications. *28th Euromicro Conference, Component-Based Software Engineering Track,* Dortmund, Germany.

Chapter XIV

Federated Process Framework for Transparent Process Monitoring in Business Process Outsourcing

Kyoung-Il Bae, IBM Business Consulting Services, Korea

Soon-Young Huh, Korea Advanced Institute of Science and Technology, Korea

ABSTRACT

Process information sharing is a beneficial tool through which a company can monitor and control its outsourced business process transparently, as if the outsourced business process is performed locally. However, autonomy and agility of insourcing companies providing outsourcing services have placed limitations in the development of process information sharing, which the previous research has not satisfactorily addressed. This chapter proposes a federated process framework and its system architecture that provide a conceptual design for effective implementation of process information sharing supporting the autonomy and agility of the insourcing companies. First, in terms of autonomy, the federated process framework supports a flexible sharing policy to control the amount of shared data so that the framework can be applied to a wide variety of practical situations, from loosely-coupled cases to tightly-coupled cases. Second, in terms of agility, the system architecture based on the federated process framework supports the entire life cycle of business process outsourcing by allowing sufficient adaptability to the changes of business environments. We develop the framework using an object-oriented database and Extensible Markup Language to accommodate all the constructs and their interactions within object-oriented message exchange model in a distributed computing environment.

INTRODUCTION

Researchers and business consulting firms increasingly emphasize the importance of effective outsourcing in terms of revenue increase and cost reduction (Gartner, 2002; Hafeez, 2002). To accommodate such goals effectively, the outsourcing target requires including the business processes as well as IT infrastructure (Berfield, 2002), and such a type of an outsourcing model is called business process outsourcing. Specifically, by outsourcing a part of local business processes with its supporting systems, a company could reduce the cost related to human resource and system development, while focusing on its core business without bothering about the outsourced part. However, Gartner's recent survey (2003) of corporate executives across Asia/Pacific shows that the fear of loss of control is one of the most prominent reasons for not outsourcing. To remove the concern for the control, a company should be able to monitor its outsourced business process transparently, as if the outsourced one is executed internally. By enabling this transparent process monitoring, the company could streamline and coordinate the internally-executed business processes with the outsourced one in its value chain. The key technique for achieving the transparent process monitoring is process information sharing (Alonso, 1999; Ball, 2002; Georgakopoulos, 1999). Process information sharing means that participating organizations in business process outsourcing provide visibility of their internal process information to each other in order to enhance process monitoring capabilities.

In the example of an online store case, most online stores outsource their delivery operations to external transportation companies for the purpose of cost efficiency, and then focus on their core business functions, such as marketing and order processing. Then, if an on-line store receives detailed delivery process information from its collaborating transportation company, it can effectively carry out and monitor full steps of order fulfillment processes, from order capturing through picking and packing, and finally to product delivery. In terms of the customer satisfaction, such process information sharing allows the online store to provide customers an extended order tracking service to monitor the overall process status for their orders. In terms of the service quality control, the online store can check the quality of the transportation company's services by monitoring the status of the delivery process.

Most of the previous research on process information sharing has focused on demonstrating such benefits (Ball, 2002; Lee, 1997; D'Amours, 1999; Zhou, 1998) and providing appropriate underlying system architecture or design for process information sharing (Alonso, 1999; Georgakopoulos, 1999; Kuechler, 2001; Mori, 1999; Workflow Management Coalition, 2000). However, research efforts considering the issues caused from the autonomy and agility that are the inherent properties of modern organizations are few, even though these issues make it difficult to accommodate process information sharing in many real situations.

Motivation and Research Questions

Autonomy means that an outsourcing service provider, called an insourcing company, can decide whether to and how much of its local data to share with an outsourcing service requester, called an outsourcing company. In spite of outsourcing agreements, most insourcing companies are usually reluctant to expose their core business information on their internal business logic and full process status to outsourcing companies (Bolcer, 1999; Georgakopoulos, 1999; Merz, 1999). Such unwillingness often conflicts with the need to share data, and therefore these two conflicting factors determine the degree of autonomy for the amount of shared data. When the degree of autonomy is determined, the insourcing and outsourcing companies establish a sharing policy on the amount of shared data in their outsourcing contract. Regarding the autonomy issue, existing studies on process information sharing have addressed little about the mechanisms to establish a sharing policy and how to control the amount of shared process information according to the sharing policy. Specifically, the autonomy problem can be further articulated by the following questions:

(i) How can the amount of shared process information be systematically represented as a sharing policy concerning the determined degree of autonomy between insourcing and outsourcing companies?
(ii) How can the system for process information sharing restrict and control the amount of shared process information according to the sharing policy, while accommodating seamless process information sharing?

On the other hand, agility means that companies constantly refine their business strategies and information systems in order to meet both customer needs and environmental changes or to take new opportunities (Goranson, 1999; Ramamurthy, 2003). For example, an insourcing company needs to refine its business to meet the changing needs of outsourcing companies and to widen its customer base. It is well recognized that agility is one of core competencies in the fast-changing modern business environment (Scott-Morton, 1994). In terms of process information sharing, we specifically consider that an insourcing company adopts either of the following two methods in response to the changes of business environments: First, the internal business process is amended to meet newly proposed market or organizational constraints or to improve business operations from the advent of new technologies (Casati, 1998; Mangan, 2002). Second, the business relationships with outsourcing companies are modified due to the changes of mutual dependencies or external environments, and corresponding sharing policies are changed accordingly (Bakos, 1998; Chircu, 2000). These methods change the schema and amount of shared process information; accordingly, the system for process information sharing should also be modified and recompiled to reflect the changes incurred in the system procedures for data sharing. The more frequently such a change arises, the more seriously the maintenance cost is to be considered. Such an agility problem can be further delineated by the following questions:

(i) How can the system for process information sharing adapt itself to the change of internal business process or sharing policy of an insourcing company without causing serious maintenance cost?
(ii) In developing the system, which system components are suitable or necessary to accommodate such adaptability?

Research Objective and Adopted Technologies

The main objective of this chapter is to propose a federated process framework as a conceptual design to support effective implementation of process information sharing between insourcing and outsourcing companies, while resolving the autonomy and agility problems in the outsourcing environment. In the actual development of process information sharing, implementation issues such as security control and data conflict resolution should be under consideration. However, with a conceptual perspective, this chapter focuses on providing answers to the foregoing four research questions related to the autonomy and agility problems.

We employ a federated database system approach (Heimbigner, 1985; Sheth, 1990) and extend it with an Internet-based system architecture. As base technologies, an Object-oriented Database (OODB) (Kim, 1995) and Extensible Markup Language (XML) (Carter, 2000) are adopted. The OODB allows flexible and natural modeling of business processes and sharing policies while ensuring efficient object persistency with transaction capabilities. Furthermore, classes constituting the object data model entirely or partially generate application programs with object-oriented programming languages. The object data model therefore serves as building blocks both to design a persistent database schema in the OODB and to develop application programs. The XML has been widely adopted as a standard language for communication among distributed software programs (Herring, 2001; Sundaram, 2001; UN/CEFACT, 2001) due to its flexibility and Internet-based architecture. In addition, it provides extensive data manipulation capabilities in the distributed computing environment, such as data integration and granular update from multiple data sources (Böhm, 2000). Because of the resurging popularity and data manipulation capabilities, the XML is adopted in the study as the most appropriate tool for vendor and platform-independent data sharing in the distributed environment. To evaluate the effectiveness of the federated process framework, a prototype system has been implemented on a commercial OODB Management System called OBJECTSTORE (Progress Software, 2003) with the JAVA programming language (Arnold, 2000).

The chapter is organized as follows: The next section introduces the basic concepts of an inter-organizational process model with an order fulfillment example. Then, the chapter describes the federated process framework and presents detailed data designs, including an object data model and XML document structure. This is followed with a presentaation of an Internet-based system architecture with its prototype system. The chapter then discusses the results, and the final section summarizes contributions of this chapter and also provides future research directions.

INTER-ORGANIZATIONAL PROCESS MODEL

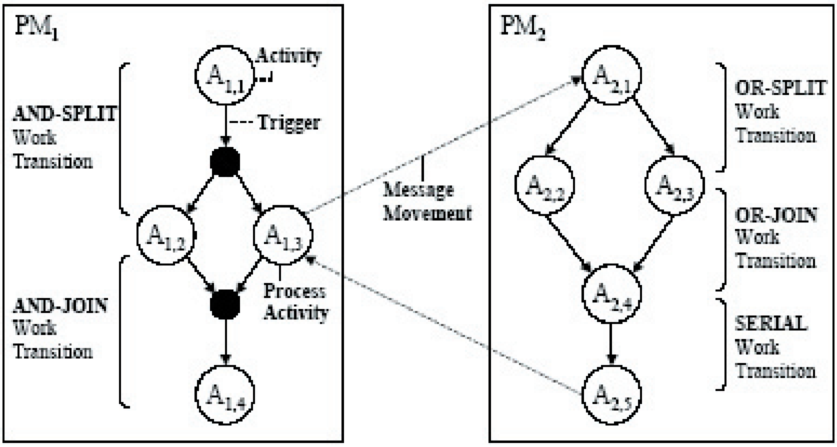
This section briefly introduces basic concepts of an inter-organizational process model and looks at an order fulfillment example to provide the perspective on the research problems of the chapter.

Basic Concepts of an Inter-organizational Process Model

A **process model** is the structure of a business process and all internal work sequences in various conditions. In terms of an object-oriented paradigm, it is a template from which a business process instance is created and executed. The **process status** specifies the status of a particular business process instance. An **inter-organizational process model** is composed of multiple collaborating local process models with their interactions. The process status on the inter-organizational process model, called a **global process status**, is obtained by merging multiple process statuses from the local process models.

Most previous studies consider an activity and work transition as the most fundamental constructs to represent a process model (Cichocki, 1998; Leymann, 2000; Workflow

Figure 1: Graphical representation of an inter-organizational process model



Management Coalition, 1998). The **activity** is a logical and independent piece of work and the **work transition** represents a flow of a business process among activities. To support process enactment and role assignment, additional constructs such as an execution rule and organization structure need to be defined (Amghar, 2000; Workflow Management Coalition, 1998). However, since the goal of this chapter is process information sharing among different organizations that focuses on the monitoring of a global process status, we view a process model as consisting of a number of activities and their associating work transitions.

Figure 1 graphically represents the basic concepts of an inter-organizational process model. As notations to depict an inter-organizational process model, **PM_i** denotes the i-th participating local process model and **$A_{i,j}$** denotes the j-th activity of PM_i. In the two local process models, PM_1 and PM_2, constituting the inter-organizational process, the circle denotes an activity and the solid arrow between circles denotes a **trigger** that represents the order of a work transition. The dotted arrow denotes message movement, representing an interaction between different process models. In addition, PM_1 considers PM_2 as a remote sub-process and uses a local activity, $A_{1,3}$, to represent the activities of PM_2. Such an activity is called a **process activity**. The message from $A_{1,3}$ causes the start of PM_2; $A_{1,3}$ is suspended during the execution of PM_2; after receiving the returned message from PM_2, $A_{1,3}$ terminates.

Figure 1 also shows five kinds of work transitions, including AND-SPLIT, AND-JOIN, OR-SPLIT, OR-JOIN, and SERIAL (Leymann, 2000; Workflow Management Coalition, 1998). The black dot between solid arrows denotes a discriminator to distinguish AND-JOIN from OR-JOIN, and AND-SPLIT from OR-SPLIT. Consider the activity $A_{1,1}$ as an example in Figure 1. When $A_{1,1}$ terminates, both $A_{1,2}$ and $A_{1,3}$ follow in parallel (we denote this as **AND-SPLIT**); when both $A_{1,2}$ and $A_{1,3}$ terminate, $A_{1,4}$ follows (**AND-JOIN**). On the other hand, when $A_{2,1}$ terminates, only one activity among $A_{2,2}$ and $A_{2,3}$ follows (**OR-SPLIT**); when either $A_{2,2}$ or $A_{2,3}$ terminates, $A_{2,4}$ follows (**OR-JOIN**). When $A_{2,4}$ terminates, $A_{2,5}$ follows without any other splitting or joining activities (**SERIAL**).

An Order Fulfillment Example

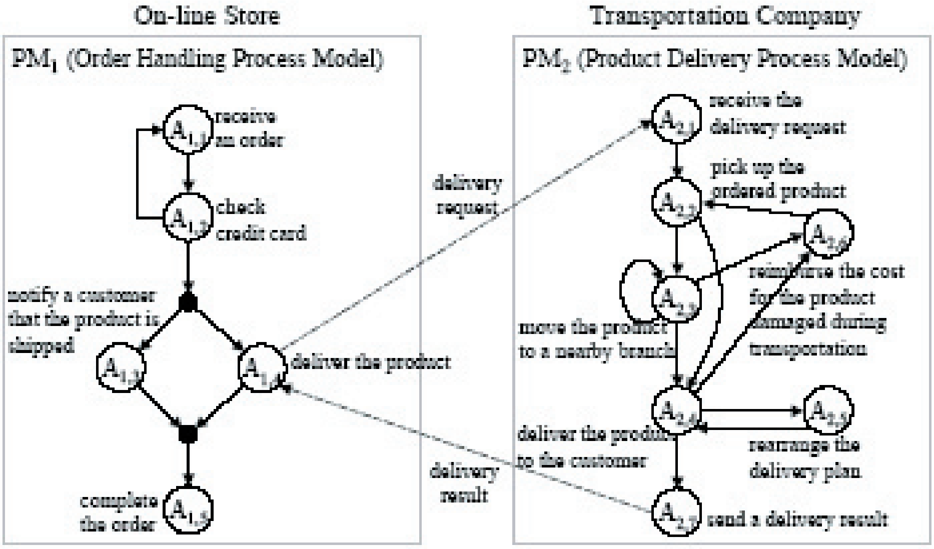
Figure 2 provides an order fulfillment process as an example of an inter-organizational process. Participating organizations include an online store (outsourcing company) and a transportation company (insourcing company), which respectively operate an order handling process model and product delivery process model. PM_1 and PM_2 denote these process models.

In what follows, the overall order fulfillment process is described. When a customer places an order to purchase a product from the online store using her credit card ($A_{1,1}$), order handling process PM_1 checks the credit card ($A_{1,2}$). If the credit card is valid, PM_1 notifies the customer that the product will be shipped ($A_{1,3}$) while it sends a delivery request to the transportation company ($A_{1,4}$). Note that $A_{1,4}$ in PM_1 is a process activity that stands for remote process model PM_2 and thus $A_{1,4}$ is suspended until PM_2 returns a delivery result.

Meanwhile, upon receiving the delivery request ($A_{2,1}$), product delivery process PM_2 picks up the ordered product at the online store's warehouse ($A_{2,2}$), ships it to a branch near the shipping address ($A_{2,3}$), and finally delivers it to the customer ($A_{2,4}$). As shown in Figure 2, $A_{2,3}$ can be omitted (as shown in the solid arrow from $A_{2,2}$ to $A_{2,4}$) or iteratively executed (the solid arrow from $A_{2,3}$ to $A_{2,3}$), depending on the shipping address and the company's transportation network. If the product cannot be delivered at $A_{2,4}$ because of the customer's absence, PM_2 rearranges the delivery plan to execute $A_{2,4}$ again ($A_{2,5}$). When the product is damaged during transportation at $A_{2,3}$ or $A_{2,4}$, the transportation company reimburses the cost ($A_{2,6}$) and exchanges the damaged product with a new one ($A_{2,2}$). After successfully delivering the product to the customer, a delivery result is sent to PM_1 ($A_{2,7}$) and then PM_1 completes the order ($A_{1,5}$).

If the transportation company provides the online store its detailed process status for product delivery, the online store can improve the operational efficiency and customer ser-

Figure 2: Order fulfillment example


On-line Store
Transportation Company
PM_1 (Order Handling Process Model)
PM_2 (Product Delivery Process Model)
$A_{1,1}$ receive an order
$A_{1,2}$ check credit card
notify a customer that the product is shipped
$A_{1,3}$
$A_{1,4}$ deliver the product
complete the order $A_{1,5}$
delivery request
delivery result
$A_{2,1}$ receive the delivery request
$A_{2,2}$ pick up the ordered product
$A_{2,6}$ reimburse the cost for the product damaged during transportation
$A_{2,3}$ move the product to a nearby branch
$A_{2,4}$ deliver the product to the customer
$A_{2,5}$ rearrange the delivery plan
$A_{2,7}$ send a delivery result

vices by accommodating extended order tracking capabilities, including both the internal order handling process and external product delivery process. To do so, the online store is to first integrate the two local process models into one inter-organizational process model. If the local process models and their interactions are specified explicitly and coordinated by one organization, it is possible to integrate the two local process models (van der Aalst, 1999). However, such an assumption of full information sharing is hardly accepted in the modern business environment because of the autonomy problem stated in section 1. For example, when the transportation company does not want its operational mistakes to be known to customers, it wants to hide the process information related to $A_{2,6}$ (reimbursement for the damaged product). Thus, the transportation company should construct a partial process model by removing $A_{2,6}$ from its product delivery process $PM_{2,}$ so that the online store uses the partial process model instead of full PM_2 to compose an inter-organizational process model. However, such a process information sharing scheme that supports sharing policies among participants has been rarely dealt with in the existing research on inter-organizational processes.

The process information sharing system of the online store constantly captures and merges the process statuses of the order handling and product delivery processes in order to provide customers the dynamically changing global process status. In doing so, the system refers to the inter-organizational process model, including the two process models, PM_1 and PM_2, and their interactions at the run time, and thus the details of the inter-organizational process model should be encoded in the system. However, achieving agility in companies often causes a change of a process model or sharing policy. When such a change arises, the system should be re-implemented and newly created to reflect the change with maintenance cost and time.

The following sections propose the federated process framework and show how the framework overcomes these problems while facilitating effective implementation of process information sharing.

FEDERATED PROCESS FRAMEWORK

The federated database system approach is referred to in the development of the federated process framework. A federated database system was originally proposed to facilitate information sharing among cooperating but autonomous local database systems (Heimbigner, 1985; Sheth, 1990). While the local database systems independently perform local operations, they only provide partial and controlled data for the requests of the federated database system. Typical development process of the federated database system is composed of the following four steps:

(i) Standardizing local database schemas;
(ii) Controlling and restricting the schemas;
(iii) Integrating the multiple schemas into one schema;
(iv) Customizing the integrated schema for end users' needs.

In this chapter, the federated database approach is adopted and extended as a framework for the process information sharing among collaborating but autonomous business processes. Specifically, this section proposes the four steps of the federated process framework in detail.

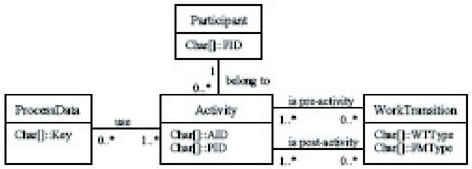
By providing an object data model and XML document structure, we also show how process models and process statuses are represented and manipulated in each step.

First Step: Transformation of Local Process Models

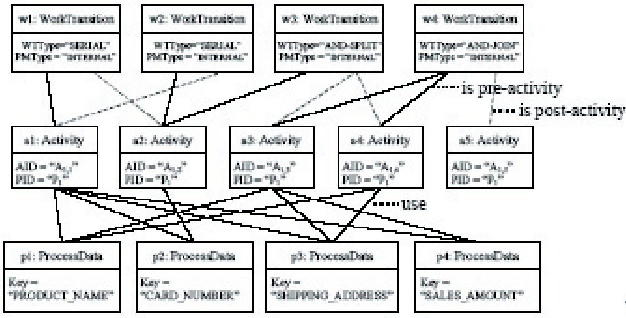
The first step of the federated process framework is to transform each of the local process models represented by diverse process modeling methods into a semantically equivalent one represented by a canonical standard method. Typically, the process modeling method representing a participant's process model varies depending on the software toolkit or process designer that has been employed to develop the participant's business process system. This diversity of the process modeling methods makes it difficult to integrate different process models (Dabke, 1999; Georgakopoulos, 1999). To address this problem and accommodate seamless integration of different business process models, many standard organizations and researchers have proposed the canonical standard methods for process modeling (Workflow Management Coalition, 1998; Object Management Group, 2000; Bolcer, 1999). As a canonical standard method, this chapter uses concepts and notations (e.g., the graphical notations in Figures 1 and 2) of the workflow reference model (Workflow Management Coalition, 1998) proposed by the Workflow Management Coalition because of its popularity among commercial software vendors, and interoperability with other standards.

In Figure 3(a), using the class diagram of the Unified Modeling Language (UML) (Booch, 1999), we define an object data model that acts as a dedicated database schema to manage process models in the OODB. In the UML, a class is represented by a rectangle, containing a class name on the upper side and attributes on the lower side. An association between classes is represented by a line that has an association name and multiplicities on

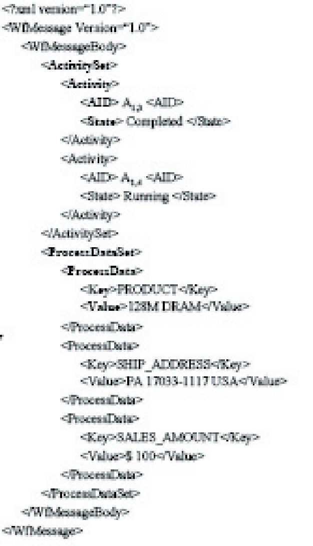
Figure 3: Representation of a process model and process status using the UML and XML



(a) Object Data Model for Process Models (Class Diagram of UML)



(b) Object Data Example for Order Handling Process PM_1 (Object Diagram of UML)



```
<?xml version="1.0"?>
<WfMessage Version="1.0">
    <WfMessageBody>
        <ActivitySet>
            <Activity>
                <AID> A1,3 <AID>
                <State> Completed </State>
            </Activity>
            <Activity>
                <AID> A1,4 <AID>
                <State> Running </State>
            </Activity>
        </ActivitySet>
        <ProcessDataSet>
            <ProcessData>
                <Key>PRODUCT</Key>
                <Value>128M DRAM</Value>
            </ProcessData>
            <ProcessData>
                <Key>SHIP_ADDRESS</Key>
                <Value>PA 17033-1117 USA</Value>
            </ProcessData>
            <ProcessData>
                <Key>SALES_AMOUNT</Key>
                <Value>$ 100</Value>
            </ProcessData>
        </ProcessDataSet>
    </WfMessageBody>
</WfMessage>
```

(c) Process Status Represented by an XML Document both sides of the connected classes. The *Participant*, *Activity*, and *WorkTransition* classes respectively stand for participants, activities, and work transitions of a process model. The *ProcessData* class stands for process data as a set of data that is referenced or modified by activities of a process model. Note that the *Participant* class has an one-to-many association, "*belong to,*" with the *Activity* class; the *WorkTransition* class has two many-to-many associations, "*is pre-activity*" and "*is post-activity,*" with the *Activity* class to respectively indicate its preceding and following activities; the *Activity* class has a many-to-many association, "*use,*" with the *ProcessData* class to indicate process data referenced or modified by the activity.

Using the object diagram of the UML, Figure 3(b) shows an object data example that represents the order handling process, PM_1, in the form of the presented object data model. An object is represented by a rectangle that has an object name on the upper side and attribute values on the lower side. An association is represented by a line. In particular, a solid line and dashed line are used to distinguish "*is pre-activity*" and "*is post-activity*" associations between *WorkTransition* and *Activity* objects. For example, *WorkTransition* object *w3* represents the AND-SPLIT work transition from $A_{1,2}$ to both $A_{1,3}$ and $A_{1,4}$ in Figure 2. As an "*is pre-activity*" association, the activity object, *a2*, is linked with a solid line, while objects *a3* and *a4* are linked with two dashed lines as *"is post-activity"* associations. In this way, the object data example in Figure 3(b) represents the whole structure of the order handling process, PM_1.

The process status is defined as a set of ongoing activities' states and a set of process data used by the activities. In PM_1, suppose that $A_{1,3}$ has notified a customer that an ordered product is being shipped, while $A_{1,4}$ is in the midst of delivering the product. Then, $A_{1,3}$ and $A_{1,4}$ are ongoing activities in the process instance and they use three process data – product, shipping address, and sales amount – according to the object data example in Figure 3(b). Figure 3(c) shows an XML representation of this process status on the basis of the following XML Document Type Declaration (XML DTD) that is a generic XML document structure representing a process status:

```
<! ELEMENT WfMessage (WfMessageBody)>
<! ATTLIST WfMessage Version CDATA #REQUIRED>
<! ELEMENT WfMessageBody (ActivitySet, ProcessDataSet)>
<! ELEMENT ActivitySet (Activity*) >
<! ELEMENT Activity (AID, State)>
<! ELEMENT AID (#PCDATA)>
<! ELEMENT State (#PCDATA)>
<! ELEMENT ProcessDataSet (ProcessData*) >
<! ELEMENT ProcessData (Key, Value)>
<! ELEMENT Key (#PCDATA)>
<! ELEMENT Value (#PCDATA)>
```

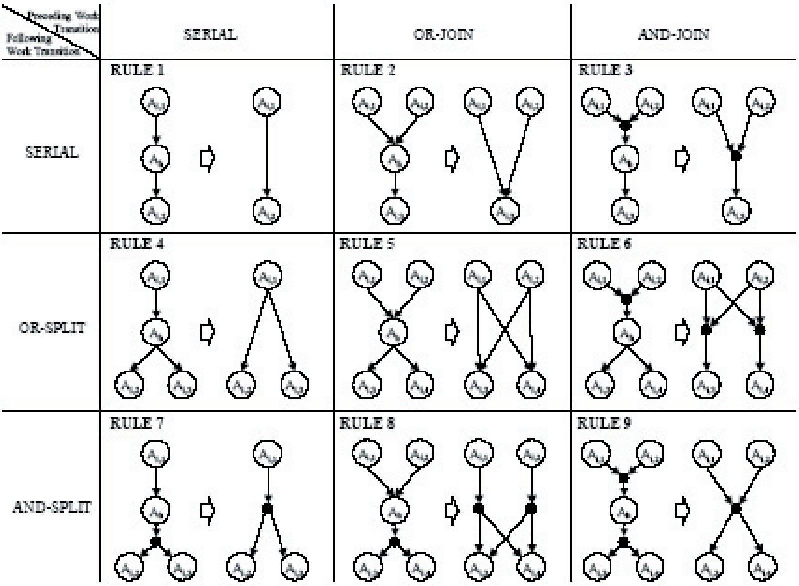
In the beginning, the XML DTD borrows the Wf-XML specification that the Workflow Management Coalition (2000) proposed as a standard XML-based representation of a process status, and thus it defines the root element, *WfMessage*, to identify a Wf-XML message and its child element, *WfMessageBody*, to indicate a message body section. In representing a process status, however, the Wf-XML covers only the state of a process instance itself, such as "running," "suspended," "terminated," etc., and does not support the activity-level process status, though an activity is a logical and independent unit of work within a business process. Thus, the proposed XML document structure extends the Wf-XML by taking an activity as a unit of process information sharing. More specifically, an *ActivitySet* element and a *ProcessDataSet* element are defined inside the *WfMessageBody* element. The *ActivitySet* element is to contain all the ongoing activities currently involved (*Activity* element), and every *Activity* element has its identity (*AID* element) and state (*State* element). Similarly, the *ProcessDataSet* element is to contain all the process data used by the ongoing activities (*ProcessData* element), and every *ProcessData* element has its identity (*Key* element) and value (*Value* element). In this way, the tree-structured XML DTD represents activities' states and associated process data systematically. Moreover, it serves as a generic message specification to facilitate data transfer between different business processes.

In brief, the first step introduces systematic representations of a process model and process status by providing an object data model and XML document structure. Using these representations, the following steps will describe how local process models can be restricted and integrated according to local sharing policies.

Second Step: Construction of an External Process Model

The second step is to establish a sharing policy and to extract a partial process model from a local process model according to the established sharing policy. Since an activity is a logical unit of process information sharing, the sharing policy is defined as a set of activities shared with other participants. For example, when the transportation company in Figure 2 wants to hide the process information related to $A_{2,5}$ and $A_{2,6}$ from the online store, the shared activities are $A_{2,1}$, $A_{2,2}$, $A_{2,3}$, $A_{2,4}$, and $A_{2,7}$. In doing so, the transportation company constructs

Figure 4: Rules for removing a hidden activity

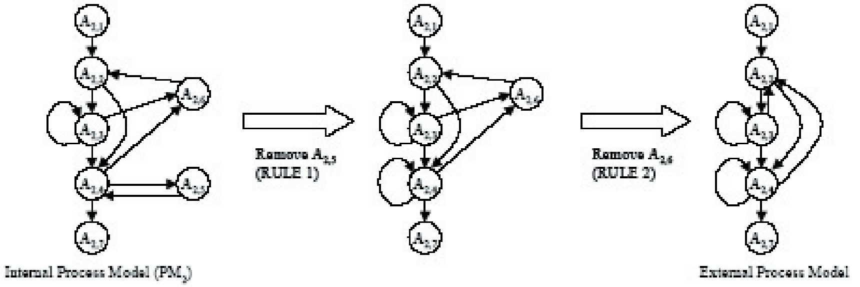


a partial process model from its original process model, PM_2, by removing $A_{2,5}$ and $A_{2,6}$, and provides the online store with it instead of PM_2. To distinguish the newly constructed process model from the original one, we call it an **external process model** and the original one an **internal process model**.

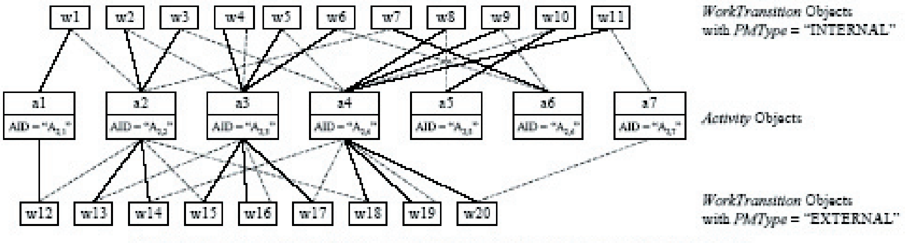
Figure 4 presents nine rules to remove a hidden activity for all the possible combinations of a preceding work transition (SERIAL, OR-JOIN, or AND-JOIN) and following one (SERIAL, OR-SPLIT, or AND-SPLIT) of the hidden activity. In each cell of Figure 4, by removing a hidden activity, A_h, from the left internal process model, the right external process model is generated. The presented rules preserve possible work paths and thus do not distort business logic even though the hidden activity is removed. For example, the left internal process model in RULE 6 has two possible work paths (i.e., after both $A_{i,1}$ and $A_{i,2}$ terminate, $A_{i,3}$ or $A_{i,4}$ follows them through A_h). After A_h is removed, RULE 6 creates two AND-JOINs (i.e., from both $A_{i,1}$ and $A_{i,2}$ to $A_{i,3}$; from both $A_{i,1}$ and $A_{i,2}$ to $A_{i,4}$) in the right external process model and consequently preserves the existing two work paths.

By applying these rules to each of hidden activities iteratively, an external process model can be constructed from an internal process model. Suppose that, in Figure 2, the transportation company hides $A_{2,5}$ and $A_{2,6}$ from the online store. Then, Figure 5(a) shows the procedure that constructs the right external process model from the left internal process model, PM_2, of the transportation company. $A_{2,5}$ is first removed according to RULE 1 and a trigger for self-iteration of $A_{2,4}$ is created. Next, $A_{2,6}$ is removed according to RULE 2 and two triggers, one from $A_{2,3}$ to $A_{2,2}$ and one from $A_{2,4}$ to $A_{2,2}$, are created.

Figure 5: Construction of an external process model and corresponding object data example



(a) Procedure for Constructing an External Process Model of the Transportation Company



(b) Object Data Example for Representing the Internal and External Process Models of the Transportation Company

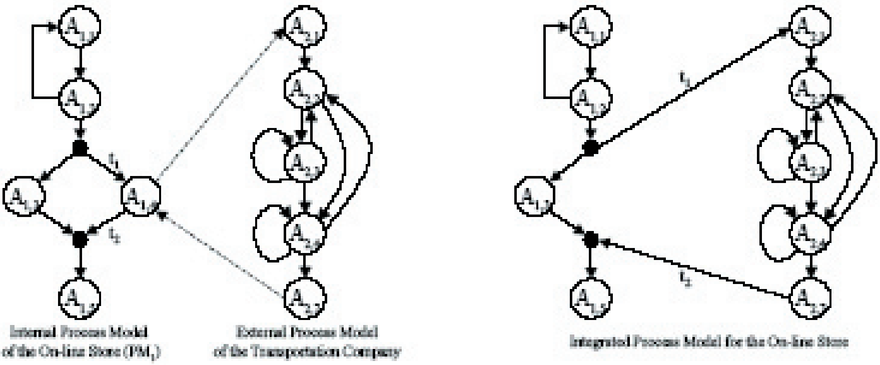
Figure 5(b) presents an object data example that represents both the internal and external process models in Figure 5(a). Note that the *WorkTransition* objects with *PMType* = "INTERNAL" (*w1* to *w11*) denote the work transitions in the internal process model and the *WorkTransition* objects with *PMType* = "EXTERNAL" (*w12* to *w20*) denote the work transitions in the external process model. Using the *WorkTransition.PMType* attribute, we can extract shared activities from the object data example, and thus this object data example shows how to represent the transportation company's sharing policy toward the online store. In terms of the XML document, an XML document based on the internal process model can be easily transformed into an XML document based on the external process model by removing *Activity* elements corresponding to hidden activities.

To accommodate the activity-level sharing policy, this step introduces the concept of an external process model and its construction method, and furthermore describes how to represent the sharing policy and corresponding external process model in the presented object data model. As a result, this step provides an answer to the first question of the autonomy problem.

Third Step: Composition of an Integrated Process Model

This step describes a method for a participant to merge its internal process model and an external process model provided by another participant. Such a merged process model is called an **integrated process model**. The method to compose the integrated process model is to substitute the process activity in the internal process model with the corresponding external process model. Earlier in the chapter, it states that the internal process model uses a process activity to represent the activities of the external process model. Figure 6(a) shows that the process activity $A_{1,4}$ of PM_1 represents the activities of the external process model of the transportation company. $A_{1,4}$ sends a request message to the starting activity, $A_{2,1}$, of the external process model, and receives a result message from the ending activity, $A_{2,7}$, of the external process model.

Figure 6: Integrated process model composed by the online store


$A_{1,1}$
$A_{1,2}$
t_1
$A_{1,3}$
$A_{1,4}$
t_2
$A_{1,5}$
$A_{2,1}$
$A_{2,2}$
$A_{2,3}$
$A_{2,4}$
$A_{2,7}$
Internal Process Model of the On-line Store (PM_1)
External Process Model of the Transportation Company
Integrated Process Model for the On-line Store

(a) Before Integration

(b) After Integration

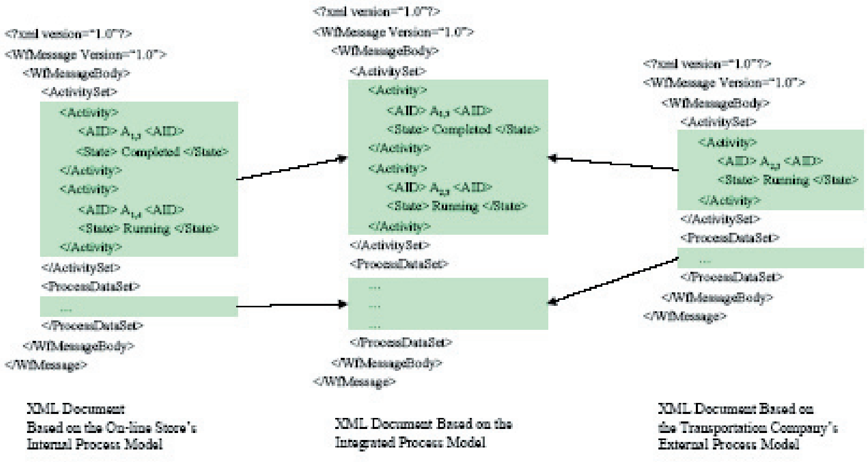
The substitution of the process activity with the corresponding external process model is performed in the following ways: first, the process activity's incoming trigger changes its destination from the process activity to the starting activity of the external process model. Second, the process activity's outgoing trigger changes its origin from the process activity to the ending activity of the external process model. Finally, the process activity is removed from the internal process model. For example, t_1 and t_2 in Figure 6(a) are $A_{1,4}$'s incoming and outgoing triggers, respectively. To constitute the integrated process model from the two process models in Figure 6(a), we change t_1's destination from $A_{1,4}$ to $A_{2,1}$ and t_2's origin from $A_{1,4}$ to $A_{2,7}$, and finally, remove $A_{1,4}$ from PM_1. Figure 6(b) shows the resultant integrated process model.

Using the *PMType* attributes of the *WorkTransition* objects, the object data example in Figure 5(b) showed how the external process model could be represented with the object data model in Figure 3(a). Similarly, the integrated process model can be represented with the object data model by assigning "INTEGRATED" to the *PMType* attributes of the *WorkTransition* objects related to the integrated process model. Then, an interaction between different process models (e.g., the work transition between $A_{1,4}$ and $A_{2,1}$) is represented in the same way as a work transition within a process model (e.g., one between $A_{2,1}$ and $A_{2,2}$). However, since two different organizations participate in the interaction, the interaction has special features such as a distributed transaction and a separated role and responsibility model. The ebXML Business Process Specification Schema (BPSS) (UN/CEFACT, 2001) that provides a detailed UML class diagram to design an interaction between collaborating business entities supports a distributed transaction and a separated role and responsibility model by considering an interaction as a business transaction between activities. Thus, the object data model in Figure 3(a) can be extended by inheriting a specialized interaction class from the *WorkTransition* class and connecting the interaction class and the UML class diagram of the ebXML BPSS; consequently, the extended data model would describe the interactions more precisely in terms of a distributed transaction and a separated role and responsibility model. However, to simplify the federated process framework and focus on the research questions, this chapter will use the object data model in Figure 3(a) in the remaining parts without extending it.

Meanwhile, the XML document based on an integrated process model can be derived by merging XML documents based on internal or external process models comprising the integrated process model. To merge XML documents based on different process models, each of their *ActivitySet* elements and *ProcessDataSet* elements is merged separately. Particularly, an *Activity* element corresponding to a process activity should be deleted since the process activity does not exist in the integrated process model. Figure 7 shows an example of merging the left XML document representing the online store's order handling process status and the right XML document representing the transportation company's product delivery process status. The resultant XML document in the middle of Figure 7 represents the global process status that means, "the online store has notified a customer that the product is being shipped, and the transportation company is fulfilling the delivery by moving the product to a nearby branch." Note that the *Activity* element corresponding to the process activity, $A_{1,4}$, disappears in the XML document based on the integrated process model.

The second and third steps present the detailed methods that restrict and seamlessly integrate local process models according to participants' sharing policies, and thus these methods provide an answer to the second question of the autonomy problem. Furthermore, when a participant's process model or sharing policy is changed, these methods support

Figure 7: XML document based on the integrated process model in Figure 6(b)



```
<?xml version="1.0"?>
<WfMessage Version="1.0">
  <WfMessageBody>
    <ActivitySet>
      <Activity>
        <AID> A1,3 <AID>
        <State> Completed </State>
      </Activity>
      <Activity>
        <AID> A1,4 <AID>
        <State> Running </State>
      </Activity>
    </ActivitySet>
    <ProcessDataSet>
      ...
    </ProcessDataSet>
  </WfMessageBody>
</WfMessage>
```

XML Document
Based on the On-line Store's
Internal Process Model

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
  <WfMessageBody>
    <ActivitySet>
      <Activity>
        <AID> A1,3 <AID>
        <State> Completed </State>
      </Activity>
      <Activity>
        <AID> A2,3 <AID>
        <State> Running </State>
      </Activity>
    </ActivitySet>
    <ProcessDataSet>
      ...
      ...
      ...
    </ProcessDataSet>
  </WfMessageBody>
</WfMessage>
```

XML Document Based on the
Integrated Process Model

```
<?xml version="1.0"?>
<WfMessage Version="1.0">
  <WfMessageBody>
    <ActivitySet>
      <Activity>
        <AID> A2,3 <AID>
        <State> Running </State>
      </Activity>
    </ActivitySet>
    <ProcessDataSet>
      ...
    </ProcessDataSet>
  </WfMessageBody>
</WfMessage>
```

XML Document Based on
the Transportation Company's
External Process Model

automatic generation of external and integrated process models. Thus, the second and third steps also provide base algorithms to solve the agility problem.

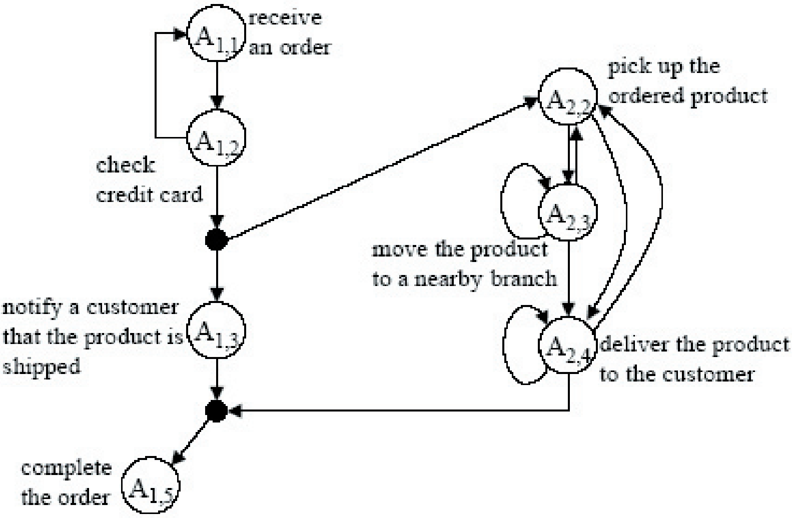
Fourth Step: Construction of a Customized Process Model

The final step of the federated process framework is to construct a partial process model from an integrated process model. The integrated process model can be quite large and complex, and thus it may need to be reduced into a partial process model to support access control for user-level securities or to provide a smaller process model appropriate to a specific task of a user or user group. Such a partial process model constructed from an integrated process model is called a **customized process model**. When a customized process model is constructed, users' needs for customization are first identified, and activities that are unnecessary for the needs are removed. To remove the unnecessary activities, the nine rules presented in the second step are iteratively applied to the integrated process model. Figure 8 exemplifies a customized process model for the online store's customer support staff, which is constructed by removing messaging activities, $A_{2,1}$ and $A_{2,7}$, from the integrated process model in Figure 6(b). This customized process model helps the customer support staff to focus only on the process information necessary for customer services.

The existing object data example can be easily extended in order to manage the customized process model by adding new *WorkTransition* objects for the customized process model and assigning "CUSTOMIZED" to its *PMType* attributes. On the other hand, in the same way as in the second step, an XML document based on the integrated process model can be transformed into one based on the customized process model by eliminating the *Activity* elements corresponding to the removed activities.

This section presents the federated process framework to facilitate process information sharing among collaborating but autonomous organizations. Specifically, the four steps of the framework facilitate seamless integration of multiple collaborating process models while accommodating flexible activity-level sharing policies. To support these features of

Figure 8: Customized process model for the online store's customer support staff


$A_{1,1}$ receive an order
$A_{1,2}$ check credit card
$A_{1,3}$ notify a customer that the product is shipped
$A_{1,5}$ complete the order
$A_{2,2}$ pick up the ordered product
$A_{2,3}$ move the product to a nearby branch
$A_{2,4}$ deliver the product to the customer

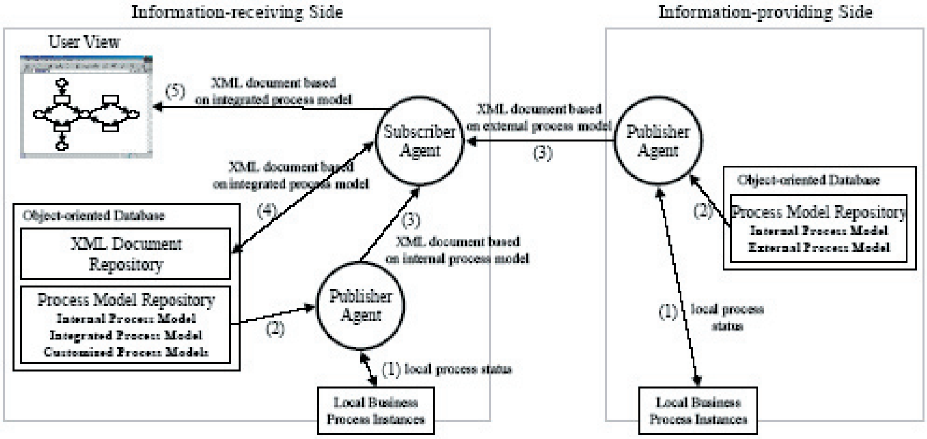
the framework systematically, this section also presents the object data model to manage a process model and the XML document structure to represent process status. Therefore, the proposed federated process framework provides answers to the two questions of the autonomy problem and consequently addresses the autonomy problem.

SYSTEM ARCHITECTURE AND ITS PROTOTYPE SYSTEM

This section presents a system architecture based on the federated process framework with its prototype system. The system architecture is composed of the following two types of system components: a repository and an agent. Figure 9 shows the components and their interactions. In terms of the repository, the system architecture contains process model repositories and an XML document repository. Employing the object data model in Figure 3(a), the **process model repository** manages internal, integrated, and customized process models on the information-receiving side, and it manages internal and external process models on the information-providing side. The **XML document repository** resides only on the information-receiving side and manages a set of XML documents that represents the execution history of global process instances.

In terms of the agent, the system architecture contains publisher agents and a subscriber agent. On both sides, the **publisher agent** detects the change of a local process status and sends the XML document representing the changed local process status to the subscriber agent that resides on the information-receiving side. The **subscriber agent** manages the XML document repository and broadcasts it to relevant user views. When the subscriber agent receives an XML document from a publisher agent, it first refers to the latest XML document based on the integrated process model from the XML document repository, modifies it according to the received XML document, and inserts the modified XML document based on the integrated process model into the XML document repository. Then, the subscriber

Figure 9: System architecture based on the federated process framework

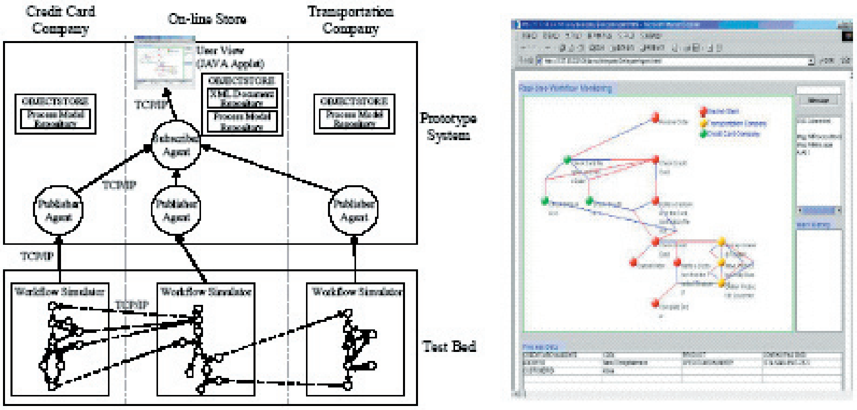

Information-receiving Side
Information-providing Side
User View
(5) XML document based on integrated process model
Subscriber Agent
XML document based on external process model
(3)
Publisher Agent
XML document based on integrated process model
(4)
Object-oriented Database
XML Document Repository
Process Model Repository
Internal Process Model
Integrated Process Model
Customized Process Models
(2)
(3) XML document based on internal process model
Publisher Agent
(1) local process status
Local Business Process Instances
Object-oriented Database
(2)
Process Model Repository
Internal Process Model
External Process Model
(1) local process status
Local Business Process Instances

agent sends the modified XML document to the relevant user views that visually provide end users current global process status.

The numbered arrows in Figure 9 show the change notification procedure ((1) to (5)) in detail. In the proposed system architecture, the change notification procedure is encoded in the application programs such as the agents and user view, but participants' process models and sharing policies are managed in the process model repository. Thus, the system procedures in the application programs are independent of contents of the process model repository. When a participant's process model or sharing policy is changed, the associated external and integrated process models are automatically generated through the second and third steps of the federated process framework. Then, the proposed system architecture can adapt itself to the change just by revising the contents of the process model repository. This adaptability of the system architecture provides the answer to the first question of the agility problem; in addition, the system components comprising the system architecture provide the answer to the second question of the agility problem. Consequently, the system architecture based on the federated process framework addresses the agility problem.

Using the system architecture, a prototype system was also developed based on a commercial OODB called OBJECTSTORE with the JAVA programming language. In the prototype system, we employed a real inter-organizational process model that is composed of three collaborating internal process models, including the online store's order handling process, the transportation company's delivery process, and the credit card company's card verification process. The internal process models are managed separately in different process model repositories that were built up in the OBJECTSTORE as the form of the object diagram in Figure 3(b). The agents and a user view, which were implemented using the JAVA programming language, access the process model repositories and interact with each other through the TCP/IP network. The lower part of Figure 10(a) shows a test bed that imitates the executions of the three business processes to evaluate the prototype system. The adopted test bed contains three workflow simulators corresponding to the three participating internal process models. According to the internal process models and their interactions, these work-

Figure 10: Evaluation of the prototype system



(a) Simulation Setup for Prototype Evaluation

(b) Resultant User View

flow simulators proceed simultaneously as independent software processes while interacting with each other. Whenever a workflow simulator's state is changed, the workflow simulator sends an event to a publish agent and then the change notification procedure is performed according to the change. Figure 10(b) shows the JAVA applet-based user view that provides real-time process status on the online store's customized process model.

All the system components of the prototype system were tested for accuracy and completeness by changing the sharing policy for the amount of shared process information (i.e., changing the external and integrated process models in the process model repositories). Furthermore, by performing scenario experiments for the change of an internal process model or a sharing policy, the prototype system was evaluated in terms of the adaptability. Consequently, the prototype system shows that it is both possible and feasible to develop the federated process framework while solving the autonomy and agility problems.

DISCUSSION

Most supporting business processes (e.g., delivery process of an online store) rarely contribute to the improvement of core competencies even though they are indispensable for steady enterprise business operations. In some cases, they even burden companies with excessive cost for human resource and system maintenance, and consequently make it difficult for the companies to focus on their core business. Then, outsourcing the supporting business processes to external partners often helps the companies accommodate cost efficiency and core business focusing.

However, an outsourcing company still needs to monitor and control the outsourced business processes, while an insourcing company wants to keep the autonomy and agility as an independent business unit. The main contribution of the chapter is to suggest a conceptual design that satisfies these two conflicting needs of insourcing and outsourcing companies by providing the federated process framework and its system architecture.

In reality, the framework supports the practical implementation of business process outsourcing and then contributes to accelerating effective business collaboration in contemporary business environments. In particular, it can be applied to a wide variety of practical circumstances such as the following two cases in which tightly-coupled and loosely-coupled business process outsourcing take place respectively: a holding company and virtual enterprise.

First, a holding company is formed to buy shares in other companies, which it then controls. The regulatory change and the desire for a large customer base have increased the number of large holding companies through tremendous industry consolidation recently. However, companies inside a holding company typically perform similar business functions such as logistics and IT system management redundantly, which impedes the economy of scale. Then, if the redundant business functions are transferred from the companies to one of them or an external insourcing company, overall cost efficiency of the holding company could be improved by virtue of the sharing of human resource and system infrastructure. Second, a virtual enterprise outsources multiple business functions and then controls them as if they were performed locally. As the online store in Figure 10 collaborates with a transportation company and a credit card company, a virtual enterprise works together with multiple insourcing companies since it may choose different insourcing companies by business functions. In doing so, a virtual enterprise tends to incline to best-of-breed virtual integration without any preferences or dependencies on specific partners; therefore, the most influential criterions to select external partners are service quality and customized service fulfillment.

These two cases show that the goals of business process outsourcing vary from global optimization of the cost structure to local optimization, depending on the governance structure and business relationship of insourcing and outsourcing companies. On the other hand, in both the cases, insourcing companies want to keep autonomy and agility, and thus hesitate about the full information sharing with an outsourcing company. To support such various cases of business process outsourcing, the federated process framework presents the mechanism to control shared process data with the shared policy.

CONCLUSIONS AND FUTURE RESEARCH

Recall that the initial goal of the chapter was to provide a system framework to facilitate process information sharing in the modern business environment. Specifically, at the beginning of the chapter, we issued the autonomy and agility problems. These two problems have not been previously addressed in the literature, though they are crucial obstacles to developing process information sharing. To achieve an efficient and applicable framework, this chapter makes efforts to address these problems in the following ways:

First, to solve the autonomy problem, we proposed the federated process framework. Specifically, the first step of the framework provides an object data model and XML document structure as systematic representations of a process model and process status. The second step distinguishes the external process model from the internal process model and provides the detailed rules to automatically generate the external process model. The third step presents the method to seamlessly integrate multiple process models according to participants' sharing policies. The fourth step tailors the integrated process model for the purposes of customization and user-level access control. In this way, the four steps facilitate the activity-level sharing policy and seamless process model integration. Consequently, the federated process framework overcomes the autonomy problem by supporting flexible sharing policies, and accommodates wide applicability to various practical situations, from loosely-coupled cases to tightly-coupled cases.

Second, to solve the agility problem, we proposed the system architecture based on the federated process framework. In this architecture, process models and sharing policies are separated from the application programs and are managed in the process model repository. When a participant's process model or sharing policy is changed, associated external and integrated process models are automatically generated through the second and third steps of the federated process framework. Then, the system architecture can adapt itself to the change just by revising the contents of the process model repository. The presented object data model and XML document structure make it possible to achieve such adaptability by providing conceptual designs that can be extended for the actual development of the process model repository and change notification procedure. As a result, the system architecture addresses the agility problem and allows sufficient adaptability to support the entire life cycle of process information sharing. By adding physical system capabilities such as security control and data conflict resolution, the proposed system architecture can be extended as a fully-fledged physical system design for process information sharing.

In future research, first, we are extending the federated process framework to cover the peer-to-peer model (Workflow Management Coalition, 1998) of the inter-organizational process, in which local processes exchange asynchronous messages at their runtime. In presenting the federated process framework, this chapter focuses on the hierarchical model (Workflow Management Coalition, 1998), in which a local process uses a process activity to represent other collaborating process. The hierarchical model is the most typical form of the inter-organizational process, but the peer-to-peer model is more suitable to represent complex business collaboration than the hierarchical model. To cover the peer-to-peer model, the third step of the federated process framework needs to be redeveloped due to its underlying assumption about the process activity.

Second, we plan to extend the system architecture in order to improve its applicability in a modern business-to-business computing environment. Particularly, the service-oriented architecture (Arsanjani, 2002) and web services (Kreger, 2003) provide underlying system framework and platform for the actual implementation of the business process outsourcing. In the service-oriented architecture, if a company outsources its business process, a corresponding insourcing company encapsulates the outsourced business process into a service. The outsourcing company invokes the service to communicate and collaborate with the outsourced business process. Thus, these technologies largely simplify the system design and implementation of the business process outsourcing. By engaging the service-oriented architecture and the web services standard in extending the federated process framework, we expect to make the framework more practical and widely used for the actual system development for business process outsourcing in a modern business-to-business computing environment.

ENDNOTES

[*] This research is supported by the Information Technology Research Center (ITRC) and sponsored by the Ministry of Information and Communication, the Republic of Korea.

REFERENCES

Alonso, G. et al. (1999). WISE: Business to business E-commerce. *Proceedings of the 9th IEEE International Workshop on Research Issues on Data Engineering. Information Technology for Virtual Enterprises (RIDE-VE 1999)*, Sydney, Australia, 23-24.

Amghar, Y. et al. (2000). Using business rules within a design process of active databases. *Journal of Database Management*, *11*(3), 3-15.

Arnold, K. et al. (2000). *The Java programming language*. California: Addison-Wesley.

Arsanjani, A. (2002). Developing and integrating enterprise components and services. *Communications of the ACM*, *45*(10), 31-34.

Bakos, Y. (1998). The emerging role of electronic marketplaces on the Internet. *Communications of the ACM*, *41*(8), 35-42.

Ball, M.O. et al. (2002). Supply chain infrastructures: System integration and information sharing. *SIGMOD Record*, *31*(1), 61-66.

Berfield, A. et al. (2002). A scheme for integrating E-services in establishing virtual enterprises. *Proceedings of the 12th International Workshop on Research Issues in Data Engineering e-Commerce/e-Business Systems (RIDE'02)*, California, USA, 134-142.

Böhm, K. (2000). On extending the XML engine with query-processing capabilities. *Proceedings of Advances in Digital Libraries (ADL)*, Bethesda, Maryland, 127-138.

Bolcer, G.A., & Kaiser, G. (1999). SWAP: Leveraging the web to manage workflow. *IEEE Internet Computing*, *3*(1), 85-88.

Booch, G. et al. (1999). *The Unified Modeling Language user guide*. Massachusetts: Addison-Wesley.

Carter, B. (2000). XML: Filling a data-description gap, Part II. *Journal of Database Management*, *11*(2), 30-33.

Casati, F. et al. (1998). Workflow evolution. *Data & Knowledge Engineering*, *24*(3), 211-238.

Chircu, A.M., & Kauffman, R.J. (2000). Reintermediation strategies in business-to-business electronic commerce. *International Journal of Electronic Commerce*, *4*(4), 7-42.

Cichocki, A. et al. (1998). *Workflow and process: Concepts and technology*. Massachusetts: Kluwer Academic Publishers.

D'Amours, S. et al. (1999). Networked manufacturing: The impact of information sharing. *International Journal of Production Economics*, *58*, 63-79.

Dabke, P. (1999). Enterprise integration via CORBA-based information agents. *IEEE Internet Computing*, *3*(5), 49-57.

Gartner. (2002). *Finance sector seeks IT outsourcing to meet business goals.* ITSV-WW-DP-0315.

Gartner. (2003). Why Asia/Pacific enterprises outsource, and why they don't. ITSC-AP-UW-0109.

Georgakopoulos, D. et al. (1999). Managing process and service fusion in virtual enterprises. *Information Systems*, *24*(6), 429-456.

Goranson, H.T. (1999). *The agile virtual enterprise: Cases, metrics, tools*. London: Quorum Books.

Hafeez, K. et al. (2002). Core competence for sustainable competitive advantage: A structured methodology for identifying core competence. *IEEE Transactions on Engineering Management*, *49*(1), 28-35.

Heimbigner, D., & McLeod, D. (1985). A federated architecture for information management. *ACM Transactions of Office Information Systems*, *3*(3), 253-278.

Herring, C., & Milosevic, Z. (2001). Implementing B2B contracts using BizTalk. *Proceedings of the 34th Hawaii International Conference on System Sciences*, Hawaii, USA, 4078-4087.

Kim, W. (1995). *Modern database systems: The object model, interoperability, and beyond*. New York: Addison-Wesley.

Kreger, H. (2003). Fulfilling the web services promise. *Communications of the ACM*, *46*(6), 29-34.

Kuechler, W. et al. (2001). Supporting optimization of business-to-business E-commerce relationships. *Decision Support Systems*, *31*, 363-377.

Lee, H.L. et al. (1997). The bullwhip effect in supply chains. *Sloan Management Review*, *38*, 93-102.

Leymann, F., & Roller, D. (2000). *Production workflow: Concepts and techniques*. New Jersey: Prentice Hall.

Mangan, P., & Sadiq, S. (2002). On building workflow models for flexible processes. *Proceedings of the 13th Australasian Database Conference ADC2002*, Melbourne, Australia.

Merz, M., & Lamersdorf, W. (1999). Crossing organizational boundaries with mobile agents in electronic service markets. *International Journal on Computer-Aided Engineering*, *6*(2), 91-104.

Mori, M. et al. (1999). Proposal of application architecture in electronic commerce service between companies. *Proceedings of the International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS)*, Santa Clara, USA, 46-49.

Object Management Group. (2000, April). *Workflow management facility specification V1.2*. [Online]. Available: http://www.omg.org/.

Progress Software. (2003). *ObjectStore*. http://www.objectstore.net/.

Ramamurthy, S., & Robinson, M.S. (2003). Simplify to succeed – Optimize the customer franchise and achieve operational scale: Retail financial institutions in 2005. *IBM Business Consulting Services Point of View Series*. Available: http://www.ibm.com/.

Scott-Morton, M.S. (1994). The 1990s research program: Implications for management and the emerging organization. *Decision Support Systems*, *12*(2), 251-256.

Sheth, A.P., & Larson, J.A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, *22*(3), 183-236.

Sundaram, M., & Shim, S.S.Y. (2001). Infrastructure for B2B exchanges with RosettaNet. *Proceedings of the International Workshop on Advance Issues of E-Commerce and Web-Based Information Systems (WECWIS)*, California, USA, 110-119.

Tuma, A. (1998). Configuration and coordination of virtual production networks. *International Journal of Production Economics*, *56*, 641-648.

UN/CEFACT & OASIS. (2001, May 11). *Business process specification schema V1.01*. [Online]. Available: http://www.ebxml.org/.

van der Aalst, W.M.P. (1999). Process-oriented architectures for electronic commerce and interorganizational workflow. *Information Systems*, *24*(8), 639-671.

Workflow Management Coalition. (1998, August 5). *Interface 1: Process definition interchange process model*, Document Number WfMC-TC-1016-P. [Online]. Available: http://www.wfmc.org/.

Workflow Management Coalition. (2000, May 1). *Workflow standard – Interoperability Wf-XML binding*, Document Number WFMC-TC-1023. [Online]. Available: http://www.wfmc.org/.

Zhou, Q. et al. (1998). An information management system for production planning in virtual enterprises. *Computers & Industrial Engineering*, *35*(1-2), 153-156.

Chapter XV

Online Analytical Mining for Web Access Patterns

Joseph Fong, City University of Hong Kong, Hong Kong

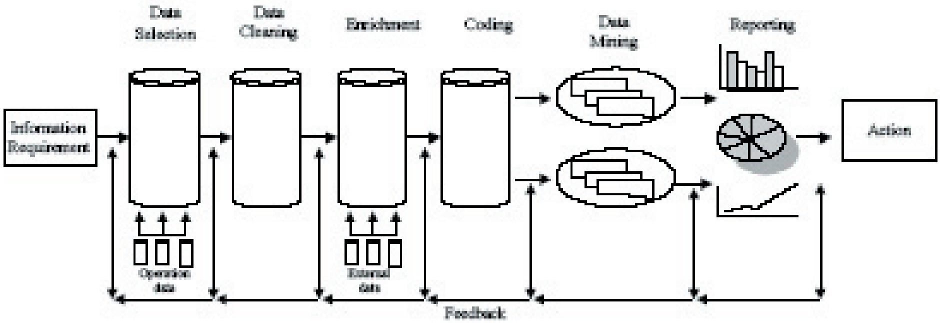
Hing Kwok Wong, City University of Hong Kong, Hong Kong

Anthony Fong, City University of Hong Kong, Hong Kong

ABSTRACT

The WWW and its associated distributed information services provide rich world-wide online information services, where objects are linked together to facilitate interactive access. Users seeking information from the Internet traverse from one object via links to another. It is important to analyze user access patterns, which helps improve web page design by providing an efficient access between highly correlated objects, and also assists in better marketing decisions by placing advertisements in frequently visited documents. We need to study the user surfing behavior through examining the web access log, browsing frequency of web pages and computing the average duration of visitors. This chapter offers an architecture to store the derived web user access paths in a data warehouse, and facilitates its view maintainability by use of metadata. The system will update the user access paths pattern with the data warehouse by the data operation functions in the metadata. Whenever a new user access path occurs, the view maintainability is triggered by a constraint class in the metadata. The data warehouse can be analyzed on the frequent pattern tree of user access paths on the web site within a period and duration. The result is an online analytical mining path traversal pattern. Performance studies have been done to demonstrate the effectiveness and efficiency of the system with the following contributions: an architecture of online analytical mining using frame model metadata, a methodology of implementing the online analytical mining, and the resultant cluster of web pages frequently visited by users for marketing use.

Figure 1: The Knowledge Discovery and Data Mining process



INTRODUCTION

Today, with the advent of the web and electronic commerce, nearly every organization has a web site where tremendous amounts of customer data have been generated and collected. These customer data contain a wealth of potentially accessible information. However, the explosive growth of data will inevitably lead to a situation such that it is increasingly difficult to access the desired information. As a result, there are great demands for analyzing data and transforming them into useful information and knowledge. Therefore, Knowledge Discovery and Data Mining (KDD) has become an important field in recent years to address the need for analyzing data in very large data repositories.

KDD is the process of automatic extraction of implicit, novel, useful, and understandable patterns in large databases. There are many steps in the KDD process, which include data selection, data cleaning, enrichment, coding, data-mining task, algorithm selection, and interpretation of discovered knowledge (Adriaans & Zantinge, 1996). This process tends to be interactive, incremental and iterative. Figure 1 illustrates the steps of the knowledge discovery process.

There is a relationship between the activities of data mining and data warehouse – the architecture foundation of decision support systems. The data warehouse sets the stage for effective data mining. The data mining can be done without data warehouse, but the data warehouse can improve the chances of success in data mining (Inmon, 1996).

Background

As the usage of the World Wide Web explodes, a massive amount of data is generated by web servers in the form of web access logs. It is a rich source of information for understanding web user surfing behavior. Web usage mining is one type of web mining activity that involves the automatic discovery of user access patterns on one or more web servers. Also, it applies data mining algorithms to web access logs to locate the regularities in web users' access patterns.

Analysis of these access data provides useful information for server performance enhancements, restructuring web sites, and direct marketing in electronic commerce. As a result, web usage mining has been used in improving web site design, business and marketing decision support, user profiling, and web server system performance, etc.

Among methods of discovering various knowledge in large databases, the association rule has attracted great attention in database research communities in recent years (Agrawal, Imielinski & Swami, 1993; Agrawal & Srikant, 1994; Brin, Motwani & Silverstein, 1997; Han & Fu, 1995; Klemettinen, Mannila, Ronkainen, Toivonen & Verkamo, 1994; Miller & Yang, 1997; Ng, Lakshmanan, Han & Pang, 1998; Park, Chen & Yu, 1995; Srikant & Agrawal, 1995; Srikant & Agrawal, 1996; Savasere, Omiecinski & Navathe, 1995; Srikant, Vu & Agrawal, 1997; Toivonen, 1996). The association rule is a form of data mining to discover interesting relationships among attributes in data. The discovered rules help decision support and business management. An example is that 98% of customers who purchase a computer and printer also buy a scanner. Since rules are simple, easy to understand, explain and catch important relationships among data in large databases. No wonder mining association rules from large data sets has been a popular topic in the recent research of data mining.

The association rule involves several major issues, including efficiency, scalability, usability and understandability. In the real world applications, data mining tasks are applied to data consisting of millions of tuples. Consequently, our first concern is the efficiency and scalability of association rules in large databases to reduce the computational complexity of the intensive data processing. Thus an essential issue in the association rule is to locate its effective algorithms.

The Frequent Pattern Growth (FP-growth) algorithm is one of the association rule algorithms which locates frequent itemsets, but unlike Apriori, it avoids the expense of generating only candidate itemsets. Because FP-growth does not need to examine both candidate and non-candidate sets and requires only two scans of the database, it is a fast algorithm for mining association patterns. We will investigate this algorithm in depth in the algorithm of Sequential FP-growth.

We propose and develop an interesting method, called online analytical mining of path traversal patterns, which integrates the recently developed data warehouse technology with an efficient association mining method. The system stores the derived web user access paths in a data warehouse and facilitates its view maintainability by frame metadata (Fong & Huang, 1997). The system updates user access paths patterns with the data warehouse by the data operation functions in the frame metadata. Whenever a user access path occurs, the view maintainability is triggered by a constraint class in the frame metadata. The data warehouse is analyzed on the frequent pattern tree of user access paths on the web site within a period. The developed method achieves incremental, extensible, and multi-dimensional association rule mining with high performance.

Association Rules

Association rules are like classification rules. Mining association rule is a form of data mining used to discover interesting relationships among attributes in those data. This methodology discovers interesting associations or correlation relationships among a large set of data, i.e., identifies sets of attribute-values (predicate or item) that frequently occur together, and then formulates rules that characterize these relationships. In general, an association rule indicates that the data occurrences of $A_1, A_2, \ldots, A_i$ will most likely associate with the data occurrences of $B_1, B_2, \ldots, B_j$.

$$A_1, A_2, \ldots, A_i \rightarrow B_1, B_2, \ldots, B_j$$

where A_i and B_j are predicates or items. Such rules are usually interpreted as, “ When items $A_1, A_2, \ldots, A_i$ occur, items $B_1, B_2, \ldots, B_j$ will occur as well in the same transaction.”

Association rules have two important measurements: Support and Confidence. Support is an argument that decides whether the candidate is frequent or not. The frequent path patterns are identified by their support values. Confidence is an argument that describes the believable degree of association rules.

$$\text{Support} = \frac{\text{Record Count}(A1 \wedge A2 \wedge \ldots \wedge Ai \wedge B1 \wedge B2 \wedge \ldots \wedge Bj)}{\text{Total Record Count of Source Data}} * 100\%$$

$$\text{Confidence} = \frac{\text{Record Count}(A1 \wedge A2 \wedge \ldots \wedge Ai \wedge B1 \wedge B2 \wedge \ldots \wedge Bj)}{\text{Record Count}(A1 \wedge A2 \wedge \ldots \wedge Ai)} * 100\%$$

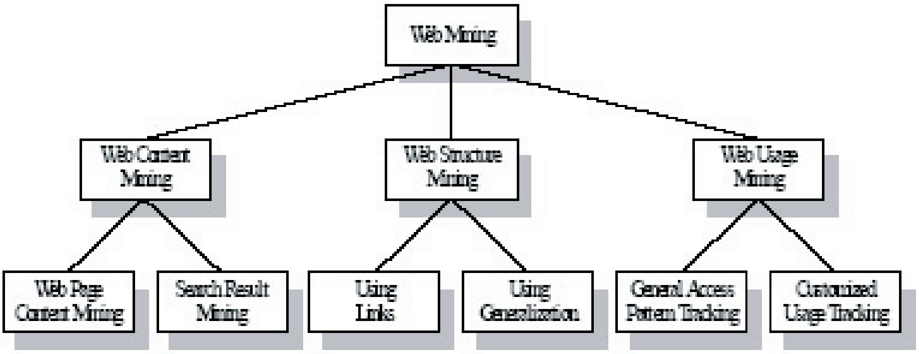
An example of an association rule is, “90% of transactions that contain beer also contain diapers; 5% of all transactions contain both of these items.” Here 90% and 5% are called the confidence level and support level, respectively. The objective is to find association rules that satifsfy user-specifed minimum support and minimum confidence threshold. A strong association rule will have a large support and high confidence level.

Web Mining

The World Wide Web serves as a huge, widely distributed, global information service center for news, advertisements, consumer information, financial management, education, government, e-commerce, and many other information services. The web contains a rich and dynamic collection of hyperlink information and web page access and usage information, providing rich sources for data mining (Han & Kamber, 2001). Naturally, a combination of the data mining and the World Wide Web are referred to as web mining.

Web mining is broadly defined as the discovery and analysis of useful information from the World Wide Web. It describes the automatic search and retrieval of information and resources available from online databases or web servers. In general, there are three knowledge discovery domains pertaining to web mining, which are classified into the

Figure 2: Taxonomy of web mining



following categories: (1) web content mining, (2) web structure mining, and (3) web usage mining. The taxonomy of web mining is depicted in Figure 2.

Briefly, web content mining is the process of extracting knowledge from the contents of documents or their descriptions. Web structure mining is the process of inferring knowledge and links between references and referents in the web. Finally, web usage mining is the process of extracting interesting patterns in web server logs (Cooley, Mobasher & Srivastava, 1999). Alternatively, web mining can be classified into web content mining and web usage mining, because web structures can be treated as a part of web contents mining.

Frame Model Metadata

A frame model is an object-oriented-like database that structures an application domain into classes and its data into relational tables. These classes are organized via generalization, aggregation and user-defined relationships. The frame model is significant as it consists of

Figure 3: The logical schema of the frame model in class format

Header Class {
Class_name
Parents
Operation
Class_type
}
	// an unique name in all system
	// a list of superclass names
	// program call for operations
	// active or static class

Attribute Class {
Attributes_name
Class_name
Method_name
Attributes_type
Association attribute
Default_value
Cardinality
Description
}
	// an attribute in this class
	// reference to header class
	// a method in this class
	// attribute data type
	// pointer to another class
	// predefined value
	// single or multi-valued
	// description of the attributes

Method Class {
Method_name
Class_name
Parameters
Method_type
Condition
Action
}
	// a method component in this class
	// reference to header class
	// number of arguments for the method
	// return type of method
	// the rule conditions
	// the rule actions

Constraint Class {
Constraint_name
Class_name
Method_name
Parameters
Ownership
Event
Sequence
Timing
	// a constraint component of this class
	// reference the header class
	// constraint method name
	// number of argument for the method
	// the class name of method owner
	// triggered event: create update or delete
	// method action time: before or after
	// the method action timer

a set of data, as well as the active and dynamic data structure of the legacy data models, with the constraints structures to resolve their synonyms and homonyms conflicts. These constraints include integrity constraint enforcement, derived data maintenance, triggers, protection, version control, etc. The frame model unifies data and rules, allowing these advanced features to be implemented effectively.

The frame model performs in the object-oriented paradigm. All the conceptual entities are modeled as objects. The same attribute and behavior objects are classified as a class. Besides, both facts and rules are viewed as objects in the frame model design. The frame model logical schema in a class format is shown in Figure 3 (Fong & Huang, 1997).

The frame model consists of two classes: static classes and active (dynamic) classes. Static classes represent factual data entities and active classes represent rule entities. An active class is event driven, obtaining data from the database when it is invoked by a certain event. The static class stores data in its own database. The two classes use the same structure. Combining these two types of objects within the inheritance hierarchy structure enables the frame model to represent hybrid knowledge.

Fong and Huang (1997) translated existing data models into a frame model of the universal database. The structure of the frame model consisted of several classes such as Header, Attributes, Methods, and Constraints classes. According to the frame model, a universal database could be formed. Therefore, old and new database systems could coexist to form a data warehouse for a decision support system.

Fong and Huang (1999) investigated architecture of universal data warehousing for the connectivity of relational and OO data model using an ORDBMS. A frame model metadata was chosen to represent the conceptual and logical schema of the universal data warehouse, which structures an application domain into classes, and its data in relational tables. The universal data warehouse, using an ORDBMS, offers a relational and an OO view for the data warehouse to accommodate different types of queries efficiently. Fong & Pang (1999) proposed a frame metadata model approach to integrate existing databases and evolve them to support new database applications. This facilitates an evolutionary approach to integrating existing databases to support new applications.

Data Warehousing and Star Schema

A data warehouse is a database specifically created to facilitate decision-making. A data warehouse retrieves data from operational and Online Transaction Processing (OLTP) system, but the data are transformed and optimized for analysis.

Nowadays, the demand for information continues to increase as companies realize that information generates revenues, reduces cost and enlarges market shares. Keen competition in rapidly changing business environments is expected and these conditions will generate increasing demand for reliable, easy-to-access decision-making information.

A star schema is a simple structure with relatively few tables and well-defined join paths. This design provides fast query response time and a simple schema that is understood by the analysts and end users. A star schema contains two types of tables: fact tables and dimension tables. Fact tables contain the quantitative or factual data about a business, the information being queried. This information is often numerical measurements and consists of many columns and millions of rows. Dimension tables are usually small verse fact tables and contain more descriptive information. Dimension tables contain the data needed to place transactions along a particular dimension.

The advantage of the star schema is the queries that it can handle in an efficient way. For example, analyzing the sales data by year, quarter, or month is possible on the time dimension without resorting to a table scan. Similar varieties of analysis are possible on the other dimensions. Furthermore, the star schema matches well to the way that users perceive and use the data, making it easier to be understood.

Online Analytical Mining

Online analytical mining (OLAM) is an architecture that integrates online analytical processing (data warehouse) with data mining. Using OLAM can preserve high quality of data in data warehouse, since most data mining tools need to work on integrated, consistent, and cleaned data, which requires costly data cleaning, data transformation and data integration as preprocessing steps. A data warehouse constructed by such preprocessing serves as a valuable source of high-quality data for data mining. Also, comprehensive information processing and data analysis can be systematically constructed surrounding data warehouses, which includes accessing, integration, ODBC DB connections, web accessing and reporting. Finally, OLAM provides facilities for data mining on different subsets of data and at different levels of abstraction, by roll-up and drill-down. This, together with visualization tools, greatly enhances the power and flexibility of exploratory data mining.

Nowadays, most web usage analysis tools lack the ability to provide true business insights about visitors' online behavior. Tools like Accrued, NetTracker and WebTrends provide only high-level predefined reports about frequent count. The reports predefined by the tools are the summary of hits, bytes transferred, a list of top requested URLs, hits per hour/day/week/month report, etc. These reports aim at providing information on the activity of the server rather than the user. Also, they do not concern the incoming and up-to-date log data. Therefore, the derived information will be outdated soon.

Keen competition in rapidly changing business environments is expected, and these conditions will generate increasing demand for reliable, up-to-date and easy to access decision-making information. Therefore, we propose a web usage mining system of OLAM of path traversal patterns for web measurement, which provides an online and up-to-date browsing capacity of user access behavior in a web site. It provides insight into the user behavior, detects and analyzes user access paths for a better understanding of how users visit a web site.

OLAM provides the basic summary reports. It integrates data mining with data warehouse, which provide online capture of the user access patterns. We choose frame metadata to implement the online feature because it can be used to develop an event-driven active data warehouse. When an event occurs, it triggers a process in the constraint class, which calls for the operations in the method class for action. Since the up-to-date view maintenance of the data warehouse is very important, by using frame metadata, data can be actively updated to maintain the view for decision support systems. The result is an active data warehousing view maintenance.

The following covers OLAM of path traversal patterns for web measurement, which includes scalable and continuous/incremental data mining and integration of data mining with database systems and data warehouse systems:

1. Architecture of OLAM using frame model metadata, which utilizes up-to-date view maintenance by continuously updating the data warehouse to interactively extract implicit knowledge from the web access log.

2. Methodology of implementing OLAM, which includes integration of data mining and data warehousing techniques into a unified framework that ensures data availability, flexibility, and integrated information-processing environment for data analysis.
3. The resultant cluster of web pages frequently visited by users for marketing use, which includes identifying potential customers for e-commerce, evolving the web sites to achieve the business objectives, enhancing the quality and delivery of Internet information services to the end user, and helping web design to improve the web site topology.

RELATED WORK

Association Rules Discovery

The concept of association rules was first introduced in Agrawal, Imielinski and Swami (1993). The problem of data mining for association rule has been studied extensively (Harinarayan, Rajaraman & Ullman, 1996; Agrawal & Srikant, 1994; Bayardo, 1998; Cheung, Han, Ng & Wong, 1996; Han, Karypis & Kumar, 1997; Park, Chen & Yu, 1995b; Savasere, Omiecinski & Navathe, 1995; Fukuda, Morimoto, Morishita & Tokuyama, 1996; Svawagi, Thomas & Agrawal, 1998). These studies covered a broad range of topics and its variations have been studied, aimed for further improvements of the performance of the algorithm. These are fast algorithms based on the Apriori Algorithm (Agrawal & Srikant, 1994), incremental updating and parallel algorithms (Cheung, Han, Ng & Wong, 1996; Park, Chen & Yu, 1995b; Han, Karypis & Kumar, 1997), and mining of generalized, multi-level rules, and multi-dimensional rules (Han & Fu, 1995; Zhao, Deshpande & Naughton, 1997). A hash-based technique was used to reduce the size of the candidate k-itemsets; a scan reduction technique was used to reduce the number of database scans; and a transaction reduction technique was used to reduce the number of transactions scanned in future iteration (Park, Chen & Yu, 1995a). Recently, a strategy based on partitioning the data showed a stronger effect than the other scan reduction methods to reduce the number of scans required to two (Savasere, Omiecinski & Navathe, 1995).

Sequential Patterns Mining

The problem of discovering sequential patterns mining is to find inter-transaction patterns such that the presence of a set of items is followed by another item in the time-stamp ordered transaction set. It was first introduced by Agrawal and Srikant (1995). The algorithm AprioriAll was to find all frequent patterns. Later, the same authors (Srikant & Agrawal, 1996a) presented the GSP algorithm that outperforms AprioriAll by up to 20 times. The GSP algorithm was a variation of the Apriori algorithm.

Mannila, Toivonen and Verkamo (1995) presented the problem of finding frequent episodes in only one long sequence of events. An episode is defined as a set of events occurring with a partially defined order and within a given time bound. They generalized their work to allow one to express arbitrary unary conditions on the individual event attributes, or to give binary conditions on the pairs of event attributes. Their experiments were performed using a web server-level log file.

Oates and Cohen (1996) introduced the problem of detecting strong dependencies among multiple streams of data. Their measure of dependency strength is based on the statistical measure of non-independence. An unexpectedly frequent or infrequent pattern was detected, and the algorithm generated rules only rather than frequent sequences.

Another important data dependency that can be discovered, using the temporal characteristics of the data, are similar time sequences (Mannila, Toivonen & Verkamo, 1995; Srikant & Agrawal, 1996b). For example, we may be interested in finding common characteristics of all clients that visited a particular file within the time period [t_1, t_2]. On the contrary, we may be interested in a time interval (within a day or within a week, etc.) in which a particular file is most accessed.

Much work has been done in user behavior analysis. Chen, Park and Yu (1998) explored to mine path traversal patterns in a distributed information environment, but only one ordered dimension, the forward referenced pages/URLs accessed, was considered.

Web Usage Mining

In the recent years, there has been an increasing number of research work done in web usage mining (Yan, Jacobsen, Molina & Dayal, 1996; Cooley, Mobasher & Srivastava, 1997; Chen, Park & Yu, 1998; Wu, Yu & Ballman, 1998; Buchner, Baumgarten, Anand, Mulvenna & Hughes, 1999; Cooley, Mobasher & Srivastava, 1999; Masseglia, Poncelet & Cicchetti, 1999; Masseglia, Poncelet & Teisseire, 1999; Masseglia, Poncelet & Teisseire, 2000; Srivastava, Cooley, Deshpande & Tan, 2000).

Most of the existing web analysis tools (Open market web reporter, 1996; Software Inc. Webtrends, 1995; net.Genesis, 1996) provided mechanisms for reporting user activity in the servers and various forms of data filtering. By using such tools, it is possible to determine the number of accesses to the server and the individual files within the organization's web space, the times or time intervals of visits, and domain names and the URLs of users of the web server. However, these tools are designed to deal with low to moderate traffic servers. Furthermore, they provide little or no analysis of data relationships among the accessed files and directories within the web space.

More sophisticated systems and techniques for discovery and analysis of patterns are now emerging. The emerging tools for user pattern discovery use sophisticated techniques from AI, data mining, psychology, and information theory to mine for knowledge from collected data. For example, the WEBMINER system (Mobasher, Jain, Han & Srivastava, 1996; Cooley, Mobasher & Srivastava, 1997) introduced a general architecture for web usage mining. WEBMINER automatically discovered association rules and sequential patterns from server access logs. Chen, Park and Yu (1996) introduced finding *maximal forward references* and *large reference sequences.* These can be used to perform various types of user traversal path analysis such as identifying the most traversed paths thorough a web locality.

Once access patterns have been discovered, analysts need the appropriate tools and techniques to understand, visualize, and interpret these patterns. Examples of such tools include a WebViz system (Pitkow & Bharat, 1994) for visualizing path traversal patterns. Others have proposed using OLAP techniques such as data cubes for simplifying the analysis of usage statistics from server access logs (Dyreson, 1997). The WEBMINER system (Mobasher, Jain, Han & Srivastava, 1996) proposes an SQL-like query mechanism for querying the discovered knowledge in association rules and sequential patterns.

Data Warehousing

A data warehouse is a subject-oriented, integrated, time-variant, and non-volatile collection of data for decision support applications. The construction of a data warehouse with data cleaning and data integration is viewed as an important preprocessing step for knowledge discovery tasks.

The proposal of the construction of a large data warehouse for multi-dimensional analysis is from Codd, who coined the term OLAP for online analytical processing (Codd, Codd & Salley, 1993). Portions of data warehouses were pre-computed and materialized for efficient processing, and such a materialized multidimensional database is called a data cube (Gray et al., 1997). From the data structure point of view, a data cube is viewed as a large multi-dimensional array which consists of a set of dimensions with respect to the analyzed data, and a set of values in each cell called measures (Chaudhuri & Dayal, 1997). From the operational point of view, a data cube is referred to as a relational operator, which computes group-by aggregations over all possible subsets of the specified dimensions (Gray et al., 1997). It treats each of the *n* aggregated attributes as an n-dimensional sub-cube, or cuboids. The aggregation of a particular set of attribute values is a point in this space. The rapid acceptance of this operator has led to a variant of the CUBE being proposed for the SQL standard.

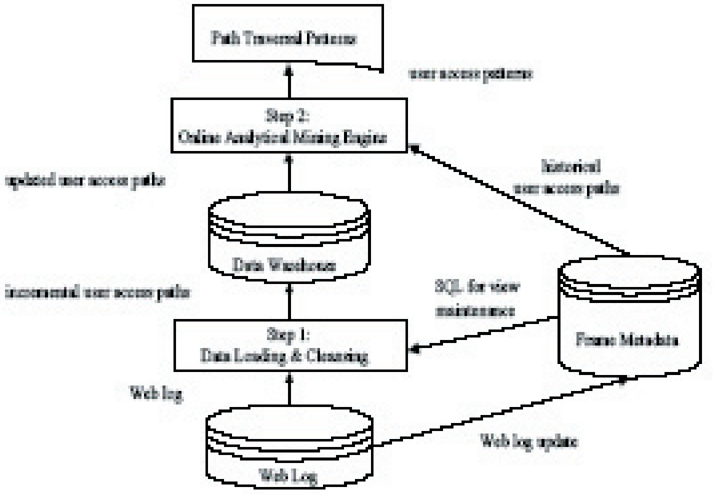
View Maintenance

The view maintenance problem has been studied extensively (Mohania, Madria & Kambayashi, 1999; Zhuge, Molina, Hammer & Widom, 1995; Griffin & Libkin, 1995; Roussopoulos, 1997; Yang, Karlapalem & Li, 1997) and the recent survey of view maintenance literature can be found (Gupta & Mumick, 1995). Ross, Srivastava and Sudarshan (1996) proposed an exhaustive enumerative algorithm for maintaining a view used for any relational algebraic expression, and have shown that the maintenance cost of view is reduced by maintaining a set of additional views along with the original view. Blakeley, Coburn and Larson (1989) found out whether an update to a base relation can affect a derived relation or not. They determined when a derived relation could be updated or not. Segev and Park (1989) considered a problem of maintaining a collection of simple Select-Project views. They developed a screen test procedure to filter out the tuples sent to remote sites. Fong and Zeng (1997) presented a life cycle of developing a data warehouse as: planning, data requirement analyzing and modeling, analytical database design, data mapping and transformation, data extraction and load, automating data management procedures and data validation and testing.

GENERAL ARCHITECTURE OF OLAM

In this section, we present the design and implementation of the online analytical mining of path traversal patterns. It is a simple, scalable and effective method for analysis of web usage. We integrate data mining techniques, and the Sequential FP-growth algorithm with the following aspects: data warehouse, frame model metadata, view maintainability and automated/incremental discovery-driven method for data exploration.

Figure 4: General architecture of OLAM on user access patterns



Overview

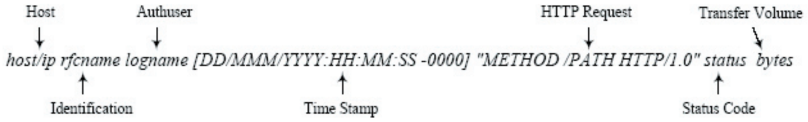
We have developed a general architecture for OLAM of Path Traversal Pattern on web usage (Fong, Wong & Fong, 2000a, 2000b). The architecture divides the web usage mining process into two main parts. The first part includes the processes of transforming the web data into suitable transaction form. This includes preprocessing, user identification, session identification and data integration components. The second part includes the generic data mining and pattern matching techniques such as the discovery of path traversal patterns as part of the system's online analytical mining engine. The overall architecture for the web usage mining process is depicted in Figure 4.

Firstly, the data collected from the web log goes through two steps. In the first step of data preprocessing, data loading and cleansing, the data is filtered to remove irrelevant information (i.e., server request failures, authentication failures, etc.). All entries of the log

Table 1: Services provided by the system

Services	Explanations
Executive summary	General statistics results for the entire time period of the log data.
Path traversal patterns	To mine web user navigation paths to find patterns in the user behavior when traversing a web site.
Requested page summary	Pages access summary such as the most and least frequently requested pages by visitors of a web site.
Date/time summary	Pages access statistics information of the total number of pages viewed for the month, week and day time-intervals.
Entry/exit summary	Pages access statistics information of the entry and exit pages viewed by visitors of a web site.

Figure 5: Common log format


Host
Authuser
HTTP Request
Transfer Volume
host/ip rfcname logname [DD/MMM/YYYY:HH:MM:SS -0000] "METHOD /PATH HTTP/1.0" status bytes
Identification
Time Stamp
Status Code

files are mapped into a relational database. After the data is cleansed, the web log is loaded into a data warehouse (relational database) and new implicit data, like frequency occurrence of access paths and the time spent by each visitor on each page, are calculated. Also the database facilitates information extraction and data summarization based on individual attributes. In the second step, web mining techniques predict and discover interesting user access paths. After the initialization of loading web log into the data warehouse, whenever a user access path is recorded in the web log file, a corresponding update is made to the frame metadata, which triggers the update of user access patterns of web pages online, and generates path traversal patterns. In summary, the system provides the following services as given in Table 1.

Web Access Log

An important source of information about web site visitors is the server transfer log file, known as the access log (web log file). This is where every transaction between the server and browser is recorded with a date and time, the IP address (domain name) of the server making the request for each page on the site, the status of that request, and the number of bytes transferred to that requester, etc. We analyze users' activities on a web site using server log files (access log). There are several kinds of log formats. The most popular one is the Common Log Format (CLF), which was used by most web servers. The common log format appears in Figure 5.

Example: Raw Data of the Access Log

```
144.214.121.52 - - [31/Mar/2001:20:38:11 +0800] “GET /an_cityu.gif HTTP/1.1” 200 90713
144.214.121.52 - - [31/Mar/2001:20:39:31 +0800] “GET /Courses.htm HTTP/1.1” 200 1213
```

Step 1: Date Preprocessing

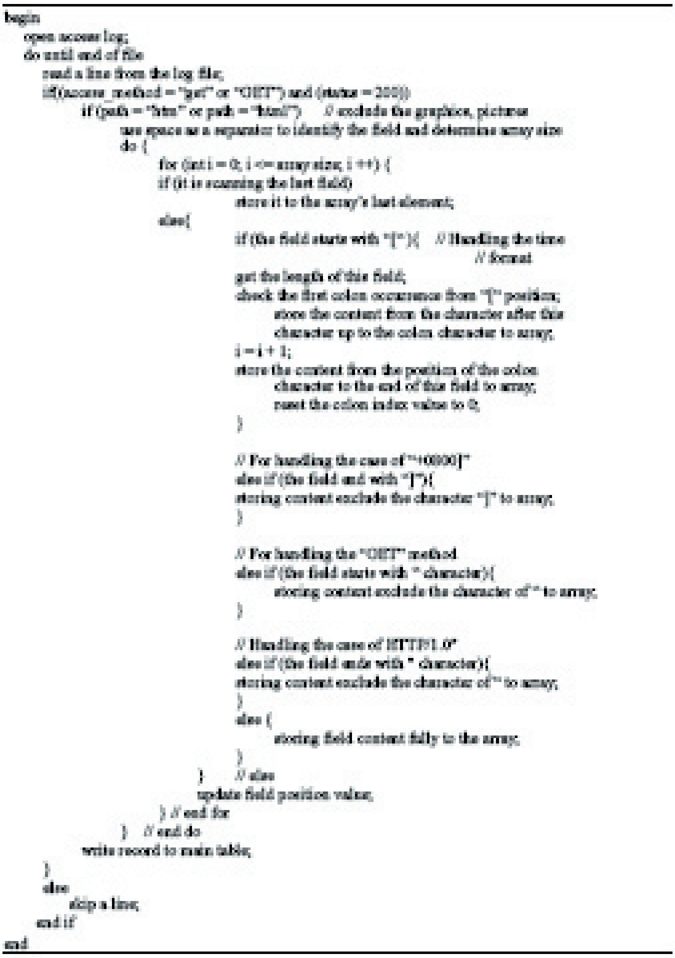
An important step of knowledge discovery is data preprocessing. Since not all the materials within the log file are useful for the mining process, a data preparation process must be performed first. Here we focus on techniques used to preprocess server-level web access log files, namely Common Log Format access log. After the data cleaning, the log entries must be partitioned into logical clusters using one or a series of transaction identification modules, which include user and session identifications.

Step 1.1: Data Loading and Cleaning

Web access log is a plain text file. Therefore it is necessary to identify each field in this file. Each field is separated by a space. Also, some fields are enclosed with special characters such as the double quotation marks, slash or open and close square brackets. Therefore these characters are used to identify what these fields are.

A large proportion of the log file is related to graphics, pictures that constitute the pages and provide no information on the usage of the web site. Data cleaning is the first step performed in the web usage mining process. As web usage mining is investigating the access path sequence made by visitors, all log entries with the picture filename suffix such as ".jpg", ".JPG", ".gif" or ".GIF" in the access path field are removed. Likewise, those records with the filename suffix as "*counter.cgi*" are also eliminated. Moreover, for those records with the methods other than using "GET" (i.e., "PUT", "POST", "HEAD") in the access method field to access the specified file are eliminated. It needs to separate the access

Figure 6: Pseudo-code for data preprocessing



```
begin
    open access log;
    do until end of file
        read a line from the log file;
        if(access_method = "get" or "GET") and (status = 200))
                if (path = "htm" or path = "html")      // exclude the graphics, pictures
                        use space as a separator to identify the field and determine array size
                        do {
                                for (int i = 0; i <= array size; i ++) {
                                if (it is scanning the last field)
                                                store it to the array's last element;
                                else{
                                                if (the field starts with "[" ){     // Handling the time
                                                                                      // format
                                                get the length of this field;
                                                check the first colon occurrence from "[" position;
                                                        store the content from the character after this
                                                        character up to the colon character to array;
                                                i = i + 1;
                                                store the content from the position of the colon
                                                        character to the end of this field to array;
                                                        reset the colon index value to 0;
                                                }

                                                // For handling the case of "+0800]"
                                                else if (the field end with "]"){
                                                storing content exclude the character "]" to array;
                                                }

                                                // For handling the "GET" method
                                                else if (the field starts with " character){
                                                        storing content exclude the character of " to array;
                                                }

                                                // Handling the case of HTTP/1.0"
                                                else if (the field ends with " character){
                                                storing content exclude the character of " to array;
                                                }
                                                else {
                                                        storing field content fully to the array;
                                                }
                                        }       // else
                                        update field position value;
                                } // end for
                        }    // end do
                write record to main table;
        }
        else
                skip a line;
    end if
end
```

Table 2: Cleaned web log data stored in main table

IP Address	Date	Time	URL Request
144.214.36.91	07/May/2001	22:42:04	A.htm
144.214.36.91	07/May/2001	22:45:06	B.htm
144.214.36.91	07/May/2001	22:49:15	D.htm
144.214.36.91	07/May/2001	22:52:44	E.htm
144.214.36.91	07/May/2001	23:40:00	B.htm
144.214.36.91	07/May/2001	23:42:00	A.htm
144.214.36.92	07/May/2001	23:43:05	A.htm
144.214.36.92	07/May/2001	23:46:06	B.htm
144.214.36.92	07/May/2001	23:47:30	C.htm
144.214.36.93	07/May/2001	23:47:50	E.htm
144.214.36.93	07/May/2001	23:48:15	C.htm

time field because separating them makes it easier to compute the time for staying on each page. For the access time field, it contains both the access date and access time, separated by a colon signal (:).

When the web server cannot retrieve those files successfully, the situation is reflected on the value of the status. The value of the status for the successful file retrieval is 200, while that of the unsuccessful retrieval is larger than 400. When the file is reloaded from the web server, the status will be 304. Therefore, those records with the status value other than 200 are eliminated. Moreover, there are some special characters enclosed at the beginning or end of each field. There such characters must be removed before storing the records in the database. Figure 6 shows the pseudo-code for data preprocessing.

After removal of all the irrelevant records from the web log file, the valid records are stored in the main table, as shown in Table 2.

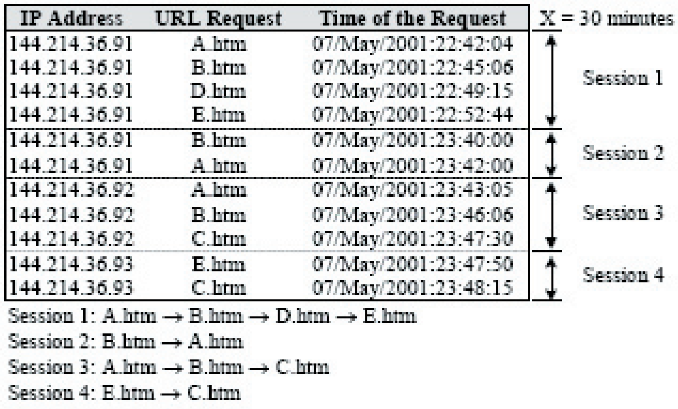
Step 1.2: User Identification and Session Identification

The cleaning techniques discussed earlier are used to preprocess a given web server log. After the data cleaning, the log entries must be partitioned into logical clusters using one or a series of transaction identification modules. In the best case, we rely on the values in fields *rfcname* and/or *logname* to accurately identify a user. But in most cases, fields *rfcname* and *logname* are empty. In the absence of such information, host name/IP information are the only available choices to identify a user. In an ideal scenario, each user is allocated a unique IP address when accessing a web site. However, this is not necessarily correct. For example, some Internet Service Providers (ISPs) randomly assign an IP address to each user's request (dynamic IP assignment); some repeat users access the web each time from a different machine or web browser.

Thus, we use the host name incorporated with user navigation session/user session to identify a user. A user session is all of the pages' references made by a user during a single visit to a web site. Identifying user sessions is similar to the problem of identifying individual users. User interaction within a web site is a collection of user navigation sessions or user sessions whose information is logged in a web server log. A user session is inferred from a web log, which represents a sequence of requests made by the user within a defined time interval.

A user session is therefore defined as a sequence of requests from the same IP address such that no two consecutive requests are separated by more than *X* minutes, where *X* is a

Figure 7: User navigation session inferred from cleaned web log



IP Address	URL Request	Time of the Request	X = 30 minutes
144.214.36.91	A.htm	07/May/2001:22:42:04	Session 1
144.214.36.91	B.htm	07/May/2001:22:45:06	
144.214.36.91	D.htm	07/May/2001:22:49:15	
144.214.36.91	E.htm	07/May/2001:22:52:44	
144.214.36.91	B.htm	07/May/2001:23:40:00	Session 2
144.214.36.91	A.htm	07/May/2001:23:42:00	
144.214.36.92	A.htm	07/May/2001:23:43:05	Session 3
144.214.36.92	B.htm	07/May/2001:23:46:06	
144.214.36.92	C.htm	07/May/2001:23:47:30	
144.214.36.93	E.htm	07/May/2001:23:47:50	Session 4
144.214.36.93	C.htm	07/May/2001:23:48:15	

Session 1: A.htm → B.htm → D.htm → E.htm
Session 2: B.htm → A.htm
Session 3: A.htm → B.htm → C.htm
Session 4: E.htm → C.htm

Figure 8: Algorithm for recording user access paths into data warehouse

```
Given: materialized view V. auxiliary relations $V_1$, ..., $V_n$, data to be updated δR into
data warehouse view and data warehouse view V' after update.

begin
    for record added in log
        extract desired data fields and map into main table;
        if access path exists
            then increment the frequency pattern by 1;
        else
            add the new user access path into fact table;
        end if
    end for

    // V' = V + Applied Group by δR' with Aggregate
    // count by re-computing total and aggregate count
    if δR comes from updates to fact table destination relation
        then V' = V ∪ δR';
    end if
end
```

given parameter. Catledge and Pitkow (1995) have studied user page view time over WWW and have recommended thirty minutes as a reasonable time interval between requests within a user session. Figure 7 illustrates the inferred user sessions from log data.

Step 1.3: Data Warehousing

After finishing the data preprocessing, with the removal of all the irrelevant records from the web log, all the cleaned data are stored in the main table for further process. We store the web usage in a data warehouse such that the log of accessing the target web page and its previous web pages are analyzed as traversal patterns. The possibility of these pages being accessed together is very likely. These web pages of user access paths records are stored in the fact table of the data warehouse, with their dates stored in the dimension table. The algorithm for recording user access paths into data warehouse is shown in Figure 8.

Figure 9 shows the star schema of web usage in access path for an interval in a period. For any user with an UID or IP address, there are many navigation paths for the user browsing the web site. For example, if the access path is P1, P2 and P3 in sequential order, its web page access path becomes from P1 to P2 to P3. *(Note: Frequency pattern count is the number of browsed frequency of the path.)*

We apply the attribute event in the constraint class of the frame model metadata to automate the data warehouse data cube continuously and incrementally. For example, the dimension table and the fact table are as follows:

Dimension table Time relation R_{TIME}		
Year	**Month**	**Day**
Year1	Month1	Day1

Dimension table Time Page relation R_{TARGET}	
Target Page	**Count**
T1	C1

Fact table destination relation R_{FACT} (Date)								
Target Page	**Date**	**CPB**	**CFP**	**FP**	**Count(CPB)**	**Count(CFP)**	**Count(FP)**	**Duration**
T1	Date1	Path 1	Path 2	Path 3	C1	C2	C3	D1

Fact table destination relation R_{FACT} (Month)								
Target Page	**Month**	**CPB**	**CFP**	**FP**	**Count(CPB)**	**Count(CFP)**	**Count(FP)**	**Duration**
T1	Month1	Path 1	Path 2	Path 3	C1	C2	C3	D1

Fact table destination relation R_{FACT} (Year)								
Target Page	**Year**	**CPB**	**CFP**	**FP**	**Count(CPB)**	**Count(CFP)**	**Count(FP)**	**Duration**
T1	Year1	Path 1	Path 2	Path 3	C1	C2	C3	D1

To be updated dimension table tuple δR (data to be updated to data warehouse)

Dimension table Time relation R'_{TIME}		
Year	**Month**	**Day**
Year2	Month2	Day2

Dimension table Time Page relation R'_{TARGET}	
Target Page	**Count**
T2	C2

To be updated fact table update δR (data to be updated to R_{FACT})

Target Page	Date	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T2	Date2	Path 4	Path 5	Path 6	C4	C5	C6	D2

Target Page	Month	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T2	Month2	Path 4	Path 5	Path 6	C4	C5	C6	D2

Target Page	Year	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T2	Year2	Path 4	Path 5	Path 6	C4	C5	C6	D2

If T1 = T2, Date1 = Date2, Path 1 = Path 4, Path 2 = Path 5, and Path 3 = Path 6, then R_{FACT} become:

Updated fact table R'_{FACT} (R_{FACT} after updated) (Date)

Target Page	Date	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T1	Date1	Path 1	Path 2	Path 3	C1 + C4	C2 + C5	C3 + C6	D1 + D2

If they are not equal, it can be simply inserted the new records in the fact table directly.

Target Page	Date	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T1	Date1	Path 1	Path 2	Path 3	C1	C2	C3	D1
T2	Date2	Path 4	Path 5	Path 6	C4	C5	C6	D2

If T1 = T2, Month1 = Month2, Path 1 = Path 4, Path 2 = Path 5, and Path 3 = Path 6, then R_{FACT} become:

Updated fact table R'_{FACT} (R_{FACT} after updated) (Month)

Target Page	Month	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T1	Month1	Path 1	Path 2	Path 3	C1 + C4	C2 + C5	C3 + C6	D1 + D2

If they are not equal, it can be simply inserted the new records in the fact table directly.

Target Page	Month	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T1	Month1	Path 1	Path 2	Path 3	C1	C2	C3	D1
T2	Month2	Path 4	Path 5	Path 6	C4	C5	C6	D2

If T1 = T2, Year1 = Year2, Path 1 = Path 4, Path 2 = Path 5, and Path 3 = Path 6, then R_{FACT} become:

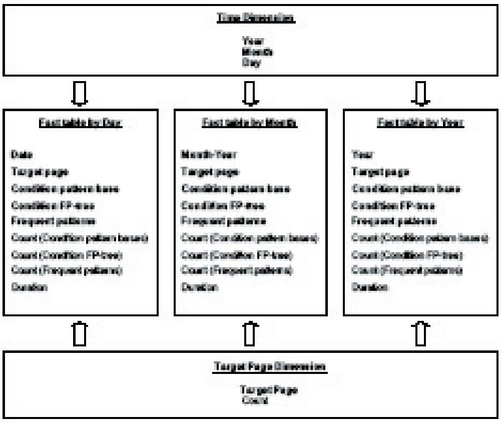
Updated fact table R'_{FACT} (R_{FACT} after updated) (Year)

Target Page	Year	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T1	Year1	Path 1	Path 2	Path 3	C1 + C4	C2 + C5	C3 + C6	D1 + D2

If they are not equal, it can be simply inserted the new records in the fact table directly.

Target Page	Year	CPB	CFP	FP	Count(CPB)	Count(CFP)	Count(FP)	Duration
T1	Year1	Path 1	Path 2	Path 3	C1	C2	C3	D1
T2	Year2	Path 4	Path 5	Path 6	C4	C5	C6	D2

Figure 9: Star schema of frequency pattern count



To get sequential patterns, if the analyzer wants to analyze the web usage within an interval, he/she repeats the analysis on different dates. After accumulating the navigation paths of the analysis, the result is sequential patterns within a period.

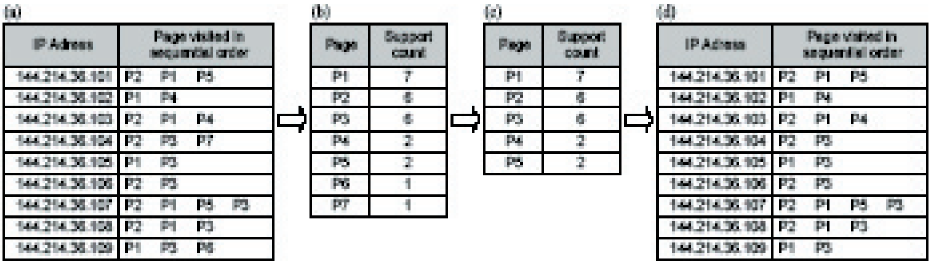
Step 2: Online Analytical Mining Engine

Online analytical mining engine is a major component of the path traversal patterns mining system. Next, we will introduce the general framework of online analytical mining engine. The detailed algorithm is discussed in the following sub-section.

Step 2.1: Sequential Frequent Pattern Growth

Given a web access pattern database, Figure 10 contains a set of pages visited in sequential order. We assume the minimum support threshold is 2. First, the database is scanned to derive a list of frequent items and the occurrences of these items. Remove all

Figure 10: The web log used for FP-tree construction



(a)

IP Adress	Page visited in sequential order
144.214.36.101	P2 P1 P5
144.214.36.102	P1 P4
144.214.36.103	P2 P1 P4
144.214.36.104	P2 P3 P7
144.214.36.105	P1 P3
144.214.36.106	P2 P3
144.214.36.107	P2 P1 P5 P3
144.214.36.108	P2 P1 P3
144.214.36.109	P1 P3 P6

(b)

Page	Support count
P1	7
P2	6
P3	6
P4	2
P5	2
P6	1
P7	1

(c)

Page	Support count
P1	7
P2	6
P3	6
P4	2
P5	2

(d)

IP Adress	Page visited in sequential order
144.214.36.101	P2 P1 P5
144.214.36.102	P1 P4
144.214.36.103	P2 P1 P4
144.214.36.104	P2 P3
144.214.36.105	P1 P3
144.214.36.106	P2 P3
144.214.36.107	P2 P1 P5 P3
144.214.36.108	P2 P1 P3
144.214.36.109	P1 P3

Figure 11: A FP-tree built from the web log

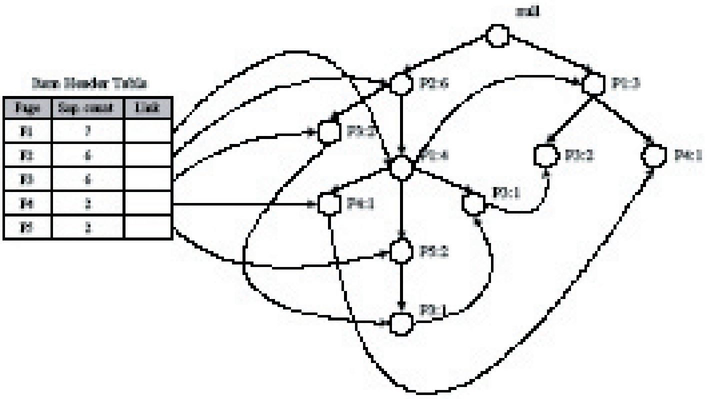
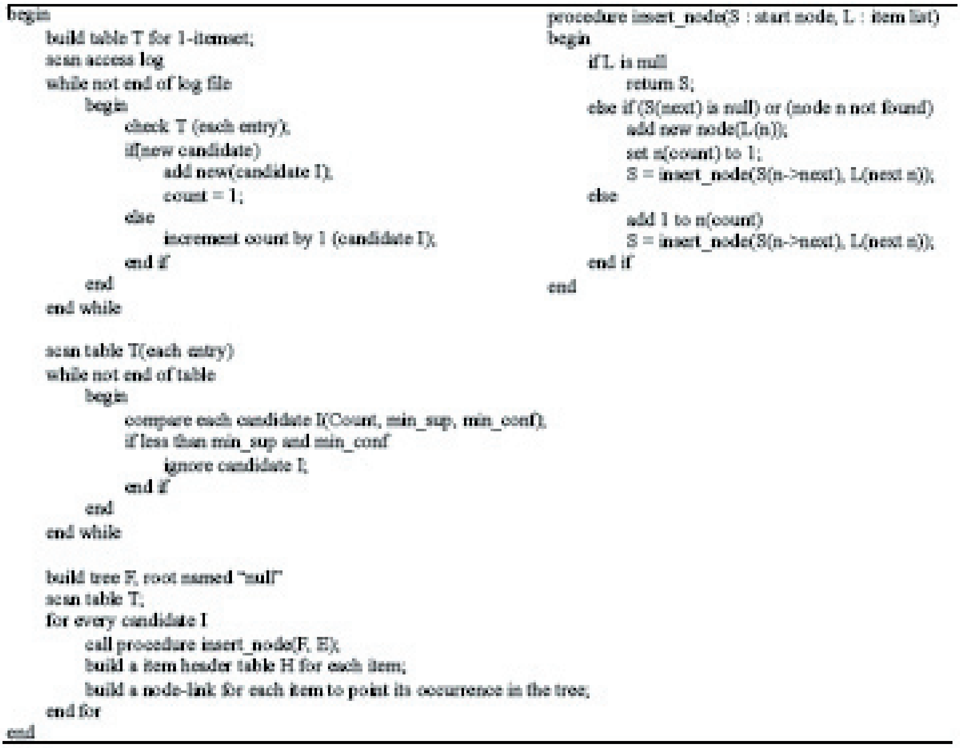


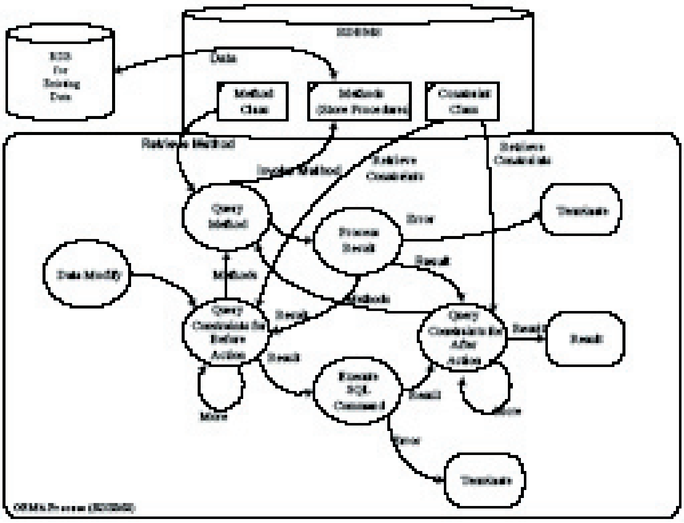
Figure 12: Pseudo-code of sequential frequent pattern growth algorithm



```
begin
    build table T for 1-itemset;
    scan access log
    while not end of log file
        begin
            check T (each entry);
            if(new candidate)
                add new(candidate I);
                count = 1;
            else
                increment count by 1 (candidate I);
            end if
        end
    end while

    scan table T(each entry)
    while not end of table
        begin
            compare each candidate I(Count, min_sup, min_conf);
            if less than min_sup and min_conf
                ignore candidate I;
            end if
        end
    end while

    build tree F, root named "null"
    scan table T;
    for every candidate I
        call procedure insert_node(F, E);
        build a item header table H for each item;
        build a node-link for each item to point its occurrence in the tree;
    end for
end
```

```
procedure insert_node(S : start node, L : item list)
begin
    if L is null
        return S;
    else if (S(next) is null) or (node n not found)
        add new node(L(n));
        set n(count) to 1;
        S = insert_node(S(n->next), L(next n));
    else
        add 1 to n(count)
        S = insert_node(S(n->next), L(next n));
    end if
end
```

Figure 13: Data flow of frame metadata model agent



items that do not satisfy the minimum support threshold. The resulting set or list is denoted L. Thus, we have L = [P1:7, P2:6, P3:6, P4:2, P5:2]. Then, the algorithm creates a tree with a root named 'null'. Next, it scans the database again. The items in each transaction are processed in L order and a branch is created for each transaction.

For example, the scan of the first transaction, "144.214.36.101: P2, P1, P5", contains three visited pages (P2, P1, P5), and leads to the construction of the first branch of the tree with three nodes: <(P2:1), (P1:1), (P5:1)>, where P2 is linked as a child of the root, P1 is linked to P2, and P5 is linked to P1. The second transaction, "144.214.36.102", contains the visited pages P1 and P4, which lead to the construction of the second branch with two nodes: <(P1:1), (P4:1)>, where P1 is linked as a child of the root and P4 is linked to P1. The third transaction, "144.214.36.103", contains the visited pages P2, P1 and P4, which would result in a branch where P2 is linked to the root. P1 is linked to P2, and P4 is linked to P1. However, this branch shares a common prefix, <P2>, with the existing path for "144.214.36.101". Therefore, we increment the count of the P2 node by 1, and create a new node, (P1:1), which is linked as a child of (P2:2). Also we create another new node, (P4:1), which is linked as a child of (P1:1). In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly. The action in reading transactions and tree construction are iterative processed until the last transaction. The tree obtained after scanning all of the transactions is shown in Figure 11. To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links. Therefore, the problem of mining frequent patterns in web log is transformed to that of mining the FP-tree. Figure 12 shows the pseudo-code for sequential frequent pattern growth algorithm.

Step 2.2: Data Warehouse Maintainability using Frame Model Metadata

The frame metadata (Fong & Huang, 1997) consists of two classes: static classes and active. The static class stores data in its own database. It captures the semantics of heterogeneous relational schemas after schema translation. With an object frame metadata model agent as shown in Figure 13, frame metadata can be processed with an object-oriented view and data operation functions. When an event occurs, it triggers a process in the constraint class, which calls for the operations in the method class for action. Data can be actively updated to maintain the view for decision support systems. The result is an active data warehousing view maintenance.

To implement the web usage mining for maintaining user access patterns online, we use frame metadata to update user access paths continuously as follows:

Header class			
Class Name	**Parents**	**Operation**	**Class Type**
V	O	Call Insert_path	Active

Constraint class							
Constraint_ Name	**Method_ Name**	**Class_ Name**	**Parameter**	**Ownership**	**Event**	**Sequence**	**Timing**
Insert_path	Insert_path	V	δR	Self	Insert	After	Repeat

Method class					
Method_Name	**Class_ Name**	**Parameter**	**Method_ type**	**Condition**	**Action**
Insert_path	V	R_S, δR	Tuple	If Code = "GET"	Insert δR into R_S

Consequently, the minimum support and confidence thresholds value must be specified by the analyst as input parameter to build the frequent tree patterns of user access paths, which will derive the user access patterns (path traversal patterns) after data mining. Support and Confidence are two measures of rule interestingness. They reflect the usefulness of certainty of discovered rules. Each measure is associated with a threshold controlled by users or domain experts. Rules that do not meet the threshold are considered uninteresting, and hence are not presented to the user as knowledge. A strong association rule has a large Support and high Confidence level.

APPLICATIONS OF OLAM OF PATH TRAVERSAL PATTERNS

Each query to a web usage mining system returns a set of user navigation paths/patterns. Then the analyst faces the nontrivial problem of evaluating these patterns and deriving reliable conclusions from them. A navigation pattern describes one or more routes among given web pages, along with statistics on how often each page of each route has been accessed. The patterns and statistics provide rules with which the analyst can determine the output of coincidence. By studying this route more closely and comparing it to other routes crossing it, the web designer can detect pages that are not properly designed or linked and redesign them.

Restructuring a Web Site According to the Mining Results

Path traversal patterns discovery helps the web designer in improving the design of web sites. Detecting user navigation paths and analyzing them results in a better understanding of how users visit a site, identifies users with similar information needs, or even improves the quality of information delivery in WWW using personalized web pages.

Also, the sequence of requests by visitors helps predict next requests or popular requests for given days, and thus improves the network traffic by caching those resources, or by allowing the clustering of resources in a site based on user motivation.

Improving Customization

Customization involves learning about an individual user's preferences or interests based on user access patterns. Thus, customization aids in providing users with pages, sites and advertisements that are of interest to them. It may also be possible for web sites to automatically optimize their design and organization based on observed user access patterns.

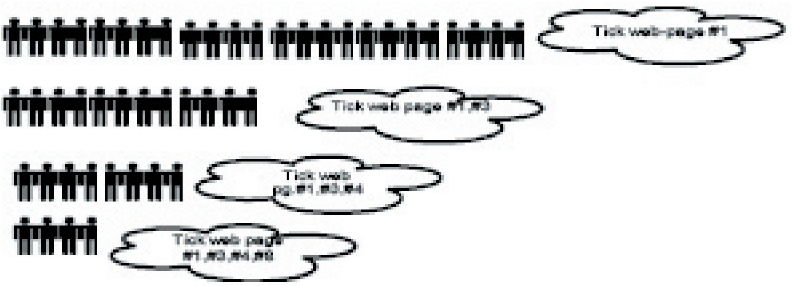
The Impact of Web Advertisements using OLAM

The openness is one of the WWW's biggest advantages. It introduces risk for information security but is also a huge issue in users analysis; not because of its vast volume in eyeball count but its random and extreme pattern of click and tick sequence on the company/institution's web site. We have therefore embedded the value of Confidence and Support level to accommodate these issues of boundary-lessness in our OLAM approach. Although the user of the prototype sets these two values solely based on heuristics, the criterion of optimality in different business domains must always associate with their expertise knowledge. As such, we restrict the user type of our prototype within the Sales Management team of a company or the Public Relation team of an institution whose web site is undergoing the mining process for increasing sales or promoting company images.

We believe that any e-customer can come to the web site and complete an e-service process from beginning to end in a user-friendly and intuitively correct manner. We need to encapsulate all our web site surfers' online experience to discover the knowledge of customer behavior. OLAM creates a list of association rules for each targeted web page determined by the user. The web pages tick sequences are represented in path traversal patterns. These patterns are analyzed to discover users' preferences. The preferred web page(s) can be identified and categorized by Internet surfers and/or e-customers, as shown in Figure 14.

In Figure 14, web page #1 is the most frequently accessed web page. Web advertisement then considers placing on this page. This is the most straightforward way without much need of data mining technique. The OLAM approach offers more. Assume web page #8 is the function page for registration as clients. All UIDs identified in their tick sequence with the visit to web page #8 are grouped as targeted e-customers. There may have been many routes that could link to web page #8. Some users may have sent their registration and placed an order/enquiry, whereas some may have skipped away. The unsuccessful cases are the target

Figure 14: Identify target customers by categorization



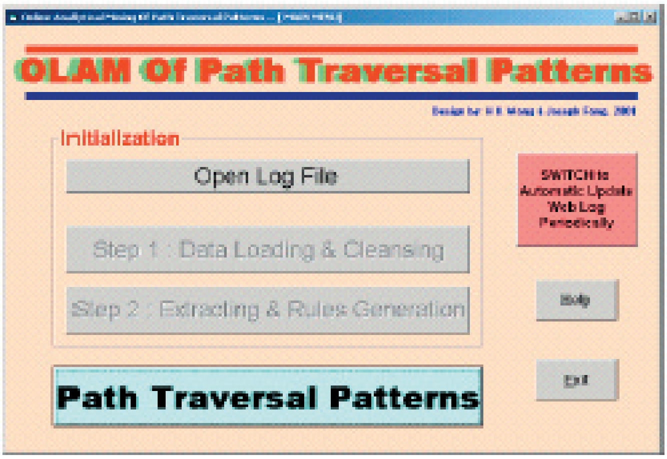
groups that need to be extracted and identified by their UIDs. Their specific path traversal patterns are required for further study. The common web page(s) in all these path traversal patterns that lead to unsuccessful registration are the critical web page(s) that required web advertisement to influence the user behavior and be targeted to change their subsequent path traversal patterns to stay on the web page #8 long enough for registration. Those web pages that never led to page #8 could be considered to be contracted by revision in the web content, merging, consolidation or even elimination, depending on individual cases and further studies on the web page content. Many web pages impressed web surfers with non-focused content or overwhelmed the surfers or e-customers with too much advertising information. The OLAM method could assist in filtering only mission-critical web pages to survive in the ultimate web site infrastructure. As our targeted result is a list of potential e-customers for a certain product or service on a web site, with the associated rules derived, we could trace this related knowledge by further analyzing the main tables in conjunction with the discovered associate rules. We could classify those UIDs by web page sequence. As the key of the main table — identification code tells the UID (User ID), we could identify the target e-customers further. We can even segment the target e-customers not only by their web page preference, but also by their gender, occupation type, income range and age group. As such, more customer-oriented web advertisement(s) could be placed in their preferred web page(s) for more effective marketing.

PROTOTYPE

Here we demonstrate the process of online web usage mining. A university has a home page that contains a lot of useful information (for example, course information, facilities provided, etc.), which is distributed over several sub-pages. The person in charge wants to know which sub-page is more popular and the whether the users who visited a particular sub-page intended to visit other sub-pages. Then the person in-charge can post relevant information or advertisements on the sub-pages more effectively.

The web log file was collected from the Computer Science Laboratory's web sites of City University of Hong Kong. The site hosts a variety of information, ranging from department information and department courses to individual web sites. We are only interested in five pages for analysis, as follows:

Figure 15: Main menu of online analytical mining of path traversal patterns


OLAM Of Path Traversal Patterns
Initialization
Open Log File
Step 1 : Data Loading & Cleansing
Step 2 : Extracting & Rules Generation
SWITCH to Automatic Update Web Log Periodically
Help
Exit
Path Traversal Patterns

Page 1: Department history, facilities and message from department head
Page 2: News, events and seminars notifications
Page 3: Listing of academic staff details
Page 4: Listing of programmes available by department
Page 5: Research groups, research projects, publications, etc.

For simplification, the above five pages are classified as A, B, C, D, and E respectively.

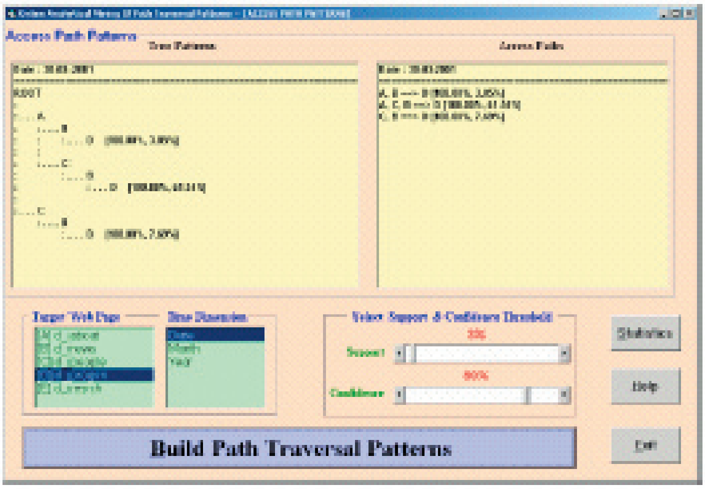
Figure 15 shows the main menu of the OLAM of path traversal patterns. It consists of three parts: initialization, switch to automatic update web log periodically and path traversal patterns. After 'initialization' is executed, a set of potential user navigation paths and user access statistics summary are generated for analysis.

Initialization

The initialization consists of three major components, including: Open Log File, step 1 of data loading and cleansing and step 2 of extracting and rule generation. We can simply click the buttons sequentially and follow the instruction to complete the process.

Figure 16 shows the screen layout of access path patterns. The program will ask users to select/specify several parameters before building the potential user access paths and statistics summary. First, users should select the target web page and time dimension that they are interested in. Also, the two thresholds values, Support and Confidence, can be set according to the user preference. Then by clicking the button, "Build Path Traversal Patterns", a set of potential access paths are generated. There are two windows in the screen. Both show the same information of user navigation paths. One is in graphic form, say FP-tree, while the other one is in text form for easy readability. As a result, analysts can obtain their desired knowledge.

Figure 16: All significant access paths (Confidence = 80% and Support = 3%)



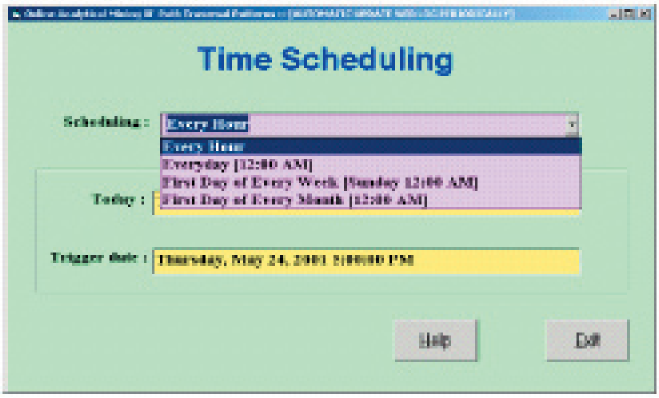
Example 1

The target page 'd_progm' and time dimension "Date" are selected and the confidence and support thresholds were set to 80% and 3% respectively. Then a set of access patterns was generated if its confidence and support levels were greater than or equal to the values inputted by the user. Figure 16 displays the result of the query.

Time Scheduling

Figure 17 shows the time scheduling menu where a user sets the time in which the user accessed path is recorded in the web log file. A corresponding update is made to the frame metadata, which triggers the update of the user access patterns on the data warehouse. As a result, an up-to-date user access patterns is maintained. The system provides four options for time scheduling.

Figure 17: Time scheduling



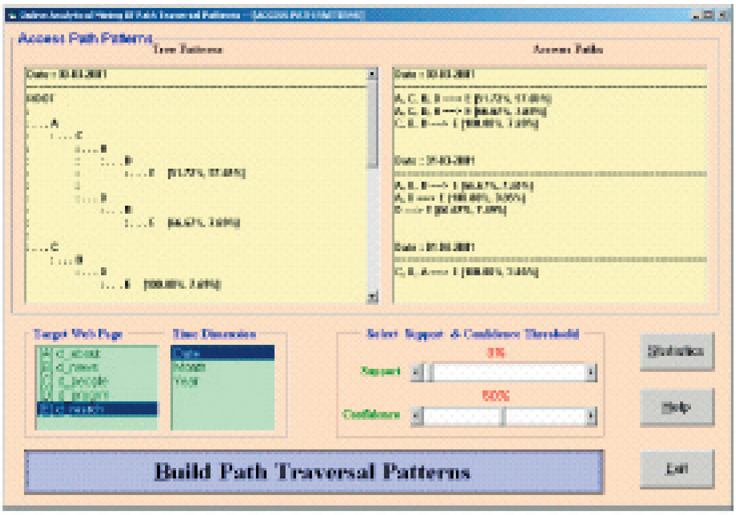
Scheduling : Every Hour

Every Hour
Everyday [12:00 AM]
First Day of Every Week [Sunday 12:00 AM]
First Day of Every Month [12:00 AM]

Today :

Trigger date : Thursday, May 24, 2001 5:00:00 PM

Figure 18: All significant access paths



Online Analytical Mining of Path Traversal Patterns

In web usage, users activities on web sites are recorded into server log files continuously, even though path traversal patterns have been derived before. As a result, the derived path traversal patterns are outdated soon. To maintain the current status of the path traversal pattern, we update the user access patterns continuously or periodically, whenever the log file is being updated. This is accomplished by time scheduling.

Suppose the access log is being updated after a period according to the time set in the time scheduling part. As a result, an up-to-date user accessed pattern has been maintained.

The system provided some Online Analytical Processing (OLAP) functions, including roll up and drill down. The following figures show the up-to-date user access patterns and statistics summary.

Example 2

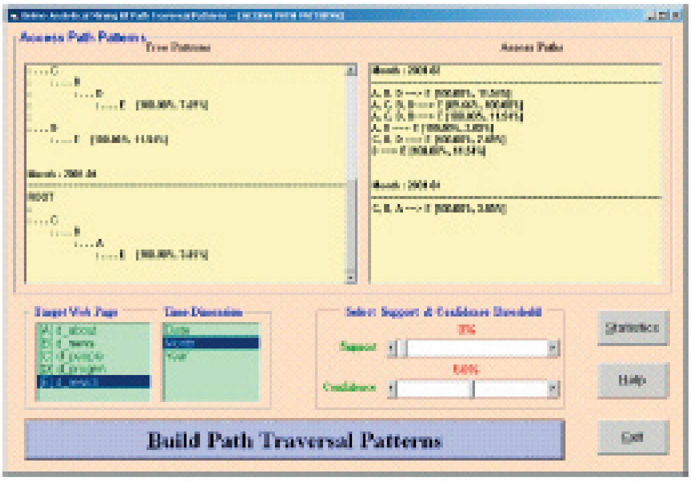
The target web page 'd_resrch' and time dimension "Date" are selected and the Confidence and Support thresholds were set to 50% and 3% respectively. Then a set of access patterns were generated if their confidence and support levels were greater than or equal to the values inputted by the user. Figure 18 displays the result of the query.

Example 3

The target web page 'd_resrch' and time dimension "Month" were selected and the Confidence and Support thresholds were set to 50% and 3% respectively. Then a set of access patterns was generated if its Confidence and Support levels were greater than or equal to the values inputted by the user. Figure 19 displays the result of the query.

Besides the user navigation paths, useful statistics are also provided for analysis. By clicking the button "Statistics", a screen appears. Specifically, the main module of OLAM provides four difference statistics. Executive summary provides a general statistic result for

Figure 19: All significant access paths



the entire time period of the log data. On the other hand, it specifies the time period of the log involved in the system. Requested page summary presents the most and least frequently requested pages by visitors of a web site. Date/Time summary summarizes information about the total number of pages viewed for the month, week and day. Entry page summary presents information about the entry pages viewed by visitors of a web site. Exit page summary presents information about the exit pages viewed by visitors of a web site.

Figure 20 shows the statistics Summary of the web usage mining from the access log.

Figure 20: Statistics summary

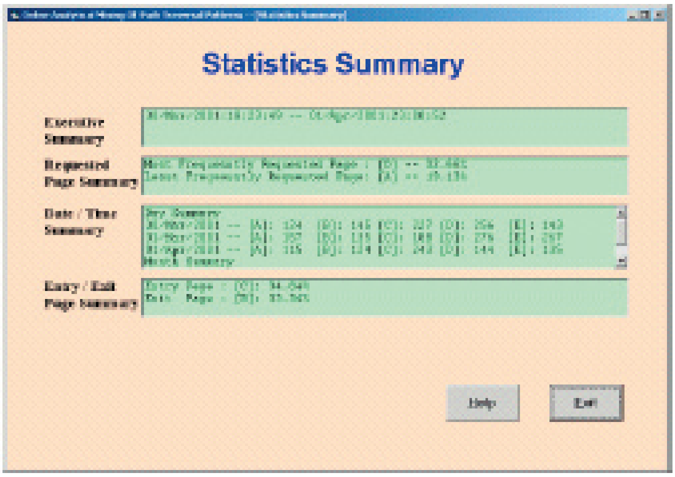
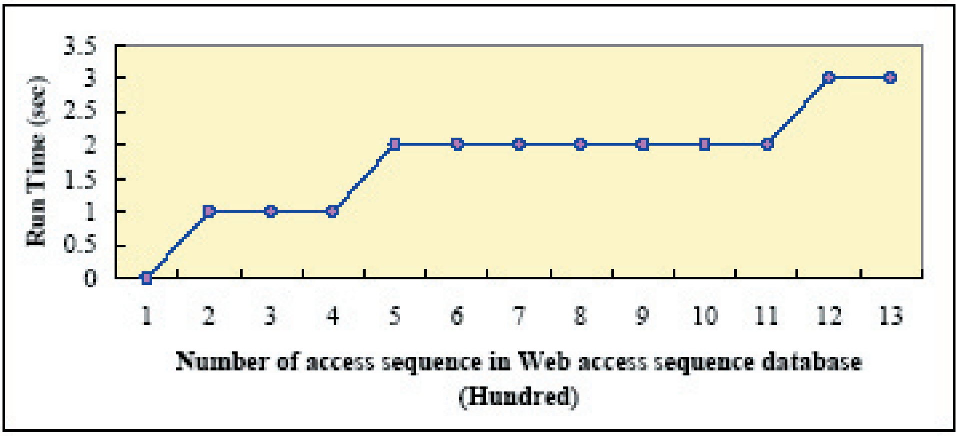


Figure 21: Experimental results



Performance Evaluation

To access the relative performance of the algorithm for discovering path traversal patterns, we performed several experiments on an IBM compatible computer with a Mobile Intel Pentium III CPU clock rate of 750 MHz, 128 Megabytes of main memory, and running Windows 2000 Professional. The data resided in the FAT 32 file system and were stored on a 20 Gigabytes Ultra-ATA hard disk. The relational database is Sybase SQL Anywhere 5.0 for data storage. All programs are written in Microsoft Visual Basic 6.0. The web log covers the year of 2001 and its size is 102 MB.

The experimental result is shown in Figure 21. The FP-tree shows linear scalability with the number of access sequences in the databases. In large databases, it is a good candidate to use for access patterns discovery. In the case study, we are interested in five web pages. The total number of combinations of the traversal patterns is ($_5C_5 + {}_5C_4 + {}_5C_3 + {}_5C_2 + {}_5C_1 = 325$) and the maximum depth of the FP-tree is 5. The FP-tree can be constructed within several seconds even though the numbers of transactions are greater than 10K. Thus, it is very efficient for online analysis purposes. The cost of FP-tree construction is O (| number of frequent items in Transaction| = 5). In general, FP-tree is an effective structure facilitating web path traversal patterns mining.

With certain extensions, the methodology of FP-tree can be applied to perform many web usage mining tasks efficiently, such as web user path traversal patterns mining.

SUMMARY

In summary, an OLAM methodology is proposed to provide the means for management investigation on e-customers' click behavior, so as to further analyze their scale of preference and habit on web site surfing for the web advertisement planning and design. A mechanism of automating the view of the data warehousing has been introduced. The view is provided by joining a dimension table and a fact table, and keeps record of user access paths in a fact table. As the click sequence and path traversal patterns represent the customer's theme, these findings could be translated into web site design and utilized to refine the web site infrastructure. The refinement of the web site design could generate much different patterns of e-customer web pages click sequence. This phenomenon is a cyclic circle. To ensure timeliness, our OLAM method takes a dynamic mining approach for most updated analysis, by providing continued refinement according to the change of the web site environment. However, the problem exists of how to synchronize the update of the based relations with the update of the view. This chapter offers a frame model metadata to facilitate the trigger event, which is invoked whenever an incremental update occurs in the based relation, i.e., access log. The frame model metadata consist of data operation, which is used to update the user access path. As a result, with OLAM, we can transform the data warehousing into an active data warehousing which can activate the incremental data update from the based relation into an existing view, after update during time interval.

The discovery of e-customer click sequence and profile can help in designing a customer-focused web site in the following ways:

1. Make web site functionality intuitive by restructuring it around e-customers' preferred surfing routes and processes. The popular web pages with the most diversified prerequisite sequences and longest surfing time can be identified and refined appropriately with their page content and infrastructure.
2. The isolated and inactive web pages imply that browsers are either incapable of access to it or simply not interested enough to arouse a click. Further analysis on these web page content and their dynamic links are necessary to decide upon whether metaphor on web sites is necessary.
3. Relate utilities[1] to relevant customer actions by easily accessible and visible utilitarian components.

The future direction is to enhance our methodology with association rules established between the UID in the end result click sequence patterns and the UID associated attributes such as the user's personal particulars, for more association semantics discovery. The discovery of targeted customers' personal online preference and offline particulars is an important source for Customer Relationship Management (CRM) to build customer-oriented web sites in the future as follows:

1. Since web log data provide information about what kind of users will access what kind of web pages, web log information can be integrated with web content and web linkage structure mining to help web page ranking, web document classification, and the construction of a multi-layered web information base as well.
2. Sequential pattern mining algorithms tend to generate a huge number of sequences. At any given time, not all of those are of interest to the user. For example, a marketing analyst may only be interested in the activity of those online customers who have visited certain pages in a specific time period. In general, discovered patterns must meet certain rules and conditions. As a result, certain constraints must be integrated with the web mining techniques to get a more reasonable and desired knowledge.

In conclusion, the importance of web usage mining will continue to grow with the popularity of the WWW and undoubtedly will have a significant impact on the study of the online users' behaviors.

(Note 1: Web site functionality allows browsers to do something useful to serve them better and faster. They normally addresses common areas of customer frustration or desire of new/extended activities.)

REFERENCES

Adriaans, P., & Zantinge, D. (1996). *Data mining.* Addison-Wesley.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Proc. 1994 Int. Conf. Very Large Databases* (pp. 487-499).

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proc. 1995 Int. Conf. Data Engineering* (pp. 3-14).

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. *Proc. 1993 ACM SIGMOD Int. Conf. Management of Data* (pp. 207- 216).

Bayardo, R. J. (1998). Efficiently mining long patterns from databases. *Proc. 1998 ACM SIGMOD Int. Conf. Management of Data* (pp. 85-93).

Blakeley, J., Coburn, N., & Larson, P. (1989). Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Transactions on Database Systems, 14*(3), 369-400.

Brin, S., Motwani, R., & Silverstein, C. (1997). Beyond market basket: Generalizing association rules to correlations. *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data* (pp. 265-276).

Buchner, A.G., Baumgarten, M., Anand, S.S., Mulvenna, M.D., & Hughes, J.G. (1999). Navigation pattern discovery from Internet data. *KDD Workshop on Web Usage Analysis and User Profiling* (WebKDD'99), 25-30.

Catledge, L., & Pitkow, J. (1995). Characterizing browsing behaviors on the World Wide Web. *Computer Networks and ISDN Systems, 27*(6).

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM SIGMOD Record, 26*, 65-74.

Chen, M.S., Park, J.S., & Yu, P.S. (1996). Data mining for path traversal patterns in a web environment. *Proc. of the 16th International Conference on Distributed Computing Systems* (pp. 385-392).

Chen, M.S., Park, J.S., & Yu, P.S. (1998). Efficient data mining for path traversal patterns. *IEEE Trans. on Knowledge and Data Engineering, 10*(2), 209-221.

Cheung, D.W., Han, J., Ng, V., & Wong, C.Y. (1996). Maintenance of discovered association rules in large databases: An incremental updating technique. *Proc. 1996 Int. Conf. Data Engineering* (pp. 106-114).

Codd, E.F., Codd, S.B., & Salley, C.T. (1993). Beyond decision support. *Computer World*.

Cooley, R., Mobasher, B., & Srivastava, J. (1997). Web mining: Information and pattern discovery on the World Wide Web. *Proc. of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97).*

Cooley, R., Mobasher, B., & Srivastava, J. (1999). Data preparation for mining World Wide Web browsing patterns. *Journal of Knowledge and Information Systems, 1*(1), 5-32.

Dyreson, C. (1997). Using an incomplete data cube as a summary data sieve. *Bulletin of the IEEE Technical Committee on Data Engineering,* 19-26.

Fong, J., & Huang, S. (1997). *Information systems reengineering*. Springer Verlag, 79-212.

Fong, J., & Huang, S. (1999). Architecture of a universal database: A frame model approach. *International Journal of Cooperative Information Systems, 8*(1), 47-82.

Fong, J., & Pang, F. (1999). Schema evolution for new database applications: A frame metadata model approach. *Proc. of Systems, Cybernetics and Informatics* (Vol. 5, pp. 104-111).

Fong, J., & Zeng, X. (1997). Data warehouse for decision support. *Proc. of the 8th International Database Workshop* (pp. 195-207).

Fong, J., Wong, H.K., & Fong, A. (2000a). Online analytical mining web-pages tick sequences. *Journal of Data Warehousing, 5*(4), 59-68.

Fong, J., Wong, H.K., & Fong, A. (2000b). Online analytical mining association rules on web-pages tick sequences. *The Second International Workshop on Information Integration and Web-based Applications & Services,* 81-94.

Fukuda, T., Morimoto, Y., Morishita, S., & Tokuyama T. (1996). Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. *ACM,* 13-23.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F., & Pirahesh, H. (1997). Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *Data Mining and Knowledge Discovery, 1,* 29-54.

Griffin, T., & Libkin, L. (1995). Incremental maintenance of views with duplicates. *Proc. of the International Conference on Management of Data.*

Gupta, A., & Mumick, I. (1995). Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin, Special Issues on Materialized Views and Warehousing, 18*(2).

Han, E.H., Karypis, G., & Kumar, V. (1997). Scalable parallel data mining for association rules. *ACM,* 277-288.

Han, J., & Fu, Y. (1995). Discovery of multiple-level association rules from large databases. *Proc. 1995 Int. Conf. Very Large Databases* (pp. 420-431).

Han, J., & Kamber, M. (2001). *Data mining: Concepts and techniques.* Morgan Kaufmann Publishers.

Harinarayan, V., Rajaraman, A., & Ullman, J.D. (1996). Implementing data cubes efficiently. *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data* (pp. 205-216).

Inmon, W.H. (1996). The data warehouse and data mining. *Communication of the ACM, 39*(11).

Klemettinen, M., Mannila, H., Ronkainen, P., Toivonen, H., & Verkamo, A.I. (1994). Finding interesting rules from large sets of discovered association rules. *Proc. 3rd Int. Conf. Information and Knowledge Management* (pp. 401-408).

Mannila, H., Toivonen, H., & Verkamo, A.I. (1995). Discovering frequent episodes in sequences. *Proc. 1995 Int. Conf. on Knowledge Discovery and Data Mining (KDD'95)* (pp. 210-215).

Masseglia, F., Poncelet, P., & Cicchetti, R. (1999). An efficient algorithm for web usage mining. *Networking and Information Systems Journal, 2*(5-6), 571-603.

Masseglia, F., Poncelet, P., & Teisseire, M. (1999). Using data mining techniques on web access logs to dynamically improve hypertext structure. *ACM SigWeb Letters, 8*(3), 13-19.

Masseglia, F., Poncelet, P., & Teisseire, M. (2000). Web usage mining: How to efficiently manage new transactions and new clients. *Proc. of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'00)* (pp. 530-535).

Miller, R.J., & Yang, Y. (1997). Association rules over interval data. *Proc. 1997 ACMSIGMOD Int. Conf. Management of Data* (pp. 452-461).

Mobasher, B., Jain, N., Han, E., & Srivastava, J. (1996). *Web mining: Pattern discovery from world wide web transactions.* Technical Report TR 96-050, Dept. of Computer Science, University of Minnesota.

Mohania, M., Madria, S., & Kambayashi, Y. (1999). *Self-maintainable aggregate views. Proc. of the 9th International Database Conference* (pp. 306-317).

net.Genesis. (1996). [Online]. Available: http://www.netgen.com

Ng, R., Lakshmanan, L.V.S., Han, J., & Pang, A. (1998). Exploratory mining and pruning optimizations of constrained association rules. *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data* (pp. 13-24).

Oates, T., & Cohen, P.R. (1996). Searching for structure in multiple streams of data. *Proc. of 13th Int. Conference on Machine Learning (ICML'96)* (pp. 346-354).

Open Market Inc. (1996). *Open market web reporter.* [Online]. Available: http://www.openmarket.com.

Park, J.S., Chen, M.S., & Yu, P.S. (1995a). An effective hash-based algorithm for mining association rules. *Proc. 1995 ACM-SIGMOD Int. Conf. Management of Data* (pp. 175-186).

Park, J.S., Chen, M.S., & Yu, P.S. (1995b). Efficient parallel mining for association rules. *Proc. 4th Int. Conf. Information and Knowledge Management* (pp. 31-36).

Pitkow, J., & Bharat, K.K. (1994). *Webviz: A tool for world-wide web access log analysis*. First International WWW Conference.

Ross, K., Srivastava, D., & Sudarshan, S. (1996). Materialized view maintenance and integrity constraint checking: Trading space for time. *Proc. of the International Conference on Management of Data.*

Roussopoulos, N. (1997). Materialized views and data warehouses. *KRDB, SIGMOD Conference,* 316-327.

Savasere, A., Omiecinski, E., & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases. *Proc. 1995 Int. Conf. Very Large Databases* (pp. 32-443).

Segev, A., & Park, J. (1989). Maintaining materialized views in distributed materialized views. *Proceedings of the IEEE International Conference on Data Engineering.*

Software Inc. Webtrends. (1995). [Online]. Available: http://www.webtrends.com

Srikant, R., & Agrawal, R. (1995). Mining generalized association rules. *Proc. 1995 Int. Conf. Very Large Data Bases* (pp. 407-419).

Srikant, R., & Agrawal, R. (1996a). Mining quantitative association rules in large relational tables. *Proc. 1996 ACM-SIGMOD Int. Conf. Management of Data* (pp. 1-12).

Srikant, R., & Agrawal, R. (1996b). Mining sequential patterns: Generalizations and performance improvements. *Proc. 5th Int. Conf. Extending Database Technology* (pp. 3-17).

Srikant, R., Vu, Q., & Agrawal, R. (1997). Mining association rules with item constraints. *Proc. 3rd Int. Conf. Knowledge Discovery and Data Mining (KDD'97)* (pp. 67-73).

Srivastava, J., Cooley, R., Deshpande, M., & Tan, P.N. (2000). Web usage mining: Discovery and application of usage patterns from web data. *SIGKDD Explorations, 1*(2), 12-23.

Svawagi, S., Thomas, S., & Agrawal, R. (1998). Integrating association rule mining with relational database systems: Alternatives and implications. *ACM*, 343-354.

Toivonen, H. (1996). Sampling large databases for association rules. *Proc. 1996 Int. Conf. Very Large Databases* (pp. 134-145).

Wu, K., Yu, P.S., & Ballman, A. (1998). Speedtracer: A web usage mining and analysis tool. *IBM Systems Journal, 37*(1), 89-105.

Yan, T.W., Jacobsen, M., Molina, H.G., & Dayal, U. (1996). From user access patterns to dynamic hypertext linking. *Proc. of the 5th International World-Wide Web Conference* (pp. 7-11).

Yang, J., Karlapalem, K., & Li, Q. (1997). Algorithms for materialized view design in data warehousing environment. *VLDB Conference,* 136-145.

Zhao, Y., Deshpande, P.M., & Naughton, J.F. (1997). An array-based algorithm for simultaneous multidimensional aggregates. *Proc. 1997 ACM-SIGMOD Int. Conf. Management of Data* (pp. 159-170).

Zhuge, T., Molina, H.G., Hammer, J., & Widom, J. (1995). View maintenance in a warehousing environment. *Proc. of the International Conference on Management of Data* (pp. 316-327).

Chapter XVI

Modeling Motion:
Building Blocks of a Motion Database

Roy Gelbard, Bar Ilan University, Israel

Israel Spiegler, Tel Aviv University, Israel

ABSTRACT

The research proposes a model for the representation and storage of motion data that enables the communication, storage, and analysis of patterns of motion, as with spoken and written languages. The basic problem is the lack of a machine-readable motion alphabet. We thus set out to define the elemental components and building blocks of motion, coming up with what we call the motion byte as the basis for a motion language that has words, phrases, and sentences. The binary-based model we develop, which is significantly different from the common "key frames" approach, is also a method of storing motion data. Comparison with a standard motion system, based on key frames, indicates a significant advantage for our binary model.

INTRODUCTION

In presenting a model for the representation of motion data, our goal is to create and define elemental motion building blocks that enable efficient, precise, and modular representation of the facets of motion, and also serve as a basis for the storage of raw motion data in a way that makes them accessible for future analysis and processing. The constraint on such a form as a channel of communication is the lack of a machine-readable alphabet for motion. The two channels commonly used to express and represent linguistic perceptions are the vocal channel of speech and the graphic channel of script. However, communication by language is not restricted to these two channels, and often uses the **motion channel**, in

Table 1: Channels of expression

Vocal Channel (single speaker)	**Graphic Channel (reduced visual channel)**	**Motion Channel (wide visual channel)**
Two dimensions	Two dimensions	Four dimensions
Components appear in sequence	Components appear in parallel	Components appear in parallel
Expression disappears	Expression preserved	Expression disappears
Building blocks = phonemes	Building blocks = script	Building blocks = ?

which expression is represented by patterns outlined in space by various moving parts of the body.

The motion channel, like the graphic one, is also a visual channel. But, while the graphic channel is expressed in **two** dimensions, the motion channel is characterized by **four**: three space dimensions and a time dimension. Moreover, while in the graphic channel features components appear in sequence and an expression can be saved for further processing, in the motion channel components appear in parallel and the expression disappears immediately after communication. Table 1 summarizes the characteristics of the channels.

General building blocks like phonemes, in the vocal channel, or alphabetic signs in the graphic channel, have few parallels in the motion channel. Exceptions are ad-hoc sign systems, such as sign languages for the deaf, some movement notations, robot programming languages and graphic simulation languages.

We aim to model the motion channel by defining a finite set of building blocks to represent the motion space. These building blocks and movement patterns will, we hope, be the basis of a motion language that will lead to computerized "understanding" and analysis of motion texts such as sign languages for the deaf or musical conducting signs. It may also give rise to a natural and convenient dialog on the use of robots and computer graphic animation applications.

The chapter has the following parts. The chapter outlines the problem and surveys other attempts at defining a motion language. Then, it defines the binary-based model and building blocks of motion. A motion database is also discussed. The next section provides a comparison of storage parameters of our model as compared to Life Forms, a key frames animation system. The chapter goes on to outline the prospects and possibilities of a motion language. The final section gives the conclusions and direction for further study of this rich area of enterprise.

THE PROBLEM

To achieve the purpose of this research of representing raw motion data, we identify and formally define the basic building blocks of motion. These must meet the following criteria:

- Refinement, generality and modularity — to enable accurate and flexible expression of the motion channel,

- Economic and feasible storage of motion data by a computer,
- Recognition of a sequence of motion coordinates,
- Definition, orthography and morphology of motion dialects.

Movement notation and sign languages for the deaf constitute first attempts at "motion reading." The flexible notation system called the Laban Notation (LN), invented in 1928 by Rudolf Laban, was at first applicable only to musical partita, but later, it was improved into what is today labeled the "Cinematography Laban". In 1958 Noah Eshkol and Abraham Wachman published a notation for describing movement based on a geometric concept (Eshkol & Wachmann, 1958; Hutchinson, 1960).

Further developments came about during efforts to make use of notations for the purpose of representing motion on the computer. Badler and Smoliar (1979) adopted parameters from Laban's notation in developing a motion simulation system using the "key frames" approach. However, neither the key frames approach nor indeed any model based on key positions, including the LN notation, fully reflects the phenomenon of motion. These approaches are all based on the following assumptions:

1. Motion is a derivative of *positions*. The transition between two sequential positions proceeds at a uniform rate, and through the shortest possible route.
2. Every change of position in space made by a limb reflects movement performed by that same limb. In other words, no distinction is made between movements and dragging and between movements and compensations. Dragging is a passive motion that occurs when a given limb changes its position without changing the internal geometric relation with adjacent links, e.g., the arm moves as a result of motion by the forearm.

We question these assumptions and suggest that they may be relaxed in defining the basic building blocks of motion.

Among the limitations of the present motion notations are a lack of tools for storing and processing the motion "text" and limited accuracy in expressing time and space (Bruderlin & Williams, 1995; Calvert & Chapman, 1982; Earnshaw, Mangnenat-Thalmann, Terzopoulos & Thalmann, 1998; Hodgins, Wooten, Brogan & O'Brien, 1995; Ko & Badler, 1996; van de Pannw, 1996). These limitations stem from the fact these notations are basically *documentation* rather than storage and analysis tools.

These limitations make it difficult to efficiently store and process motion signs and symbols, and to analyze them by means of computer systems. To do this we need to define basic motion building blocks that can pave the way to a comprehensive motion language that has an internal consistency as is common in any language. Recent work focuses on linguistic aspects of the control of robots by means of a motion description language (Egerstedt, 2001). A survey of issues and challenges in motion modeling is given in Agarwal et al. (2002).

In addition to the problem of representation, there is the issue of storage. Motion data is usually attained through a "tracking system" that chooses single motion units at a frequency of up to 5 Khz. Such a database very quickly becomes intolerably large. Thus, for example, the data required to store the movement of 22 links (as with the Life Forms system; see below) takes approximately **1.21 MBits per second**, and this is for representing a *figures frame only*:

22 (space coordinates) * 23 (edge cursors) * 48 (floating representations) * 50 (sampling frequencies) = 1.21 MBits per second!

Clearly, an alternative representation is needed for more economic storage. These two aspects of our study, the building blocks for representation and the storage of motion, are now presented, suggesting a new model for motion representation.

THE MODEL

Our theoretical-conceptual model for the representation of motion data defines motion variables using a *binary approach* (Spiegler & Maayan, 1985). It is the basis for a storage scheme of a motion database.

Building Blocks

Motion is defined as the alteration of geometric relations between two adjacent links, or between a body's links and the ground surface with which it comes into contact. The alteration of the geometric relations between two adjacent links can be measure by three angles referring to three spatial axes: pitch, roll, and spin axes.

- **Pitch angle:** the extent of rotation of the limb/body on a vertical plane.
- **Roll angle:** the extent of rotation about the *longitudinal axis* of the limb/body. This determines the rotational state of the body.
- **Spin:** (turn) angle — the extent of rotation about the vertical axis of the limb/body. This determines the direction the body turns on a *horizontal* plane.

Figure 1 illustrates these terms graphically over a three-dimensional motion space.

Figure 1: Motion axes in three-dimensional space

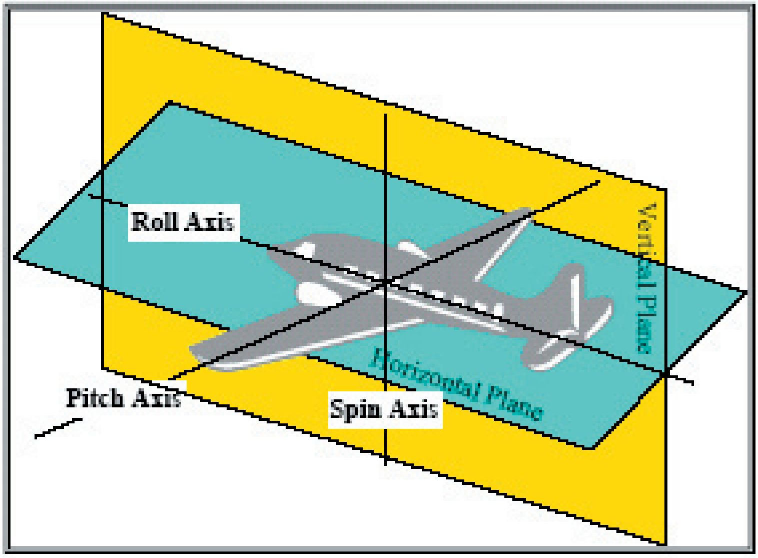
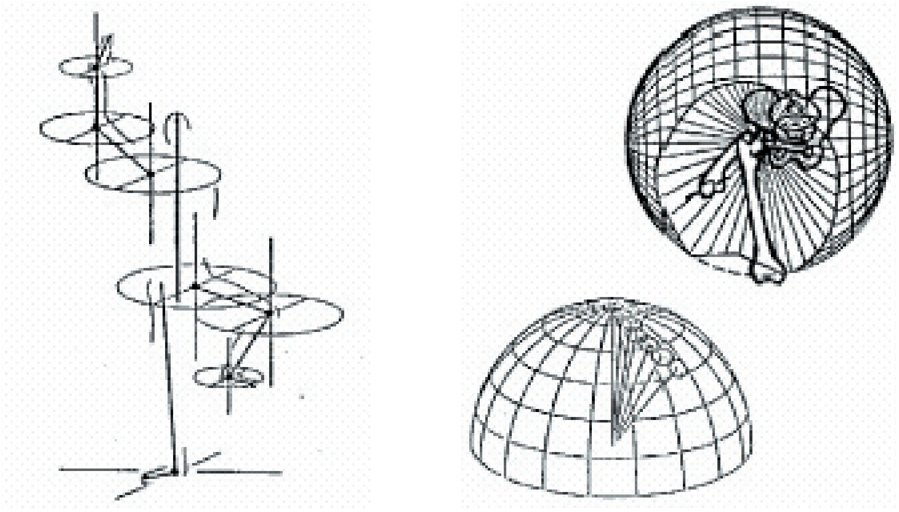


Figure 2: Virtual globes and globographic motion



Conceptually, we assume a virtual globe at each link connecting a joint. We treat a joint, located at the base of every limb, as a permanent motion center. Thus, movement made by a limb takes place on the surface of a sphere, i.e., **every movement outlines an imaginary shoreline on the sphere's surface**. This is shown in Figure 2.

A totally different representation approach is based on indicating the location of every joint in the external space, rather than on relative changes of geometric relations, that is, on the **absolute position** of the body's links. This so-called **key frames** approach to motion representation is very common in computer graphics, as well as in robotics.

Using the definition of motion above, it is possible to represent and store the basic data required for executing motion performed by any vertebrate body using **seven bits per joint per time unit**. This is defined as the **motion byte**. Table 2 depicts this basic 7-bit structure of the motion byte, showing all possible combinations of motion states.

Generally, the motion byte does not need to use all of the 7-bit binary combinations. The "not relevant" cells in Table 2 represent the unused combinations. Moreover, on certain occasions there is no need to use all seven bits of the motion byte, as follows:

- To represent motion between two limbs with a joint that has only *one* degree of freedom, the motion byte consists of only three bits: one bit for the coordination aspect, and two more bits to represent the space aspect (Space 1st D).
- To represent motion between two limbs with a joint that has only *two* degrees of freedom, the motion byte needs only five bits: one bit for the coordination aspect, and four more bits to represent the space aspect (Space 1st D and Space 2nd D).

Thus, the number of bits in the motion byte according to the motion degrees of freedom used by the represented body may be reduced. A creature such as an anthropomorphic

Table 2: The 7-bit motion byte

Coordination	**Space 1st D**	**Space 2nd D**	**Space 3rd D**
Bits 1	**Bits 2 + 3**	**Bits 4 + 5**	**Bits 6 + 7**
1 = motion	1 0 = pitch angle increasing	1 0 = roll angle increasing	1 0 = spin angle increasing
0 = rest	0 1 = pitch angle decreasing	0 1 = roll angle decreasing	0 1 = spin angle decreasing
	0 0 = no change in pitch angle	0 0 = no change in roll angle	0 0 = no change in spin angle
	1 1 = not relevant	1 1 = not relevant	1 1 = not relevant

robot, that is, a robot with joints that have only one degree of freedom, can be represented by a 3-bit motion byte only.

The two bits representing the time aspect are in fact superfluous since the time aspect can be computed on the basis of the time unit rate (sampling frequency). It is possible, however, to extend information on the time aspect (acceleration and deceleration of velocity) as redundancy data. In this case the motion byte will need **nine bits per limb per time unit**. Bits 8 and 9 will represent acceleration (1 0), deceleration (0 1) and no change in velocity (0 0).

Motion Components

Space

The space component of motion with its dimensions and axes is as described above.

Time

While the time component is redundant data, as mentioned above, because it can be computed based on the time unit rate (sampling frequency) by the formula V * T = S, it can be represented by two bits (bits 8-9), as follows:

- Bits 8-9 represent the **velocity** of the motion performed. They express a single unit change in the velocity at which the limb is moving, representing three possible situations: acceleration, deceleration, and maintaining current velocity.

Coordination

The coordination component defines the **state change from rest to motion**, and vice versa; that is, it deals with a body's "**entering** or **exiting**" an active category. The one-bit coordination component is thus used to express two states: **motion** and **rest**. Here, it is necessary to pinpoint precisely the beginning and end of each movement, according to the following guidelines.

- Movement ends ("**exit**") when velocity goes from any value to zero, and the joint's opening angle freezes in place.

- Movement begins ("**enter**") when velocity goes from zero to any other value (even if it is smaller than the basic velocity unit) and the opening angle goes from fixed to any other size.

Thus, in sum, in the representation and storage of motion, there is no need to actually have the time bits, since we can compute time from sampling frequency. That is, using either 3, 5, or 7 bits is sufficient for the **motion byte** to store motion in accordance with the relevant joint degrees of freedom.

Motion Storage

We view the motion database as a three-dimensional cube. This view is not new and is found in the temporal oriented model of data (TOMOD) (Ariav, 1986; Ozsoyoglu & Snodgrass, 1995; Tansel et al., 1993; Etzion, 1998). Applying this view, the sets of the seven motion bits defined above may be seen as a series of two-dimensional tables (a cube) giving a motion database. The cube's measurements are as follows:

- **Objects axis** - the creature's limbs axis.
- **Characteristics/ features axis** - the 7-bit axis.
- **Time axis** - points on this axis represent the different moments at which motion was sampled, producing the basic motion data.

It is therefore possible to view such a cube as a collection of seven two-dimensional tables, each table depicting the current state of one particular characteristic (the state of one of the seven bits), all through the time axis (i.e., the sampling process), for each of the objects (i.e., the creature's limbs). This is illustrated in Figure 3.

Figure 4 illustrates four motion databases built from 3-bit motion bytes. All examples assume the following constants:

Figure 3: Motion database cube

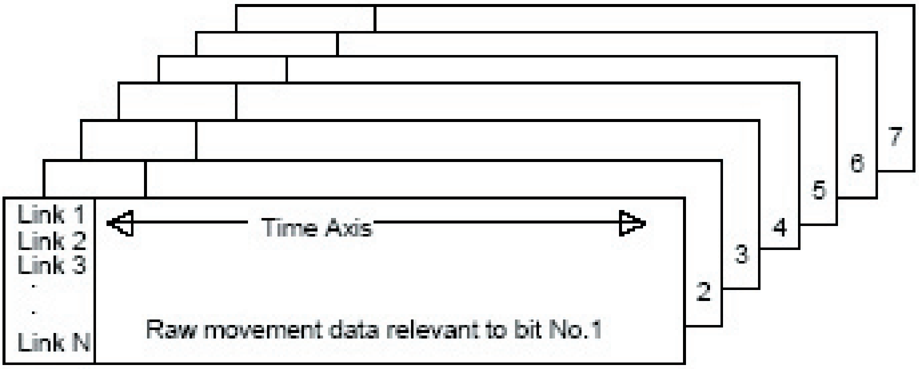


Figure 4: Motion databases built with 3-bit motion bytes

I	Forearm	000 *100 100* **101** *100 100* **101** *100 100* **101** *100 100* **101** 000 *Start 45° 90° 135° 180° End*
II	Forearm	000 *100* **101** *100* **101** *100 100* **101** *100 100 100 100* **101** 000 *Start 45° 90° 135° 180° End*
III	Forearm	000 *100 100* **101** *100 100* **101** *100 100* **101** *100 100* **101** 000
	Arm	000 *100 100 100 100 100* **101** *100 100 100 100 100* **101** 000
IV	Forearm	000 *100 100* **101** *100 100* **101** *100 100* **101** *100 100* **101** 000
	Arm	000 *100 100* **101** *100 100* **101** 000 000 000 000 000 000 000

- Unit scale of 45 degrees.
- Sampling frequency of 12 samples per second.

The bold, underlined and italic notations, as well as the text, are used to emphasize the 3-bit motion bytes representing motion time units.

- **Example *I*:** a 180 degree movement of the forearm (over the shoulder). The movement takes one second and it is performed at constant velocity.
- **Example *II*:** The same movement, but this time the velocity changes with acceleration.
- **Example *III*:** In addition to the constant velocity of the movement of the forearm, as shown in example *I,* here not only is the forearm moving (over the shoulder), but also the arm moves (over the forearm — at the elbow). The movement of the forearm and the movement of the arm start and end at the same time, i.e., the arm performs a movement of 90 degrees at half the speed of the forearm.
- **Example *IV*:** The forearm and the arm start to move at the same time, and with equal velocity. Since the arm performs a movement of 90 degrees, while the forearm performs a movement of 180 degrees, the arm motion ends before that of the forearm.

Constants

In addition to the database, we assume the header of every motion "text" to have global parameters that remain constant during the execution of the motion data. Among the constants are the following:

1. **Anatomy:** the hierarchy and order of limbs, their attributes, measurements, structure and weight, mechanical features, and constraints of movement on joints at the base of each limb.

2. **Starting position:** among these are the initial, starting position of the limbs, i.e., the geometrical relationship between limb positions prior to the start of active movements.
3. **Unit scales:** among these are the unit of change in the angle of every joint's first and second degrees of freedom (measured in partial degree units), the value of the unit of change in the motion velocity of each joint (measured in percentage units), and velocity measurement units (km/hr, cm/min., etc.).
4. **Sampling frequency:** the number of samplings per second.
5. **Motion constants:** i.e., constants, constraints and relationships that remain consistent throughout motion performed by a number of limbs, as for example in the **synergistic** movements of the finger joints of the hands. These are in addition to various parameters contained in the "creature's anatomy" category, which relate separately to each of the creature's individual limbs.

As in any process where analog signals are converted into digital signals (the Nyquist Law (Casavant & Mukesh, 1994; Stallings, 1988) for sampling for the purpose of A/D conversion), the accuracy and sensitivity of the represented motion depend on the rate of sampling used for producing data. Thus, a high sampling rate makes it possible to use very small units of change, and at the same time produce quick movements. For example, a sampling frequency of 5 Khz (5000 samplings per second, as is common in motion simulating systems), allows us to define an *angular unit of change* with a magnitude of 1.08 degree-minutes, at a rate of 90 degrees per second. It also supports using a *unit of velocity change* at a rate of 0.1% to facilitate an acceleration value, for which the velocity increases by a factor of 150 in the span of one second.

Motion Patterns

In the representation and storage of motion, our model make use of *partitas* and *measures* (borrowed from the "language" of music), which employ the binary building blocks to define motion patterns, i.e., a beginning of a motion text.

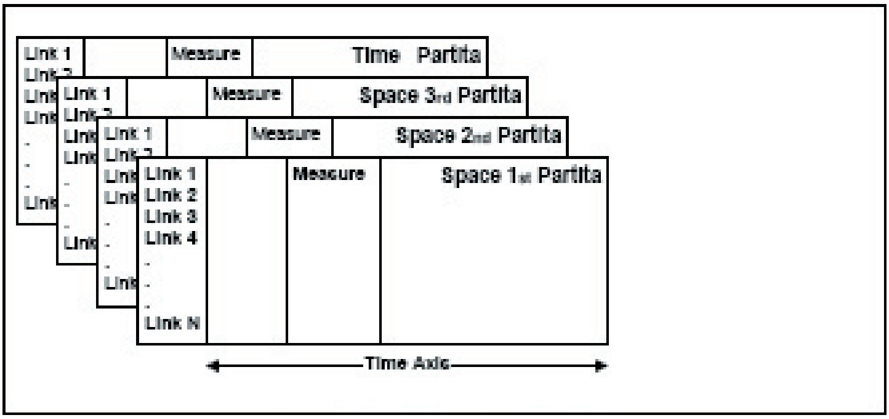
Partitas and Measures

A **partita** is a special structure of motion data with one of the following four formats, each of which comprises the coordination aspect and one space dimension or time dimension (see Figure 5):

1. The space component's first dimension partita - composed of bits 1+2+3
2. The space component's second dimension partita - composed of bits 1+4+5
3. The space component's third dimension partita - composed of bits 1+6+7
4. The time component partita - composed of bits 1+8+9

Despite its redundancy, it is still worthwhile storing the time component. The **generality** of each partita structure is achieved by using the 9-bit structure of the motion byte, even though not all the binary combinations are fully used. As mentioned previously, the coordination component represents the precise moment at which a given limb goes from rest to motion, and vice versa. It delineates starting and ending points of every movement ("enter"-"exit"). This piece of information is necessary in each and every partita.

Figure 5: The partita database cube



As partitas have a common general structure and operate independently on separate components of motion, we can use the same method to examine all of them. Indeed, the partita examination aims to detect **motion patterns**. For this purpose there is an advantage to using partitas over the 9-bit motion byte, because they make it relatively easy to detect repeating structures. These repeating structures are called **measures** in our model.

As already noted, each partita is made up of bit files (rows of three bits). When a certain bit file is repeated over the time axis of a certain link (for example: 001 001 001) it can represent a relevant movement related to the certain link. This is called an **elementary measure**.

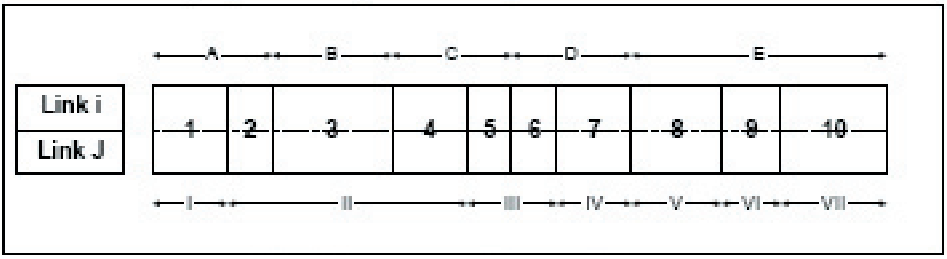
Similarly, a certain array of bit files, that is, the relevant links of a given limb (or its link to the whole body) may repeat over the time axis, representing motion using a **motion pattern.** This is called a **composed measure**. We can relate to a partita as a collection of measures, with no necessity for symmetry in the occurrence of measures in different partitas.

Figure 6 demonstrates the partita of a limb composed of two links (i & j), where:

- Motion of link i contains five elementary measures (letters A - E).
- Motion of link j contains seven elementary measures (numerals I - VII).
  The lowest "common denominator" of the five elementary measures of link i and the seven elementary measures of link j represents the composed measures of the two links. These composed measures are denoted by the numerals 1 - 10, where:
  - A continuous vertical line, at a link's row, represents an **edge** between two elementary measures.
  - A vertical line, across the rows of the two links (whether a continuous or a broken line), represents an edge between composed measures.

Figure 7 illustrates a partita of a limb composed of two links — forearm and arm, that perform the same movement as that demonstrated in Figure 4, line *IV*, where:

Figure 6: Elementary and composed measures



- Motion of forearm contains three elementary measures (between two continuous lines).
- Motion of arm contains three elementary measures (between two continuous lines).

The lowest "common denominator" of the forearm's three elementary measures and the arm's three elementary measures produces four composed measures of the two links. These composed measures are denoted by the numerals 1 - 4.

Each partita can be seen as a collection of measures, where **each measure is actually an ordered structure, made up of sets of three bits**. As already noted, due to the general nature of their structure, each partita can be independently examined.

Figure 8 illustrates five composed measures:

- Measure 1: **ascending structure**. The relevant limb's dimension (velocity or any space dimension) changes in a gradual manner, according to the limb hierarchy, starting with the lightest limb, the one borne by all the other limbs, and concluding with the heaviest limb, the one that lies at the base of the limb structure.
- Measure 2: **a vertical structure**, where all limbs change their positions simultaneously.
- Measure 3: **descending structure**, where the change begins at the base limb, and continues at a steady pace up the hierarchy to the lightest limb.
- Measure 4: **descending graded structure**, which is very similar to the descending messenger structure, except that the time lapse within which changes occur are not uniform.

Figure 7: Binary composed measures

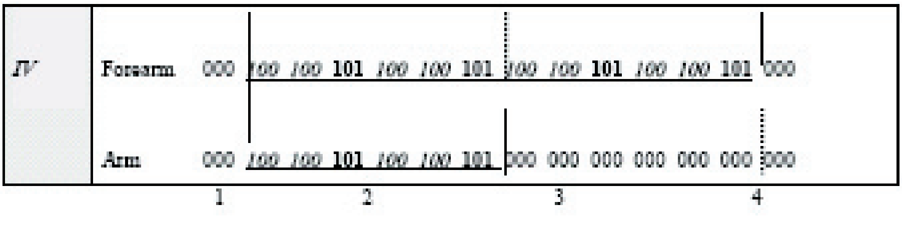
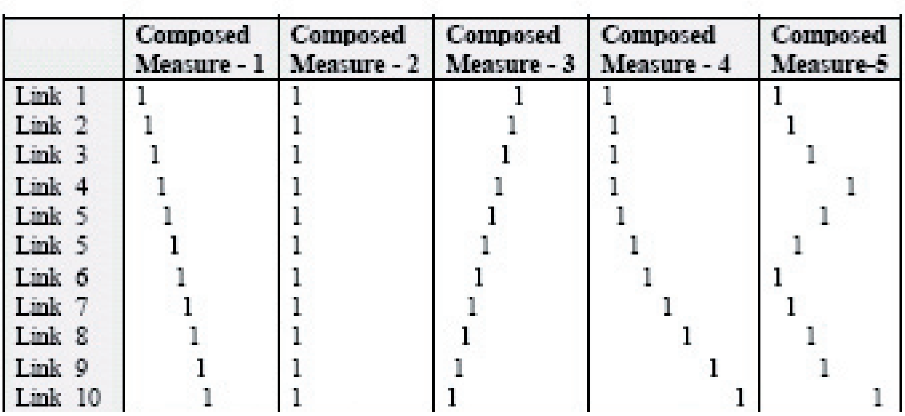

IV
Forearm 000 100 100 101 100 100 101 100 100 101 100 100 101 000
Arm 000 100 100 101 100 100 101 000 000 000 000 000 000 000
1
2
3
4

Figure 8: Motion ptterns — Composed measure structures



	Composed Measure - 1	Composed Measure - 2	Composed Measure - 3	Composed Measure - 4	Composed Measure-5
Link 1	1	1	1	1	1
Link 2	1	1	1	1	1
Link 3	1	1	1	1	1
Link 4	1	1	1	1	1
Link 5	1	1	1	1	1
Link 5	1	1	1	1	1
Link 6	1	1	1	1	1
Link 7	1	1	1	1	1
Link 8	1	1	1	1	1
Link 9	1	1	1	1	1
Link 10	1	1	1	1	1

- Measure 5: **random structure**. This structure should be regarded as a composed measure only if it appears several times in the "motion text"; otherwise it should be regarded as a quasi measure.

An elementary measure is an ordered vector, made of repeated identical bit files. A composed measure is an ordered matrix, in which the matrix rows are elementary measures. The number of matrix rows is equal to the number of relevant links, and the length of the elementary measure is according to the lowest "common denominator".

STORAGE COMPARISON

To learn more about motion storage and performance, we conducted several comparisons between our binary model and a common system. Life Forms, a desktop application developed at Simon Fraser University (Calvert, Bruderlin, Dill, Schiphorst & Welman, 1993), is a representative system based on the key frames approach. A collection of convenient tools for the development of multiple human figure animation, it runs on platforms ranging from a low-end Macintosh up to a high-end Silicon Graphics Iris.

The Key Frames Approach

In the key frames approach, motion is represented by the absolute **position** of every limb while in the binary model storage is based on the internal geometric **relations** between the creature **links**, i.e., on **relative changes**. Position in our model is a derivative of motion. This difference is more than conceptual; it affects storage in terms of data redundancy, as described below.

Movement and Dragging

The key frames approach does not distinguish between movement and dragging. Dragging occurs when a given limb changes its location in space; i.e., it is a change in position, without changing the internal geometric relation with adjacent links. Consider for example the case of the arm in Figure 9, which changes its position in the absolute space, due to movement of the forearm over the shoulder. Here, the forearm moves while the arm is just dragged, without changing the internal geometric relation with the adjacent link, i.e., with the forearm.

Intermediate Position

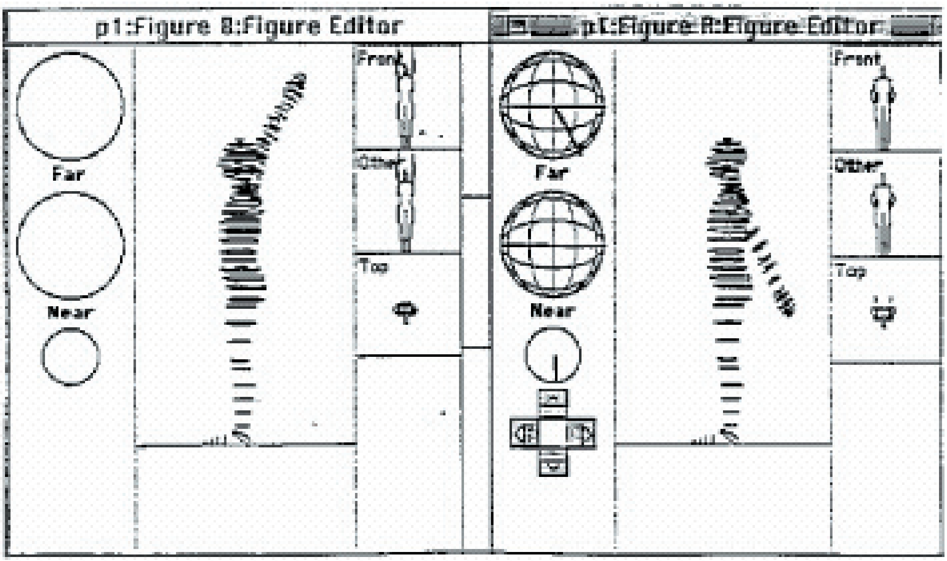
The key frames approach "moves" the creature's limbs along the shortest route between any two sequential positions. Thus, in addition to extreme positions, we are bound to define every **intermediate position** that contributes to the goal of directing the system along the desired path of motion.

Consider for example an upward movement of the forearm of more than 180 degrees along the vertical axis, beginning behind the back and ending behind the head (Figure 9). Using the key frames approach, we need to represent this movement by means of at least three positions. Describing it by means of the two extreme positions only would cause the animation to flow in the opposite direction, using the **shortest path**. In other words, the hand would be raised from behind the back, instead of the front!

In addition, the key frames approach requires an **intermediate position** of the **whole image,** not only in order to direct the movement through the right path (as discussed above), but also in the following cases:

- During the course of movements carried out at varying velocities with respect to different sections.
- During the course of movements in which there is a beginning or ending of movement by any of the limbs involved to allow timing of the movements of the various limbs.

Figure 9: Intermediate position



The Life Forms system consists of many variables designed to represent key positions as frame series, in accordance with the number of frames per second that the user stipulates. Among them are the figure's physical dimensions, presentation mode, stage measurements, user standpoint, location in relation to stage location, position relative to other figures, etc.

The human figure in the Life Forms application is composed of 22 limbs. The physical position is characterized by indicating the location of a reference point in space, and the position of each of the 22 limbs. The reference point represents three coordinates (X,Y,Z) in the spatial region of the stage, applying to the pelvic region of the figure, or alternatively, to the figure's lowest point, i.e., the limb or part of a limb that possesses the smallest Z component. Furthermore, each of the limb positions is defined by means of three angles on the Y, X and Z axes, indicative of the Pitch, Roll and Spin angles.

The Life Forms application uses 69 basic variables to represent the position of an image consisting of 22 limbs, that is, **69 floating point variables** whose storage volume varies according to the computer platform used.

Life Forms *Storage Volume*

Table 3 presents Life Forms variables in a Macintosh environment. The volume of raw data relating to figure animation variables (not including the figure environment) is compared to the volume of raw data required by our binary model by means of sets of 7 bits per joint per time unit of the database cube.

Lines 4 - 7 show the components for defining a position. Their total volume range is 424 - 442 bytes, i.e., 3392 - 3536 bits. It should be noted that these volumes underwent compression by the program. Even if we do not include all the possible location variation, the volumes of all the 69 floating parameters for representation of the figure's limbs in the position can alone reach 4416 bits (69*64 bits per floating variable). Adding 56 bytes to represent the rest of the components (4 - 6) results in **4864** total bits for image position.

Note that the storage volume of the Life Forms application does not vary in accordance with the situations indicated above; it depends on the number of positions defined, rather than on the sampling frequency, motion type, or number of limbs that actually perform the movement. The advantages of the binary model become even more evident when we compare the volume consumed for motion storage, whose representation by means of the key frames approach demands the definition and representation of an intermediate position.

Examples

1. To represent and store **one second** of an arm movement through an angle of more than 180 degrees (as illustrated in Figure 9), three positions are required. Thus, the Life Forms application requires **14,592 bits** (4,864 *3). The binary model, on the other hand, requires between 324 and 1,540 bits (depending on the sampling frequency), an advantage factor of 9.5 to 45.
2. To represent motion where the forearm starts slightly after the arm and finishes the movement slightly earlier, the Life Forms application requires five key frames, that is, **24,320 bits**. The binary model, on the other hand, takes only 324 - 1540 bits (depending on frequency of sampling), giving a comparative factor of 15.8 to 75.
3. Defining three different motion velocities, e.g., starting with acceleration, continuing at constant velocity, and coming to a stop, means added storage. The Life Forms application requires seven positions (key frames), or **34,048 bits**. The binary model

Table 3: Life forms application variables

1	The image environment	2620 bytes	Stage, time line, figure editor, panel
2	Every image	1492 bytes	
3	Frame per second variation	20 bytes	To define motion velocity
4	Change of location	20 bytes	Location change on the horizontal plane
5	Change of altitude	18 bytes	Location change on the vertical plane
6	Change of facing - rotation	18-36 bytes	The rate of change dictates the volume
7	Limb representation in every position	368 bytes	

takes only 324 - 1540 bits (depending on sampling frequency), a factor of 22.1 to 105 in storage saving.

We should note, however, that this velocity definition is limited and distorted, since it ascribes a velocity profile of motion as a whole, instead of a specific velocity profile for each limb separately.

Thus, in a **worst-case** analysis with a sampling frequency of 10 frames per second (FPS) (most animation programs use a 3 FPS default, an animation sufficiently smooth for the human eye), our model needs **1,540 bits per second** (10 samplings per second * 7 bits per limb * 22 limbs). Note that there is no need to use all seven bits for each of the 22 limbs, because few limbs have more than one degree of freedom; that is, most limbs can be represented by a 3-bit byte per time unit. In the examples we used the full 7-bit motion byte in volume consumption calculations, to be on the safe side. In a **best-case** analysis**,** given a sampling frequency of three frames per second, eight limbs with one degree of freedom (i.e., representation by a 3-bit motion byte), seven limbs with two degrees of freedom (represented by a 5-bit motion byte), and seven limbs with three degrees of freedom (represented by a 7-bit motion byte), our model requires **324 bits per second** (3 samplings per second * [8 limbs * 3 bits + 7 limbs * 5 bits + 7 limbs * 7 bits]).

In contrast, to represent and store **one second** of an arm movement through an angle of more than 180 degrees (as illustrated in Figure 9), the Life Forms application requires **14,592 bits.** This gives a ratio of **1:45** in favor of our model (depending on sampling frequency).

These examples, and others performed for the comparison, clearly demonstrate that in the key frames approach storage volume grows significantly in accordance with the number of moving limbs and the velocity of each limb participating in the motion. The binary model, on the other hand, is virtually independent of motion and velocity of the various limbs. The only significant parameters are the number of limbs and the desired sampling frequency. The advantages of the binary model become more evident when we compare the volumes needed for motion storage, whose representation by means of the key frames approach demands the definition and representation of an intermediate position.

MOTION LANGUAGE

While the design of a motion language is beyond the scope of this paper, our work provides some guidelines for producing the “letters”, “words”, and “sentences” leading to a motion text. The examples in this paper show some initial steps: the 7-bit motion byte defines a letter, motion words and sentences are shown in Figures 4, 5, and 6 using the concept of partita, a measure used in music. Note the complex expression shown in Figure 6 describing the motion of two links (arm and forearm) moving in parallel and coordination. The equivalent description in verbal language would hardly reach that level of efficiency.

We illustrate a simple sign language for forearm direction. Two expressions are defined, HALT and SLOW, by means of a single limb motion in the vertical plane only (e.g., right forearm). A 3-bit building block is needed to define this rudimentary language, where:

Bit 1	defines motion (1), and rest (0)
Bits 2+3	defines increase (10), decrease (01), and no change (00) in forearm angle.

Given a base position in which the forearm is at rest, hanging down along the side of the body, we assume:

- A sampling frequency of two samples per second.
- A basic measuring unit of 90 degrees, i.e., a motion change of 90 degrees takes ½ sec.
- H = a HALT message is discharged by a motion of 180 degrees (decrease angle relative to arm), and staying at this position for at least ½ second.
- S = a SLOW message is given by moving the arm 90 degrees (decreasing angle relative to arm), and staying at this position for at least ½ second.
- B = back to base position takes place either from H or S (increasing angle).
- SH = moving the forearm from SLOW to HALT (decrease angle relative to arm).
- Remaining at a certain position (S, H, or Base position) is denoted by a (.).
  There is no meaning to any other motion, e.g., transition from HALT to SLOW.

Table 4 depicts the different levels of the forearm motion language.

Time Scale -	Sets the time in elapsed seconds.
Bit Level -	Describes the binary building blocks stored in the database.
Motion Level -	Shows motion quantities and directions (increase/decrease).
Pattern Level -	Shows information patterns in the motion database.
Language Level -	Gives a linguistic meaning to the motion patterns.

Given a motion language and database, we can begin to look at a "motion query language" (MQL) by defining a two-term component of the movement message (H, S, base) and a number that represents motion duration in time units. The data in Table 4 may be viewed as:

Slow-5
Halt-3
Base-1
And queries may take the form:

```
Select "Slow"
From database-name
Where time is between 0 and 8.
```

Table 4: Illustrating a motion language

Language Level			S	.	.	.	.	.		SH	.	.	.			B	.
Pattern Level		P1							P1					P2			
Motion Level		+90							+90					+90	+90		
Bit Level	000	101	000	000	000	000	000	000	101	000	000	000	000	110	110	000	000
Time Table	**0**		**1**		**2**		**3**		**4**		**5**		**6**		**7**		**8**

The query will return the specific limb in which "Slow" was found. In the example, the reply will be: Forearm, 1,5 where motion took 1 sec. to complete, and that position was held for 5 units, i.e., 5 half seconds.

This is a very rudimentary example and is used only to illustrate the direction and first steps of a full motion language, which is beyond the scope of our current study.

When looking at the sequence of building blocks within the raw motion data, it is possible to identify "measures", i.e., recurring structures. In fact, these structures represent **motion words**. Every motion word can be recursively split into its original building blocks. Moreover, it is possible to write motion economically by ascribing a code to each measure, while noting the proportions and magnitudes of its components. In a well-defined "**motion world**" it is possible to characterize **a finite number of motion building blocks**. The diversity of forms that can be realized using binary bits to represents building blocks of motion is quite high. These building blocks are thus fine, precise, as well as general, modular and generative.

The importance of such motion building blocks and motion words lies in the ability to define a motion language around them. Such a language will enable a significant reduction in the storage volume needed for raw motion data and improve the processing and query capabilities once the data is stored on computer media. It will also enable a natural man-machine interface (by using words, sentences and paragraphs), and computerized pattern recognition algorithms, i.e., recognition of patterns, to be applied to the "motion text". These capabilities are useful and important in many fields: robotics, animation, graphics, athletics, and more (Rose, Cohen & Bodenheimer, 1998).

A full-scale motion language will facilitate the following possibilities:

- A significant reduction in storage volume of raw motion data.
- Improved processing capabilities, as a binary form fits computer internal storage well and can utilize bit-maps in the search and retrieval process.
- Definition of any movement, not only in a fine and modular way, but also in a natural and user-friendly way (using words, sentences, and paragraphs).
- Application of computerized techniques from artificial intelligence, pattern recognition, information retrieval, and database management.
- Ability to make "syntactic" and "semantic" analyses, and to discover the inner rules of a motion text.
- Ability to represent informatively and achieve an artificial understanding of the linguistic perceptions represented by motion text.

SUMMARY AND CONCLUSIONS

A model for the representation and storage of motion data has been presented. The model uses a binary framework by which the various components of motion are identified and represented. In this framework, a 7-bit **motion byte** is defined in its general form, covering space, time, and coordination components.

The motion model also handles the issue of storage, i.e., the ability to store movement data of an object in an efficient and economic way such that the data can be accessed and processed. A comparison between storage requirements of our model and a key frames system suggests significant storage saving, and hence a performance lead by the binary model.

The binary format used for representing and storing motion has several obvious advantages. First, it is a simple and easy way to represent motion over the four dimensions of space and time. Such a representation method is both user-friendly and appropriate for computer storage and processing. Second, we show the economy and efficiency of the model in terms of the volume needed for storing motion data. This is important in light of the high level of storage required by common animation tools that use position and sampling frequencies. Third, the model is **informative** in the sense that there is no loss of information due to any mathematical or other operations performed on the data. Stored in a binary form, motion data is amenable to future computerized retrieval and manipulation. As such, the binary representation is a first step in defining a full motion language, relating to the movement of bodies as in the vocal channel of speech and the graphic channel of script.

The research contributes to several fields, such as knowledge representation and databases. It presents a formal and general model for the representation of raw motion data that can be extended to the imitation of linguistic perceptions expressed through the motion channel.

Several areas are suggested for further study. Since the motion model is represented only by binary parameters, it has substantial raw data compression possibilities (Spiegler & Maayan, 1985; Samet, 1984; Samet & Webber, 1988a; Samet & Webber, 1988b). A detailed analogy between optical character recognition (OCR) and motion processing is needed to extend the pattern recognition discipline to the field of motion. Similarly, an analogy between natural language processing (NLP) and motion processing will provide means of recognizing and understanding motion words. We have shown a direction to define a motion building block. A finite number of precise, general, and generative building blocks can be characterized in a well-defined motion world such as a robot motion environment (Egerstedt, 2001), enabling the construction of words, sentences, and motion text about the said motion world.

Binary representation of motion also opens the way to studies in diagnosis and measurement of physiological phenomenon, such as motor disorders, changes in motor capabilities under the effect of different medical treatments or drugs, athletic training, comparative variations in motor capabilities of different individuals, as well as ways in which they carry out motor tasks.

REFERENCES

Agarwal, P.K. et al. (2002). Algorimic issues in modeling motion. *ACM Computing Surveys, 34*(4), December.

Ariav, G. (1986). A temporally oriented data model. *ACM Trans. on Database Systems, 11*(4), December, 449-527.

Badler, N.I., & Smoliar, S.W. (1979). Digital representations of human movements. *ACM Computing Surveys, 11*(1), March, 19-38.

Bruderlin, A., & Williams, L. (1995). Motion signal processing. *Computer Graphics Proceedings - SIGGRAPH95,* August, 97-104.

Calvert, T.W., & Chapman, J. (1982). Aspects of the cinematic simulation of human movement. *IEEE CG&A*, November, 41-50.

Casavant, T.L., & Mukesh, S. (1994). *Readings in distributed computing systems.* USA: IEEE Computer Society Press.

Calvert, T.W., Bruderlin, A., Dill, J., Schiphorst, T., & Welman, C. (1993). Desktop animation of multiple human figures. *IEEE CG&A,* May, 18-26.

Earnshaw, R., Mangnenat-Thalmann, N., Terzopoulos, D., & Thalmann, D. (1998). Computer animation for virtual humans. *IEEE CG&A*, September/October, 20-23.

Egerstedt, M. (2001). Linguistic control of mobile robots**.** *Proceedings of IEEE Conference on Intelligent Robots and Systems, 2001, Maui, Hawaii,* 877-882.

Eshkol, N., & Wachmann, A. (1958). *Movement notation*. Weidenfeld and Nicolson.

Etzion, O. et al. (Ed.). (1998). *Temporal databases: Research and practice*. Springer.

Hodgins, J.K., Wooten, W.L., Brogan, D.C., & O'Brien, J.F. (1995). Animation human athletics. *Computer Graphics Proceedings – SIGGRAPH95,* August, 71-78.

Hutchinson, A. (1960). *Labanotation*, second edition. New York: Theater Arts Books.

Ko, H., & Badler, N.I. (1996). Animation human locomotion with inverse dynamics. *IEEE CG&A,* March, 50-59.

Ozsoyoglu, G., & Snodgrass, R.T. (1995). Temporal and real-time database: A survey. *IEEE Transactions on Knowledge and Data Engineering, 7*(4), August, 513-532.

Rose, C., Cohen, M.F., & Bodenheimer, B. (1998). Verbs and adverbs: Multidimentional motion interpolation. *IEEE CG&A*, September/October, 32-40.

Samet, H. (1984). The Quadtree and related hierarchical structures. *ACM Computing Surveys, 16*(2), June, 187-260.

Samet, H., & Webber, R.E. (1988). Hierarchical data structures and algorithms for computer graphics - Part I. *IEEE CG&A,* May, 48-68.

Samet, H., & Webber, R.E. (1988). Hierarchical data structures and algorithms for computer graphics - Part II. *IEEE CG&A,* July, 59-75

Spiegler, I., & Maayan, R. (1985). Storage and retrieval considerations of binary databases. *Information Processing & Management, 21*, 233-254.

Stallings, W. (1988). *Data and computer communication*, second edition. Macmillan.

Tansel, C., Gadia, Jajodia, Segev, & Snodgrass. (1993). *Temporal databases: Theory, design and implementation*. USA: Benjamin Cummings.

van de Pannw, M. (1996). Parameterized gait synthesis. *IEEE CG&A,* March, 40-49.

About the Editor

Keng Siau is Associate Professor of Management Information Systems (MIS) at the University of Nebraska, Lincoln (UNL) (USA). He is currently serving as the Editor-in-Chief of the *Journal of Database Management* and as the Book Series Editor for *Advanced Topics in Database Research*. He received his Ph.D. from the University of British Columbia (UBC), where he majored in Management Information Systems and minored in Cognitive Psychology. His master and bachelor degrees are in Computer and Information Sciences from the National University of Singapore. Dr. Siau has more than 150 academic publications. He has published more than 55 refereed journal articles, and these articles have appeared in journals such as *Management Information Systems Quarterly, Communications of the ACM, IEEE Computer, Information Systems, ACM-SIGMIS's Data Base, IEEE Transactions on Systems, Man, and Cybernetics, IEEE Transactions on Information Technology in Biomedicine, IEICE Transactions on Information and Systems, Journal of Database Management, Journal of Information Technology, International Journal of Human-Computer Studies, International Journal of Human-Computer Interaction, Behaviour and Information Technology, Quarterly Journal of Electronic Commerce*, and others. In addition, he has published more than 65 refereed conference papers, edited/co-edited nine books, edited/co-edited nine proceedings, and written more than 15 book chapters. He served as the Organizing and Program Chairs of the International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD) (1996-2004). He also co-chaired a number of minitracks at AMCIS and HICSS. For more information, please visit his web site at URL: http://www.ait.unl.edu/siau/.

About the Authors

Kyoung-Il Bae is a Senior Consultant at the IBM Business Consulting Services, Korea. He received a Ph.D. in Management Information Systems at the Korea Advanced Institute of Science and Technology (KAIST), Seoul. His research includes synchronous collaboration in groupware/virtual enterprise, system integration in the heterogeneous computing environments, component-based business modeling & development, and customer relationship management. His e-mail address is kibae@kr.ibm.com.

Ajantha Dahanayake is an Associate Professor in the Department of Information and Communication Technology at the faculty of Technology, Policy and Management, Delft University of Technology, The Netherlands. She previously served as an Associate Professor in the Department of Information Systems and Algorithms at the Faculty of Information Technology and Systems. She received her B.Sc. and M.Sc. in Computer Science from the University of Leiden, and Ph.D. in Information Systems from Delft University of Technology. She has served in a number of Dutch research and academic institutions. Her research interests are distributed Web-enabled systems, CASE, methodology engineering, component-based development and m-business. She is Research Director of the research program Building Blocks for Telematic Applications Development and Evaluation (BETADE).

Bassel A. Daou is a Ph.D. student at the University of Ottawa, Canada. He received his B.S. degree in Computer Science from the American University of Beirut, Lebanon, and his M.S. degree in Computer Science from the Lebanese American University - Beirut, Lebanon. His research interests include software engineering and database management systems.

Oscar Dieste is Assistant Professor with the School of Computing at the Universidad Complutense de Madrid, Spain. Dr. Dieste has a B.S. and a Ph.D. in Computing. He has been guest editor of the *International Journal of Software Engineering and Knowledge Engineering,* and reviewer of journals such as *IEEE Computer* or *IEEE Software.*

Jan L.G. Dietz is Professor in Information Systems Design at Delft University of Technology (The Netherlands) since 1994, after having been professor in MIS at the University of Maastricht for six years. He holds an M.Sc. in Electrical Engineering and a Ph.D. in Computer Science, and has practices business automation for 10 years. His current research interests are in organization engineering, enterprise architectures and systems ontology.

Eladio Domínguez is a Full Professor of Computer Science at the University of Zaragoza, Spain, and academician of the Royal Academy of Sciences of Zaragoza. He was previously Associate Professor of Topology at Zaragoza University and the Polytechnical University of Madrid. He has taught Mathematics at the Universities of Sevilla and Valencia. Dr. Dominguez received his Ph.D. in 1974 from the University of Zaragoza. His main research interests are in digital topology, information systems and phenomenology.

Anthony Fong received his B.E.E. degree from Villanova University, Pennsylvania (1969). He received his M.Sc. in Computer Science from the State University of New York at Buffalo in 1973. He was awarded a Ph.D. degree from University of Sunderland in 2003. At present he is Associate Professor and the Director of the EDA Centre in the Department of Electronic Engineering at the City University of Hong Kong. Dr. Fong has been awarded six United States patents, all on computer architecture and design. He has published more than 40 papers on computer architecture and design, and database. At present he is working on a computer system project called HISC (High-level Instruction Set Computer) for object-oriented computing. There is a U.S. patent issued and several others pending on HISC.

Joseph Fong is an Associate Professor in the Computer Science Department at City University of Hong Kong. He was Chair of the Hong Kong Computer Society Database Special Interest Group, and is Founder Chair of Sybase Hong Kong User Group, Hong Kong Web Society, and an annual International Conference on Web-based Learning. He has published many research journal papers in database, data warehousing and data mining, a monograph on Information Systems Reengineering by Springer Verlag in 1997, and two patents on Universal Database. Fong received his B.Sc. from State University of New York at Buffalo in Electrical Engineering (1976), M.Sc. from State University of New York at Stony Brook in Electronic Engineering (1977), M.B.A. from Golden Gate University in 1985, and Ph.D. from University of Sunderland in Computing (1993).

José Galindo has a Ph.D. in Computer Science by the University of Granada (Spain) and is a Professor of Computer Science on School of Engineering at University of Málaga (Spain). He is author of several books and papers on computer science, databases, information systems and fuzzy logic. His research interests include fuzzy logic, fuzzy databases and ethical issues in the technological age. He is a member of IDBIS research group and RITOS-2 ibero-american research net.

Roy Gelbard is a Lecturer at the Information Systems Department, Bar Ilan University, Israel. He held a faculty position at the Industrial Engineering and Management of Ben Gurion University, Beer Sheva, and Tel Aviv University. He received his Ph.D. and M.Sc. degrees in Information Systems from Tel Aviv University. He holds also degrees in Biology, Philosophy and Economics. His work involves systems analysis and design, modeling of motion, clustering and data mining.

Marcela Genero is Assistant Professor at the Department of Computer Science at the University of Castilla-La Mancha, Ciudad Real, Spain. Dr. Genero has an M.Sc. and a Ph.D. in Computing. She has published several papers in prestigious conferences and journals, and in 2002 she co-edited the book *Information and Database Quality.*

Terry Halpin, B.Sc., DipEd, B.A., MLitStud, Ph.D., is Distinguished Professor and Vice President (Conceptual Modeling) at Northface University, USA. His doctoral thesis formalized Object-Role Modeling (ORM/NIAM). After leaving academia to work on data modeling technology at Asymetrix Corporation, InfoModelers Inc., Visio Corporation, and Microsoft Corporation, he returned to academia, specializing in software development using a business rules approach to informatics. With a research focus on conceptual modeling and conceptual query technology, he has authored more than 100 technical publications and five books, including *Information Modeling and Relational Databases* and *Database Modeling with Microsoft Visio for Enterprise Architects*. He is a member of IFIP WG 8.1 (Information Systems) and is Editor or Reviewer for several academic journals.

Ramzi A. Haraty is an Associate Professor of Computer Science at the Lebanese American University - Beirut, Lebanon. He received his B.S. and M.S. degrees in Computer Science from Minnesota State University - Mankato, Minnesota, and his Ph.D. in Computer Science from North Dakota State University - Fargo, North Dakota. His research interests include database management systems, artificial intelligence, and multilevel secure systems engineering. He has more than 50 book, journal and conference paper publications. He is a member of the Association of Computing Machinery, Arab Computer Society, and International Society for Computers and their Applications.

Soonyoung Huh is an Associate Professor of Management Information Systems at the Korea Advanced Institute of Science and Technology (KAIST). He received a Ph.D. from the University of California, Los Angeles. Dr. Huh has published articles in such journals as *Decision Sciences, International Journal of Intelligent Systems in Accounting, Finance and Management, Omega,* and *Journal of Systems and Software.* His research deals with abstraction database application to business systems, model management, object-oriented database approach to decision support systems, and intelligent approaches in financial trading systems. His e-mail address is syhuh@kgsm.kaist.ac.kr.

Natalia Juristo is a Professor of Software Engineering with the School of Computing at the Universidad Politecnica de Madrid, Spain. Since 1992 she has been the Director of the M.Sc. in Software Engineering. Dr. Juristo has a B.S. and a Ph.D. in Computing. She was fellow of the European Centre for Nuclear Research (CERN) in Switzerland in 1988, and staff of the European Space Agency (ESA) in Italy in 1989 and 1990. During 1992 she was resident affiliate of the Software Engineering Institute at Carnegie Mellon

University. She was Program Chair for SEKE97 and General Chair for SEKE01 and SNPD02. Professor Juristo has been key speaker for CSEET03. She has been Guest Editor of special issues in several journals, including the *Journal of Software and Systems, Data and Knowledge Engineering* and the *International Journal of Software Engineering and Knowledge Engineering*. Dr. Juristo has been member of several editorial boards, including *IEEE Software* and the *Journal of Empirial Software Engineering*. She is senior member of IEEE.

Akos Ledeczi s a Senior Research Scientist at the Institute for Software Integrated Systems, Vanderbilt University, USA. His current research interests include model-based synthesis and simulation of embedded systems, network embedded systems and sensor networks. He received an M.Sc. from the Technical University of Budapest in 1989. He received a Ph.D. in Electrical Engineering from Vanderbilt University in 1995. He can be reached at akos.ledeczi@vanderbilt.edu.

Nashat Mansour is an Associate Professor of Computer Science at the Lebanese American University - Beirut, Lebanon. He received his B.E. and M.S. degrees in Electrical Engineering from the University of New South Wales, Australia, and M.S. in Computer Engineering and Ph.D. in Computer Science from Syracuse University, New York. His research interests include software testing and maintenance, and natural computation algorithms. He is a member of IEEE Computer Society and Arab Computer Society.

Miklos Maroti is a Research Assistant Professor at the Institute of Software Integrated Systems, Vanderbilt University, USA. His current research interest includes formal specification and analysis of embedded systems, active libraries of middleware components, and the constraint satisfaction problem. He received a Ph.D. in Mathematics from Vanderbilt University. He can be contacted at miklos.maroti@vanderbilt.edu.

Wai Yin Mok is an Assistant Professor of Management Information Systems at the University of Alabama in Huntsville, USA. His papers appear in *ACM Transactions on Database Systems (TODS)*, *IEEE Transactions on Knowledge & Data Engineering (TKDE)*, *Data & Knowledge Engineering (DKE)*, *Information Processing Letters* and *Journal of Database Management (JDM)*. He referees papers for *TODS*, *TKDE*, *DKE* and other journals. Currently he is on the Editorial Review Boards of *JDM* and *Informing Science Journal (ISJ)*. He is a program committee member of ER2003 International Conference on Conceptual Modeling and a track chair of 5th Annual Global Information Technology Management (GITM) World Conference in San Diego. He received a B.S., an M.S., and a Ph.D. in Computer Science from Brigham Young University in 1990, 1992, and 1996 respectively.

Ana M. Moreno is Associate Professor with the School of Computing at the Universidad Politecnica de Madrid, Spain. Dr. Moreno has a B.S. and a Ph.D. in Computing. Since 2001 she has been Director of the M.Sc. in Software Engineering. Dr. Moreno has been Visiting Scholar at the Vrije Unviersiteit (Amsterdam, The Netherlands) and Visiting Professor at the Unviersity of Colorado at Colorado Springs (USA). She was Program Chair for NLDB'01 and SNPD'02 and General Chair for CSEET03. She has been Guest Editor of special issues in several journals including *Data & Knowledge Engineering* and

International Journal of Software Engineering and Knowledge Engineering, and reviewer of journals like *ACM Computing Reviews, IEEE Transactions on Software Engineering, IEEE Computer* and *IEEE Software.* In 2001, she published a book titled *Basics on Software Engineering Experimentation.*

David Paper is an Associate Professor at Utah State University in the Business Information Systems Department, USA. He has several refereed publications appearing in journals such as *Information & Management*, *Journal of Information Technology Cases and Applications, Information Resource Management Journal, Communications of the AIS*, *Long Range Planning, Creativity and Innovation, Accounting Management and Information Technologies*, *Journal of Managerial Issues, Business Process Management Journal, Journal of Computer Information Systems*, and *Information Strategy: The Executive's Journal*. He has worked for Texas Instruments, DLS, Inc., and the Phoenix Small Business Administration. He has performed IS consulting work for the Utah Department of Transportation (Salt Lake City, UT) and the Space Dynamics Laboratory (Logan, UT). His teaching and research interests include change management, process reengineering, database management, e-commerce, and enterprise integration.

George C. Philip is a Professor in Information Systems and director of the M.S. in Information Systems program in the College of Business Administration at the University of Wisconsin Oshkosh, USA. His areas of publications and teaching include software design and development, database design, and expert systems. He teaches seminars and provides consulting services in these areas.

Mario Piattini holds an M.Sc. and Ph.D. in Computer Science from the Politechnical University of Madrid. He is Certified Information System Auditor by ISACA (Information System Audit and Control Association). He is also Full Professor at the Department of Computer Science at the University of Castilla-La Mancha in Ciudad Real (Spain). He has authored several books and papers on databases, software engineering and information systems. He leads the ALARCOS Research Group of the Department of Computer Science at the University of Castilla-La Mancha in Ciudad Real. His research interests are advanced database design, database quality, software metrics, object-oriented metrics, and software maintenance.

V. Ramesh, Ph.D., is an Associate Professor, Ford Teaching Fellow, and the Director of the M.S. in Information Systems at the Kelley School of Business, Indiana University, USA. His research focuses on database design, collaborative technologies, software design and technology implementation. He has also conducted studies on usability of mobile devices and has been a consultant to software engineering firms. He has published more than 40 research articles in leading journals and conferences such as *Communications of the ACM*, *ACM Transactions on Information Systems*, the *Journal of Management Information Systems*, and *Information and Management*, among others. He completed his Ph.D. in Information Systems from the University of Arizona, his M.S. in Computer Science from the University of Iowa and his B.E. in Computer Science from Birla Institute of Technology (Mesra), India.

Hajo A. Reijers is an Assistant Professor in Business Process Management at the Faculty of Technology Management of Eindhoven University of Technology, The Netherlands. He holds a Ph.D. in Computing Science from the same university. In the past eight years he has worked for several management consultancy firms, most recently as a manager within the Deloitte & Touche consultancy practice. He has published in various scientific journals, such as the *Journal of Management Information Systems* and the *International Journal of Cooperative Information Systems.* His main research interests are business process management, workflow management systems, Petri nets, and simulation.

Angel Luis Rubio is a Lecturer in Computer Science at the University of La Rioja, Spain. He received his B.Sc. in Mathematics from the University of Zaragoza, Spain (1994), and his Ph.D. in Computer Science from the University of La Rioja (2002). His current research interests are focused on metamodeling and method engineering.

Henk Sol is Dean of the Faculty of Technology, Policy and Management at Delft University of Technology, The Netherlands, and the Chair of Systems Engineering. He previously served as Chairman of the Department of Information Systems of the Faculty of Information Technology and Systems. He is Founder of the Faculty of Technology, Policy and Management as well as Founder of the Delft Institute for Service Engineering (DITSE). He is a pioneer in simulation and decision support research.

Israel Spiegler is a Professor and past chair of the Information Systems department at Tel Aviv University, Israel. He holds an M.Sc. and Ph.D. in Computers and Information from UCLA. His work included data sharing on computer network, a project of the ARPANET, a forerunner of the Internet. He held faculty positions at Boston University, School of Information Science at Claremont Graduate University, and UCLA. His main areas of interest are data modeling, databases, artificial intelligence and knowledge management in which he has published extensively.

Zoran Stojanovic is currently a Researcher for the Faculty of Technology, Policy and Management, at Delft University of Technology, The Netherlands. His research interests are in the areas of component-based development, Web services, enterprise and system modeling, Geographic Information Systems (GIS) and location-based services. He received his graduate engineering degree and Master of Philosophy in Computer Science and GIS from the Faculty of Electronic Engineering, University of Nis (Yugoslavia)(1993 and 1998 respectively). Since 1993, he has been working as a Researcher and Teaching Assistant in the fields of Computer Science, Software and System Engineering, first with the University of Nis (Yugoslavia) and after February 2000 with the Delft University of Technology (The Netherlands). During this period he has been an author of a number of publications.

Heikki Topi, Ph.D., is an Associate Professor of Computer Information Systems at Bentley College, USA. His research focuses on usability issues in the fields of data management and systems analysis and design, management and commercial utilization of advanced telecommunications technologies with a special emphasis on mobile solutions, and the effects of time availability constraints on human-computer interaction. He is Co-editor of *Auerbach's IS Management Handbook* and has published in a number of journals and conferences in Information Systems and Information Science such as the *Journal of Database Management*, *Journal of the American Society for Information Science and Technology*, *Information Processing & Management*, *Small Group Research*, and the *Communications of AIS*. He has a Ph.D. from Indiana University and an M.Sc. from the Helsinki School of Economics (Finland), both in Management Information Systems.

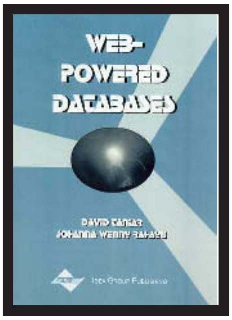
Angélica Urrutia, Ph.D., is Associate Professor at the Maule Catholic University, Chile, in the Computer Science Department. She is a member of the Chilean Computer Science Society and RITOS-2 (Red iberoamericana de tecnologías del software para la década del 2000), working group of CYTED. She is Founder and President of the Chilean Workshop on Data Base. In 2003, she obtained with Sobresaliente Cum Laude por unanimidad her Ph.D. in Computer Science at the Castilla-La Mancha University, Spain. The Master and Engineer Degrees in Computer Science were obtained respectively from the Concepcion University and Santiago University, Chile. Her major research topics are fuzzy database and information systems. In these arenas she has authored several original scientific papers.

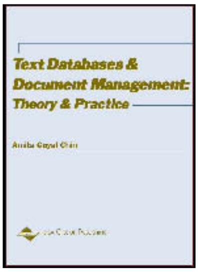
Peter Volgyesi is a Research Assistant at the Budapest University of Technology and Economics. He participated in the development of the Generic Modeling Environment (GME) at ISIS, Vanderbilt University. Recently he worked on domain specific modeling environments for networked embedded systems as part of the DARPA NEST project. His current research interests center on design and modeling of embedded real-time systems. He received an M.Sc. in Technical Informatics from the Budapest University of Technology and Economics. He can be contacted at volgyesi@mit.bme.hu.

Hing Kwok Wong is a full-time Ph.D. student in the Department of Computer Science at City University of Hong Kong. He received a Bachelor of Science in Information Technology, with First Class Honors, from the Electronic Engineering Department, and a Master of Philosophy in Computer Science, from the Computer Science Department at City University of Hong Kong (1999 and 2001, respectively). He has published several research journal and conference papers in data mining and XML databases. His current research interests are XML database system and XML-enabled database.

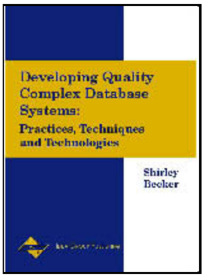
María Antonia Zapata is an Associate Professor in Computer Science at the University of Zaragoza, Spain. She received her B.Sc. in Mathematics (1988) and her Ph.D. in Computer Science (1994) from the University of Zaragoza. Her main research interests are in metamodeling, behavior modelling and method engineering.

Index


WEB-
POWERED
DATABASES
DAVID TANIAR
JOHANNA WENNY RAHAYU
Idea Group Publishing



Text Databases &
Document Management:
Theory & Practice
Amita Goyal Chin



Developing Quality
Complex Database
Systems:
Practices, Techniques
and Technologies
Shirley
Becker