Advanced Topics in Database Research

Keng Siau

IDEA GROUP PUBLISHING
Advanced Topics in Database Research
Vol. 1

Keng Siau
University of Nebraska–Lincoln, USA

This book is a release of the
Advanced Topics in Database Research Series
New from Idea Group Publishing

- Data Mining: A Heuristic Approach
  Hussein Aly Abbass, Ruhul Amin Sarker and Charles S. Newton/1-930708-25-4
- Managing Information Technology in Small Business: Challenges and Solutions
  Stephen Burgess/1-930708-35-1
- Managing Web Usage in the Workplace: A Social, Ethical and Legal Perspective
  Murugan Anandarajan and Claire Simmers/1-930708-18-1
- Challenges of Information Technology Education in the 21st Century
  Eli Cohen/1-930708-34-3
- Social Responsibility in the Information Age: Issues and Controversies
  Gurpreet Dhillon/1-930708-11-4
- Database Integrity: Challenges and Solutions
  Jorge H. Doorn and Laura Rivero/1-930708-38-6
- Managing Virtual Web Organizations in the 21st Century: Issues and Challenges
  Ulrich Franke/1-930708-24-6
- Managing Business with Electronic Commerce: Issues and Trends
  Aryya Gangopadhyay/1-930708-12-2
- Electronic Government: Design, Applications and Management
  Åke Grönlund/1-930708-19-X
- Knowledge Media in Healthcare: Opportunities and Challenges
  Rolf Grutter/1-930708-13-0
- Internet Management Issues: A Global Perspective
  John D. Haynes/1-930708-21-1
- Enterprise Resource Planning: Global Opportunities and Challenges
  Liaquat Hossain, Jon David Patrick and M.A. Rashid/1-930708-36-X
- The Design and Management of Effective Distance Learning Programs
  Richard Discenza, Caroline Howard, and Karen Schenk/1-930708-20-3
- Multirate Systems: Design and Applications
  Gordana Jovanovic-Dolecek/1-930708-29-7
- Managing IT/Community Partnerships in the 21st Century
  Jonathan Lazar/1-930708-33-5
- Multimedia Networking: Technology, Management and Applications
  Syed Mahbubur Rahman/1-930708-14-9
- Cases on Worldwide E-Commerce: Theory in Action
  Mahesh Raisinghani/1-930708-27-0
- Designing Instruction for Technology-Enhanced Learning
  Patricia L. Rogers/1-930708-28-9
- Heuristic and Optimization for Knowledge Discovery
  Ruhul Amin Sarker, Hussein Aly Abbass and Charles Newton/1-930708-26-2
- Distributed Multimedia Databases: Techniques and Applications
  Timothy K. Shih/1-930708-29-7
- Neural Networks in Business: Techniques and Applications
  Kate Smith and Jatinder Gupta/1-930708-31-9
- Information Technology and Collective Obligations: Topics and Debate
  Robert Skovira/1-930708-37-8
- Managing the Human Side of Information Technology: Challenges and Solutions
  Edward Szewczak and Coral Snodgrass/1-930708-32-7
- Cases on Global IT Applications and Management: Successes and Pitfalls
  Felix B. Tan/1-930708-16-5
- Enterprise Networking: Multilayer Switching and Applications
  Vasilis Theoharakis and Dimitrios Serpanos/1-930708-17-3
- Measuring the Value of Information Technology
  Han T.M. van der Zee/1-930708-08-4
- Business to Business Electronic Commerce: Challenges and Solutions
  Merrill Warkentin/1-930708-09-2

Excellent additions to your library!

Receive the Idea Group Publishing catalog with descriptions of these books by calling, toll free 1/800-345-4332
or visit the IGP Online Bookstore at: http://www.idea-group.com!
Advanced Topics in Database Research

Table of Contents

Preface .......................................................................................................................... viii

Keng Siau, University of Nebraska–Lincoln, USA

Section I: Enhancement of Current Database Models

Chapter I
Enforcing Cardinality Constraints in the ER Model
With Integrity Methods ......................................................................................... 1

Mira Balaban and Peretz Shoval
Ben-Gurion University, Israel

Chapter II
Ternary Relationships: Semantic Requirements and
Logically Correct Alternatives ........................................................................... 17

Trevor H. Jones, Duquesne University, USA
Il-Yeol Song, Drexel University, USA

Chapter III
Object-Oriented Database Benchmarks ......................................................... 34

Jérôme Darmont, Université Lumière, France
Michel Schneider, Université Blaise Pascal, France

Chapter IV
FOOM–Functional and Object-Oriented Methodology
For Analysis and Design of Information Systems ........................................... 58

Peretz Shoval and Judith Kabeli
Ben-Gurion University, Israel

Chapter V
Object-Process Methodology Applied to Modeling
Credit Card Transactions ..................................................................................... 87

Dov Dori, Israel Institute of Technology, Israel
Chapter VI
The Psychology of Information Modeling ................................................. 106
Keng Siau, University of Nebraska–Lincoln, USA

Section II: Refinement of the Contemporary
Database Model and Technique

Chapter VII
A Case Study of the Use of the Viable System Model
in the Organization of Software Development ..................................... 120
Peter Kawalek, Manchester Business School, UK
David Wastell, University of Salford, UK

Chapter VIII
Modeling of Business Rules For Active Database
Application Specification ............................................................................. 135
Youssef Amghar, Madjid Meziane and André Flory
The National Institute of Applied Sciences, France

Chapter IX
Algorithm Development, Simulation Analysis
and Parametric Studies For Data Allocation
in Distributed Database Systems ............................................................. 157
Amita Goyal Chin, Virginia Commonwealth University, USA

Chapter X
Using Weakly Structured Documents at the
User-Interface Level to Fill in a Classical Database ............................. 190
Frederique Laforest and André Flory
The National Institute of Applied Sciences, France

Chapter XI
Cooperative Query Processing via Knowledge
Abstraction and Query Relaxation .......................................................... 211
Soon-Young Huh, Kae-Hyun Moon and Jin-Kyun Ahn
Korea Advanced Institute of Science and Technology, Korea
Section III: Integration with the Internet Technology

Chapter XII
CMU-WEB: A Conceptual Model With Metrics
For Testing and Designing Usability in Web Applications .............. 230
Akhilesh Bajaj and Ramayya Krishnan
Carnegie Mellon University, USA

Chapter XIII
Managing Organizational Hypermedia Documents:
A Meta-Information System .............................................................. 250
Woojung Suh, POSCO Research Institute, Korea
Heeseok Lee, Korea Advanced Institute of Science and Technology, Korea

Chapter XIV
Changing the Face of War Through Telemedicine and Mobile E-Commerce ............................................................ 267
James A. Rodger, Indiana University of Pennsylvania, USA
Parag C. Pendharkar, Penn State Harrisburg, USA
Mehdi Khosrowpour, Penn State Harrisburg, USA

Section IV: Applications of Unified Modeling Language

Chapter XV
How Complex Is the Unified Modeling Language? ...................... 294
Keng Siau, University of Nebraska–Lincoln, USA
Qing Cao, University of Missouri–Kansas City, USA

Chapter XVI
Information Analysis in UML and ORM: A Comparison ............... 307
Terry Halpin, Microsoft Corporation, USA

Chapter XVII
Formal Approaches to Systems Analysis Using UML:
An Overview ...................................................................................... 324
Jonathan Whittle, NASA Ames Research Center, USA
Chapter XVIII
Extending UML For Space- and Time-Dependent Applications .............................................. 342
   Rosanne Price, RMIT University, Australia
   Nectaria Tryfona, Aalborg University, Denmark
   Christian S. Jensen, Aalborg University, Denmark

Chapter XIX
The Role of Use Cases in the UML: A Review and Research Agenda .................................. 367
   Brian Dobing, University of Lethbridge, Canada
   Jeffrey Parsons, Memorial University of Newfoundland, Canada

About the Authors .......................................................................................................................... 383

Index ............................................................................................................................................ 390
Preface

A database is an indispensable component of any information system. As database systems become increasingly widespread and important to businesses, database management thus becomes crucial to business success, particularly with the current emphasis on data, information, and knowledge as essential resources of any organization. Different database management products continue to emerge, as well as various extensions and complementary offerings to existing products. While commercial vendors have sought to accommodate new user requirements in the past few years, extensive academic research has also helped to enhance the functionality of current database systems, improve information storage, refine existing database models, meet the needs of Internet and e-commerce applications, and develop advanced applications such as data warehousing and data mining. This book presents the latest research ideas and topics in the field of database management.

This book is designed to provide readers with the latest research articles in database and database management. Emphasis is placed on improving theoretical understanding of database systems. It is expected that researchers in universities and research institutions will find such discussions particularly insightful and helpful to their current and future research. In addition, this book is also designed to serve technical professionals, since it is related to practical issues in database applications and aimed to enhance professional understanding of the capabilities and features of new technologies and methodologies.

This book is organized into four major sections: Enhancement of Current Database Models, Refinement of the Contemporary Database Model and Technique, Integration with the Internet Technology, and Applications of Unified Modeling Language.

Enhancement of Current Database Models

This section includes six chapters related to the ER models and the object-oriented approach, two of the most popular models in database literature. In “Enforcing Cardinality Constraints in the ER Model with Integrity Methods,” Balaban and Shoval suggest extending the Enhanced ER (EER) data model with integrity methods that can enforce cardinality constraints. In “Ternary Relationships: Semantic Requirements and Logically Correct Alternatives,” Jones and Song provide a basis for manipulating ternary relationships and offer key insights into methodologies and mechanisms for dealing with them. In “Object-Oriented Database Benchmarks,” Darmont and Schneider give an overview of the benchmarks aimed at evaluating the
performance of Object-Oriented Databases, with particular emphasis on Object-Clustering Benchmarks. Whereas Shoval and Kabeli propose a functional and object-oriented methodology (FOOM) in system analysis and design in “FOOM—Functional and Object-Oriented Methodology for Analysis and Design of Information Systems,” Dori suggests an object-process methodology (OPM), which addresses problems with the object-oriented methods, and presents an application of OPM that models the basic electronic commerce process of credit card transactions in “Object-Process Methodology Applied to Modeling Credit Card Transactions.” The last chapter in this section is “The Psychology of Information Modeling.” In this chapter, Siau reviews cognitive psychology, discusses its application to information modeling and method engineering, and proposes the use of cognitive psychology as a reference discipline for information modeling and method engineering.

**Refinement of Contemporary Database Model and Technique**

The five chapters included in this section propose applications of theoretical models and techniques to address issues in current database systems design and architecture. For example, in “A Case Study of the Use of the Viable System Model in the Organization of Software Development,” Kawalek and Wastell employ the Viable System Model to deal with business adaptation in a case of a software development enterprise. “Modeling of Business Rules for Active Database Application Specification” by Amghar, Meziane and Flory, develops new techniques to model business rules (e.g., active rules, integrity constraints, etc.). Chin presents the Partial REALLOCATE and Full REALLOCATE heuristics for efficient data reallocation in distributed database systems in “Algorithm Development, Simulation Analysis, and Parametric Studies for Data Allocation in Distributed Database Systems.” In “Using Weakly Structured Documents at the User-Interface Level to Fill in a Classical Database,” Laforest and Flory attempt to provide a system that associates document capture freedom with database storage structure. In “Cooperative Query Processing via Knowledge Abstraction and Query Relaxation,” Huh, Moon, and Ahn propose an abstraction hierarchy as a framework to facilitate approximate query answering.

**Integration With the Internet Technology**

The three chapters in this section focus on the Internet’s impact on database management systems. In “CMU-WEB: A Conceptual Model with Metrics for Testing and Designing Usability in Web Applications,” Bajaj and Krishnan propose a 3-dimensional classification space for WWW applications, attempting to provide a conceptual model that measures quantifiable metrics. In “Managing Organizational Hypermedia Documents: A Meta-information System,” Suh and Lee refine metadata roles, suggest a metadata classification and the corresponding metadata schema for organizational hypermedia documents (OHDs), and propose a Hyperdocument Meta-Information System. In “Changing the Face of War through Telemedicine and Mobile E-commerce,” Rodger, Pendharkar and Khosrowpour illustrate how Web-base capabilities can enhance medical management of battle spaces.
Applications of Unified Modeling Language

The five chapters in this section elaborate on application of Unified Modeling Language (UML) to information systems analysis and design. “How Complex Is the Unified Modeling Language?” by Siau and Cao provides a reliable and accurate quantitative measure of UML complexity. In “Information Analysis in UML and ORM: A Comparison,” Halpin examines the relative strengths and weaknesses of Unified Modeling Language (UML) and Object-Role Modeling (ORM) for conceptual data modeling, and indicates how models using one notation can be translated into the other. In “Formal Approaches to Systems Analysis Using UML: An Overview,” Whittle surveys recent attempts to provide a more precise description of UML, as well as techniques for formally analyzing UML models. In “Extending UML for Space- and Time-Dependent Applications,” Price, Tryfona and Jensen describe a UML extension, Extended Spatiotemporal UML (XSTUML) and introduce a technique for modeling composite data. In “The Role of Use Cases in the UML: A Review and Research Agenda,” Dobing and Parsons trace the development of use cases and outline a framework for future empirical research to resolve problems with use case applications and theoretical underpinnings.

Keng Siau
University of Nebraska–Lincoln, USA
October 2001
Section I

Enhancement of Current Database Models
Enforcing Cardinality Constraints in the ER Model With Integrity Methods

Mira Balaban and Peretz Shoval
Ben-Gurion University of the Negev, Israel

Entity-Relationship (ER) schemas include cardinality constraints that restrict the dependencies among entities within a relationship type. The cardinality constraints have direct impact on application transactions, since insertions or deletions of entities or relationships might affect related entities. Application transactions can be strengthened to preserve the consistency of a database with respect to the cardinality constraints in a schema. Yet, once an ER schema is translated into a logical database schema, the direct correlation between the cardinality constraints and application transaction is lost, since the components of the ER schema might be decomposed among those of the logical database schema.

We suggest extending the Enhanced-ER (EER) data model with integrity methods that can enforce the cardinality constraints. The integrity methods can be fully defined by the cardinality constraints, using a small number of primitive update methods, and are automatically created for a given EER diagram. A translation of an EER schema into a logical database schema can create integrity routines by translating the primitive update methods alone. These integrity routines may be implemented as database procedures, if a relational DBMS is utilized, or as class methods, if an object-oriented DBMS is utilized.

INTRODUCTION

The Entity-Relationship (ER) data model was introduced by Chen (1976) as a means for describing in a diagrammatic form, entities and relationships among entities in the subject domain. The ER model enjoys widespread popularity as a tool for conceptual database design, and received many extensions and variations, which are generally termed the Enhanced-ER (EER) model. An EER schema can be translated into logical database schemas, usually relational, and implemented with some specific DBMS, using its specific data.
definition language (DDL). Application programs that manipulate the database access the DBMS via its data manipulation language (DML), either directly or through a host programming language.

EER can be used not only to design a conceptual schema that will later on be translated into a logical schema, but also as a platform for database integration, i.e., to create a meta-schema for a multi database environment in which there are heterogeneous databases, utilizing different data models. Cooperation or federation of such databases is possible if a common meta-schema is created. EER can be the high-level model used for that purpose. Similarly, the EER model is used in database re-engineering; the data model of a legacy-system is first reverse-engineered to an EER schema, and later on translated and implemented in a new DBMS. Yet, the EER model deals only with the static (structural) aspects of the data model (namely, entities, relationships and attributes), but not with behavioural aspects (namely, procedures to manipulate the data that is defined by the schema, and to preserve the integrity of data). These aspects are taken care of at the implementation level, either by the DBMS (for example, when a relational DBMS performs referential integrity checks), or by the application programs.

An EER schema supports the specification of cardinality constraints, which restrict the dependencies among entities within a relationship type (see, for example, Lenzerini & Santucci, 1983; Lenzerini & Nobili, 1990; Ferg, 1991; Thalheim, 1992; Thalheim, 1998; Balaban & Shoval, 2001). For example, the cardinality constraints can specify that a department must have at least five workers and at most two hundred; or that an employee must work for one department only. The cardinality constraints have direct impact on maintenance transactions of the target system, since insertions or deletions of entities or relationships might affect related entities. This impact can be captured by operations that a transaction must trigger in order to preserve the cardinality constraints. Yet, once an EER schema is translated into a logical database schema, the direct correlation between the cardinality constraints and maintenance transactions is lost, since the components of the EER schema are usually decomposed among those of the target database schema. Moreover, at this level it is up to application programmers to correctly capture the constraints.

In this chapter, we suggest to enhance the EER model with the behavioural aspects of the cardinality constraints (see also Lazarevic & Misic, 1991; Balaban & Shoval, 2001). Specifically, we suggest extending the EER data model with integrity methods, i.e., methods that maintain the consistency of data according to the schema definition. Maintenance of consistency means that any attempt to add, delete or change entities and relationships in the subject domain of a schema is checked against the schema definition. Maintenance of integrity methods can be fully defined by the cardinality constraint. Hence, once the enhanced EER schema is mapped to a logical schema, the integrity methods can be mapped to respective integrity routines. These integrity routines may be implemented as database procedures, if a relational DBMS is utilized, or as class methods, if an object-oriented DBMS is utilized. The integrity methods are built on top of primitive update methods that perform the update transactions. This separation adds a layer of abstraction that enables the mapping of a behavior-enhanced EER schema into some target database schema, in terms of the primitive methods alone.

Another popular data model, which is closely related to the EER model, is the Object-Oriented (OO) model. The OO model supports static data abstraction using object classes, and models system behavior through methods (procedures) that are attached to object classes, with message passing as a means for communication among objects. Moreover, unlike EER, which is mainly used for conceptual data modeling, the OO approach is also
Enforcing Cardinality Constraints in the ER Model With Integrity Methods

utilized on the OO-DBMS level. Consequently, there is a direct transition from an OO data model to its implementation as an OO-DBMS. Therefore, there is a tendency to assume that the OO approach can/will replace EER as a data-modeling tool (see, for example, Kornatzky & Shoval, 1994; Dittrich, 1987; Elmasri & Navathe, 2000).

Nevertheless, the EER model might still be superior for the task of conceptual data modeling, since it enables finer distinctions between entity types to relationship types, and provides an explicit account for the associated constraints. Indeed, the direct transition between the levels of analysis, design and implementation is just one criterion. Other preference criteria among modeling tools involve various perspectives, e.g., learn-ability, comprehension (i.e., how easy it is for users to understand the schema), and quality of design (i.e., how accurate and complete is the schema that is being designed). Indeed, experimental comparisons show that EER is in many cases superior to OO with respect to the criteria of comprehension, quality of design, time needed for completion of a design task, and designers’ preference of models (Shoval & Frumermann, 1994; Bock & Rian, 1993; Shoval & Shiran, 1997). Shoval & Shiran (1997) concluded that even if the objective is to design and implement an OO schema, within an OO-DBMS or any other OO programming environment, the following strategy is recommended: first, design an EER schema; then map it to an equivalent OO schema; finally, augment the OO schema with the necessary behavioral constructs (e.g., methods and messages).

In the last few years, the Unified Modeling Language (UML) emerged and became a de facto industry standard of many aspects of object modeling (Fowler, 1997; Booch, Rumbaugh, & Jacobson, 1999). UML is a collection of visual languages (notations) that covers the static data, process and behavioral perspectives in modeling. The data modeling aspect in UML is handled by class diagrams and object diagrams, and also provides an Object Constraint Language. The static part of UML is richer than the conventional object schemas in object-oriented databases and in programming languages. The EER model is an integral part of UML—actually, UML includes all constructs of EER schemas, except for weak entity types. Indeed, we use the EER model as a representative for the static part of object modeling, in general. Our work can be viewed in the wider aspect of extending UML with structure sensitive methods. The extension of the EER data model with methods blurs the difference between OO schemas to EER schemas.

In the following section, the EER data model is briefly reviewed, and subsequent sections describe the suggested enhancement with structure methods.

THE ENHANCED-ENTITY-RELATIONSHIP (EER) DATA MODEL

EER is a data model for describing entities, their properties, and inter-relationships. A set of entities that share a common structure is captured as an entity type. Regular properties of entities are captured as their attributes. The attributes are associated with the entity types, and can be either simple or composite. In most entity types, entities are identified by an attribute (or attributes) called a key (or keys). In some entity types, entities are identified by their inter-relationships to other entities, possibly also by their partial-key attributes. Such entity types are termed weak.

Interactions among entities are modeled by relationships. A relationship type relates several entity types; it denotes a set of relationships among their entities. The number of entity types related by a relationship type is its arity (≥ 2). The role of an entity type within
a relationship type is specified by its role-name. Role-names are mandatory in case an entity
type plays several roles within a single relationship type. Two-ary relationship types are
called binary. A binary relationship type that relates an entity type to itself is called unary. Specialization and generalization inter-relationships among entity types are singled out as special kinds of relationships.

The cardinality constraints are set on relationship types, and characterize numerical
dependencies among entities within the relationship types. Existing EER models support a
variety of cardinality constraints, and sometimes use the same notation with different
semantics (Lenzerini & Santucci, 1983; Lenzerini & Nobili, 1990; Ferg, 1991; Thalheim,
1992; Thalheim, 1998). There are two main kinds of cardinality constraints: participation
and look-across. Both kinds of cardinality constraints can be imposed either on the entity
types related by the relationship type, or on the entities that already occur in the relationship
type. For example, let’s assume a ternary relationship sale among the entity types Salesper-
son, City and Product.

(a) Every Salesperson entity must be involved in at least three sales events—a participation constraint on the entity type Salesperson.
(b) Every Salesperson entity must sell at least two and at most five products in every city—a look-across constraint on the entity type Salesperson.
(c) If a Salesperson entity is already active, i.e., involved in some sales event, then it is involved in at least two more sales and in at most seven—a participation constraint on the relationship type sale.
(d) Every Salesperson entity that sells a product in some city can be constrained to sell that product in at least two more cities—a look-across constraint on the relationship type sale.

In this study we consider only look-across cardinality constraints. For binary relationship
types, the constraints are imposed on the related entity types (and thus, happen to coincide with participation constraints); for non-binary relationship types, the cardinality
constraints are imposed on the relationship types themselves.

Figure 1 presents an EER diagram for a medical clinic. This example will be used
throughout the paper. Rectangles describe entity types; diamonds describe relationship
types; circles describe attributes, where bolded circles signify multi-valued attributes. A key
attribute name is underlined, and a partial-key attribute name (of a weak entity type) has a
dotted underline. Solid lines among rectangles describe entity type hierarchies, and dotted
line rectangles and diamonds stand for weak entity types and their relationships to the
respective owner (strong) entity types.

We now turn to a more formal description of the EER model (see also Calvanese,
Lenzerini, & Nardi, 1998). An EER schema consists of entity type symbols, relationship type
symbols (each with associated arity), role-name symbols, and attribute symbols. Entity type
symbols can be strong, denoting non-weak entity types (e.g., Patient, in Figure 1), or weak
(e.g., Visit). Attribute symbols can be simple or composite. A composite attribute symbol is
associated with a set of attribute symbols (other than itself), e.g., address of Employee.
Furthermore, each attribute symbol is either single-valued or multi-valued. For example, a
physician may have many hobbies.

Every entity type symbol is associated with a set of attribute symbols, called its attributes. An attribute symbol can be singled out as a key. A strong entity type symbol has at least one mandatory key (e.g., ID of Employee). A relationship type symbol can also have attributes (e.g., date-of-study relationship), and may be associated with an optional key. Every relationship type symbol of arity n is associated with n entity type symbols, each with
Enforcing Cardinality Constraints in the ER Model With Integrity Methods

an associated role-name symbol, and minimum and maximum cardinality constraints. This complex association is captured by the syntactic relationship construct

\[ R(RN_1 : E_1[min_1, max_1], \ldots, RN_n : E_n[min_n, max_n]) \]

where \( E_i \) (\( 1 \leq i \leq n \)) are entity type symbols, \( RN_i \) (\( 1 \leq i \leq n \)) are role names, and \( min_i \) and \( max_i \) are natural numbers or the special symbol \( \infty \). For example, the above look-across constraint on the relationship type sale: “Every Salesperson entity that already sells a product in some city is constrained to sell that product in at least three cities and at most five” is captured by the relationship construct:

\[ \text{sale}(Salesperson : Salesperson[1, \infty], product : Product[1, \infty], city : City[3, 5]) \]

Another example is the ternary relationship study (see Figure 1). The meaning of the constraints on that relationship type is as follows:

- From the point of view of School: a certain physician obtains a certain specialization in at least one school, but he/she can obtain that specialization in no more than two schools;
- From the point of view of Physician: a certain school provides a certain specialization to at least one physician;
- From the point of view of Specialization: a certain physician acquires from a certain school at least two specializations, but not more than three.

The role names \( RN_i \) are used to identify the components of the relationship constructs. That is, \( RN_i(R) = E_i \), for \( 1 \leq i \leq n \). In practice, role names are optional; if they are not provided

*Figure 1: An EER diagram for a medical clinic information system*
within the schema, then they are schema-created. In Figure 1, supervises and supervised-by are the two role names of Physician in the supervision relationship; namely: a certain physician may supervise any number of other physicians (including none), and may be supervised by at most one other physician. A conventional simplified cardinality notation uses 1 for \( \min_i = \max_i = 1 \), and a letter (e.g., \( n \)) for \( \min_i \geq 0, \max_i = \infty \). So we get cardinality constraints such as \( 1:n:m \), \( 1:n, 1:1 \), etc. (Note: in ER diagrams the letter \( N \) is usually used to signify \( \infty \).)

An entity type symbol may be associated with a set of entity type symbols (other than itself) that form its specialization or sub-typing. The entity type symbol is called the super-type of the specialization, and the associated entity type symbols are called its subtypes. In Figure 1, Physician and Nurse are subtypes of Employee. The super-type and subtype relations are extended to include their transitive closure. The sets of attributes of an entity type and of any of its super-types are disjoint (no over-writing). The specialization has a kind, which is one of three values \( X, T, XT \). The kind \( X \) marks disjointness of the specialization subtypes, the kind \( T \) marks that the specialization covers the whole super-type, and the kind \( XT \) marks both. An entity type symbol may participate, as a subtype, in at most a single specialization.

A key of a type is a means for identifying the instances of the type via their attributes. A key of a strong entity type symbol and of a relationship type (if any) is an attribute of the type. A key of a weak entity type symbol is defined through related owner entity type symbols. Every weak entity type symbol \( E \) is associated with one or more binary relationship type symbols \( R_1, \ldots, R_s \), termed the identifying relationship types of \( E \), such that for each identifying relationship type \( R_i \) of \( E \), the EER schema includes a relationship construct:

\[
R_i(\ RN : E[n, m], RN' : owner(E)[1, 1]).
\]

Each entity type symbol \( owner(E) \) is termed an owner entity type of \( E \) for \( R_i \). The cardinality constraints for \( E \) in the relationship constructs for its identifying relationships mean that every entity of \( E \) is associated with a single owner entity, for every identifying relationship. The key of \( E \) consists of the keys of its owners, and possibly also its own partial key, which is any of its attributes. That is, \( \text{key}(E) = (\text{key}(owner(E)), \ldots, \text{key}(owner(E)), A) \), where \( A \) is an attribute of \( E \). In our example (Figure 1), Visit is identified by Patient ID plus Visit date. (Note that if a weak entity has two or more owners, it can be identified by their keys only, and must not have a partial key.)

A database instance \( D \) of an EER schema \( ER \) is defined by a non-empty finite domain of entities \( D \), a domain assignment \( dom \) for the attributes, and a meaning assignment for the symbols of the schema. \( dom \) is a partial mapping that associates a pair \((A, T)\) of a simple attribute symbol \( A \) and a type symbol \( T \) (entity or relationship) with a value domain \( dom(A, T) \). \( dom \) is extended to composite attributes by defining \( dom(A, T) = dom(A_1, T) \times \ldots \times dom(A_n, T) \) for a composite attribute symbol \( A \) that is associated with the attribute symbols \( A_1, \ldots, A_n \). The legal values of an attribute \( A \) of a type \( T \) are the values in \( dom(A, T) \). The application of the meaning assignment to a symbol \( s \) of \( ER \) is denoted \( s^D \). It is defined as follows:

For an entity type symbol \( E \), \( E^D \) is an entity type, i.e., a subset of \( D \). The elements of \( E^D \) are entities.

For a relationship type symbol \( R \) with arity \( n \), \( R^D \) is a relationship type, i.e., an \( n \)-ary relation over \( D \). The elements of \( R^D \) are relationships, and their components are labelled with role names; that is, instead of viewing relationships as ordered tupels, they are viewed as sets of labelled components. If the role names in the relationship construct of \( R \) are \( RN_1, \ldots, RN_n \),
then we refer to the relationships in $R^D$ as sets of the form $r = \{RN_i : e_i, ..., RN_n : e_n\}$. We define $RN_i(r) = e_i, (1 \leq i \leq n)$. The role name symbols $RN_i, ..., RN_n$ are referred to as the roles of $R^D$.

For an attribute symbol $A$ of a type symbol $T$(entity or relationship), $(A,T)^D$ is an attribute of $T^D$, i.e., a partial function from $T^D$ into either $dom(A,T)$ - if $A$ is single-valued, or into the power set of $dom(A,T)$ - if $A$ is multi-valued.

A database instance of a schema $ER$ is consistent if it satisfies the intended meaning of keys, relationship constructs, cardinality constraints, and sub-typing relationships. An EER schema is consistent if it has a consistent database instance. The constraints set by keys mean, for strong entity types and for relationship types, that the key attribute values uniquely identify the instances of the types. For weak entity types, their owner entities and their key attribute values uniquely identify them. A sub-typing of $E$ by $E_1, ..., E_n$ in $ER$, constrains $E^D_1$ to be a subset of $E^D(1 \leq i \leq n)$. If the kind of the specialization is $X$, then the sets $E^D$ are disjoint; if the kind is $T$, then the sets $E^D_i$ cover the set $E^D$, if the kind is $XT$, then both constraints hold. Note that these constraints imply inheritance of attributes and relationships through specialization relationships.

A relationship construct $R(RN_i : E_i[min_i, max_i], ..., RN_n : E_n[min_n, max_n])$ in $ER$ imposes the following constraints:

1. $R^D \subseteq E^D_1 \times ... \times E^D_n$.
2. The cardinality bounds on the i-th component delimit the number of elements in $E^D_i$ that can be related to given elements of the other entity types. The meaning for binary and non-binary relationship types is different: In binary relationship types the cardinality constraints in one entity type apply to every entity of the other entity type, while in non-binary relationship types the cardinality constraints in the i-th component apply only to every $R^D$-related combination of entities from the other entity types.

a. Binary relationship types:
For i = 1,2 and j = 2,1, respectively:
For all $e_i$ in $E^D_i, \min_i \leq \text{cardinality} \left( s_{RN_j = e_j \ (R^D)} \right) \leq \max_i$.
For example, in Figure 1 the constraint
supervision(supervises : Physician[0, 1], supervised-by : Physician[0, ∞])
means that in every consistent database instance $D$, for every Physician$^D$ entity $p$, selection of $p$ on the role name supervised-by in the relationship type supervision$^D$, yields at most a single relationship.

b. Non-binary relationship types:
For all $\{RN_i : e_i, ..., RN_n : e_n\}$ in $R^D$: For all $1 \leq i \leq n$,$\min_i \leq \text{cardinality} \left( s_{RN_j = e_j \ ... \ RN_l = e_l \ ... \ RN_{i-1} = e_{i-1} \ RN_i+1 = e_{i+1} \ ... \ RN_n = e_n} \ (R^D) \right) \leq \max_i$.
For example, the constraint:
sale(Salesperson : Salesperson[1, ∞], product : Product[1, ∞], city : City[3, 5])
means that in every consistent database instance $D$, for every sale$^D$ relationship with a Salesperson $s$ and a product $p$, selection of $s$ and $p$ on the role names Salesperson and product, respectively, in the relationship type sale$^D$, yields between three to five relationships.

EER EXTENDED WITH INTEGRITY METHODS

We suggest extending the EER schema with integrity methods, which are update methods that are sensitive to the cardinality constraints. The integrity methods should be defined on top of primitive update methods, which are integrity insensitive. The rational
behind this separation is that of the Abstract Data Types (ADT) approach: the primitive update methods serve as an abstraction barrier between the integrity methods and a logical database implementation of the EER schema. Every such implementation can be defined in terms of the primitive update methods alone. The integrity methods stay intact. The advantage of this approach is that the integrity methods are defined, and their properties are proved, once and for all, on the EER level. Since the definitions and proofs are rather complex, the advantage is clear.

**Primitive Methods**

The primitive methods perform *insertion, deletion, retrieval* and *attribute modification* in a database instance $D$ of an EER schema $ER$. They are associated with the entity and the relationship type symbols of the schema. Insertions always involve the creation of a new entity or relationship. Consequently, in a database instance created only with the primitive methods, entity types that are not related by the subtype/super-type relation have no entities in common. Similarly, all relationship types are mutually disjoint.

The primitive methods should be sensitive to the subtype and super-type relations in the sense that an insertion of an entity to the entity type $E'$ inserts it also to all super-entity types of $E'$. Similarly, deletion of an entity from $E'$ deletes it also from all sub-entity types of $E'$.

The addition of methods requires operations for retrieving the components of an EER schema, and of the information associated with instances in a database instance of a schema. These operations can be denoted as follows:

1. **Schema level operations**: For an entity type symbol $E$, the relationship type symbols that their constructs involve $E$ or a super-type symbol of $E$, and the corresponding role names, are given by $E.rels = \{[R, RN] | RN(R) = E', E' = E \text{ or is a super-type symbol of } E \}$. For a relationship type symbol $R$, the role names are given by $R.role_names$.

2. **Database level operations**:
   a. For an entity $e$ of $E^0$, $e.A$ retrieves the value on $e$ of attribute $(A, E^0)'$, where $E'$ is either $E$ or a super-type of $E$). $e.A$ is uniquely defined since the sets of attribute symbols associated with $E$ and its super-types are mutually disjoint. A legal key value for $A$ of $E$ in $D$ is a value in the domain that $D$ assigns to a key attribute symbol $A$ of $E$. For every $[R, RN]$ in $E.rels$, $e.relationships([R, RN])$ are the $R^0$ relationships whose $RN$ component is $e$, and $e.no_of_relationships([R, RN])$ is their number.
   b. For a relationship $r$ of $R^0$, $r.A$ retrieves the value of attribute $(A, R^0)$ on $r$. $r.RN$ retrieves the $RN$ entity component of $r$, for every role name $RN$ in $R.role_names$. A legal relationship for $R$ in $D$ is a labeled set $\{RN_1 : e_1, ... , RN_n : e_n\}$, such that $R.role_names = \{RN_1, ... , RN_n\}$, and $e_i$ is an entity in an entity type identified by the role name $RN_i$.

**Primitive Methods for an Entity Type Symbol $E$**

$v$ is a legal key value for $A$ of $E$ in $D$.

1. **E.insert $(A : v)$**—creates a new $E^0$ entity $e$, such that $e.A = v$, and for every $[R, RN]$ in $E.rels$, $e.no_of_relationships([R, RN]) = 0$, and $e.relationships([R, RN]) = \phi$. The entity $e$ is added to all super entity types of $E^0$. The return value is $e$.

2. **E.delete $(A : v)$**—deletes from $E^0$ and from all entity types $E'$ such that $E'$ in $E.subs$, the entity $e$ identified by the value $v$ of $(A, E^0)$ (i.e., $e.A = v$), if any.
3. `E.retrieve(A : v)`—Retrieves from \( E^0 \) the entity \( e \) identified by \( e.A = v \), if any. The return value is either \( e \), or NULL, if there is no such entity.

4. `E.retrieve_all()`—Retrieves all entities in \( E^0 \).

**Primitive Methods for a Relationship Type Symbol R**

Let \( r = \{ RN_1 : e_1, \ldots, RN_n : e_n \} \) be a legal relationship for \( R \) in \( D \).

1. `R.insert(r)`—Creates an \( R^0 \) relationship \( r \) with \( r.RN_i = e_i (1 \leq i \leq n) \). The return value is \( r \). The entities \( e_i (1 \leq i \leq n) \) are updated as follows: \( e_i.no_of_relationships([R, RN_i]) \) is increased by one, and \( r \) is added to \( e_i.relationships([R, RN_i]) \).

2. `R.delete(r)`—Deletes from \( R^0 \) the specified relationship, if any. If there is a deletion, the entities \( e_i (1 \leq i \leq n) \) are updated to decrease \( e_i.no_of_relationships([R, RN_i]) \) by one, and remove \( r \) from \( e_i.relationships([R, RN_i]) \).

3. `R.retrieve(r)`—Retrieves \( r \) from \( R^0 \), if any. The return value is either \( r \), or NULL, if there is no such relationship.

4. `R.retrieve_all()`—Retrieves all relationships in \( R^0 \).

**Primitive Methods for Attribute Modifications**

These methods perform modification, removal, and retrieval of a value of an attribute of an instance of a type. Simultaneous modification of multiple attributes can be handled by introducing compound update methods. Let \( T \) be a type symbol (entity or relationship). If \( T \) is an entity type symbol, let \( v \) be a legal key value for \( T \) in \( D \). If \( T \) is a relationship type symbol, let \( v \) be a legal relationship for \( T \) in \( D \). Let \( A \) be an attribute symbol associated with \( T \), and \( val \) a legal attribute value for \( (A, T) \) in \( D \), i.e., \( val \) belongs to \( \text{dom}(A, T) \).

1. `T.modify(v, A, val)`—If \( A \) is a single-valued attribute symbol, \( val \) is substituted for any previous value of the attribute \( (A, T)^0 \) on the instance (entity or relationship) identified by \( v \). If \( A \) is a multi-valued attribute symbol, \( val \) is added to any previous value of the attribute \( (A, T)^0 \) on the instance identified by \( v \).

2. `T.remove(v, A, val)`—If \( A \) is a single-valued attribute symbol, and the value of the attribute \( (A, T)^0 \) on the instance identified by \( v \), is \( val \), it is replaced by NULL. If \( A \) is a multi-valued attribute symbol, \( val \) is removed from the value of the attribute \( (A, T)^0 \) of the instance identified by \( v \).

3. `T.get(v, A)`—Retrieves the value of attribute \( (A, T)^0 \) of the instance identified by \( v \).

**Integrity-Preserving Policies**

In order to preserve the consistency of the database, an integrity update method might invoke associated update methods or be refused. We distinguish four integrity-preserving policies:

1. **Reject**—the update operation is refused. This is in some sense a brute force action for integrity preservation. It should only be used with caution, in order to not block database updates.

2. **Propagate** (also termed **Cascade**)—an insertion or deletion of an instance violates a cardinality constraint, and invokes appropriate deletion or insertion actions. Propagation is achieved by dispatching the impact of a newly inserted or deleted entity or relationship to its neighbouring relationships. Among the four integrity-preserving actions, **Propagate** is the most faithful to the policy of integrity preservation. But it is also the most expensive, and one should be careful not to embark on an unlimited sequence of update operations. Since the schema is consistent, it has consistent (finite)
database instances. In general, it is worthwhile that the user can guarantee full propagation before actual updates are applied.

3. **Nullify**—violation of a cardinality constraint is relaxed by the insertion of a new *null entity*, and including it in a relationship. **Nullify** is a compromise between the desire to preserve integrity, and the inability or unwillingness to propagate. In a **Nullify** operation, a “fictional” entity is inserted to an entity type and connected to “real” entities, so that their integrity is preserved. The assumption is that cardinality constraints do not apply to null entities. A null entity can be replaced by a real one, by reconnecting its related entities to a real entity.

4. **Schema revision**—integrity violation is removed by revising, or re-defining, the cardinality constraints. The revision can only decrease a minimum cardinality constraint, or increase a maximum cardinality constraint. **Schema revision** is intended to resolve impossible cardinality constraints, or emerges from new definition of the domain. It seems that one should be careful not to abuse this intention by using this action as a replacement for simple **Propagate** or for **Nullify** so to temporarily preserve all constraints.

These integrity-preserving policies represent conventional approaches for integrity maintenance (Etzion & Dahav, 1998). We suggest that these policies should be determined in an interactive mode, and not be fixed in advance for the different types of the schema (as suggested in Lazarevic & Misic, 1991).

**Integrity Methods of Entity Types**

The **integrity_insert** and **integrity_delete** operations might invoke the **Propagate** policy for integrity preservation. Propagation for insertion is caused by non-zero minimum constraints in binary relationship constructs, since they imply that a newly added entity must be related to another entity. Minimum constraints in non-binary relationship constructs do not pose any restriction on a new entity, since they apply only to already existing relationships. Propagation for deletion is caused by relationships in which the deleted entity participates. Maximum constraints are not violated by insertions or deletions of entities, but should be considered when an already existing entity is connected to a new relationship (see next subsection).

**The Integrity_Insert Method**

The **integrity_insert** method for an entity type symbol $E$, and a legal key value $v$ for $A$ of $E$ in a database instance $D$, involves the following operations: If $A : v$ does not identify an already existing entity in $E^D$, a new entity with the key value $v$ for $A$ is inserted into $E^D$, and the insertion effect is propagated. The propagation involves observation of all binary relationship type symbols with which $E$ or any ancestor of $E$ is associated. If the minimum cardinality constraints that they set on an entity in $E^D$ are not met by the new entity, then the involved relationship types are asked to provide the missing relationships with the new entity. This way the insertion is propagated from $E$ to its related relationship type symbols, and from there it can further propagate to new entity type symbols.

**Example:** Consider the EER schema from Figure 1. In order to insert a Physician entity with a license number ph12345 to a database instance, **Physician.integrity_insert** (license-no: ph12345) is applied. Integrity preservation requires that several constraints be checked before the real insertion takes place. First, we need to check that this physician is not already in the database. Second, we need to observe all binary relationship type symbols...
whose constructs involve \textit{Physician} or any ancestor of \textit{Physician}, to see whether their cardinality constraints are not violated. The binary relationship type symbols \textit{supervision, physician-schedules} and \textit{manages} do not constrain the new entity since they have zero minimum constraints (namely, a new physician must not supervise any other physician, nor must he/she have any schedule or manage any department).

The binary relationship type symbols \textit{treating} and \textit{works-for} provide minimum requirements on the number of relationships involving each physician instance, i.e., every physician must have at least one visit, and must work for some department. So, we have to ask the user for a candidate visit to be related to the new physician through the treating relationship type, and for a candidate department entity as well. The user might provide an existing entity or suggest a new one. Indeed, the required department entity may be an existing one, but the required visit must be a new entity, since for any visit there is exactly one physician. So, every visit entity existing in the database already has its physician related to it through \textit{treating}. Once the user provides a new \textit{Visit} entity, the process of integrity preservation has to repeat itself. The new visit might not be related to a \textit{Medicine} entity through \textit{prescriptions}, but it must have a \textit{Patient} entity through \textit{patient-visits}. (Note that \textit{Visit} is a weak entity type of \textit{Patient}.) Again we need to ask the user for a candidate patient, who in this case can be an existing or a new patient. In any case, since there are no minimal restrictions on \textit{patient-visits} relationships, the propagation stops. In order to avoid or terminate propagation, the user might decide to nullify missing entities in relationships. For example, if the department of the new physician does not exist in the database yet (or cannot be determined), the user can insert a new null entity to the \textit{Department} entity type, and connect it to the new physician in a \textit{works-for} relationship. Later on, when the missing department is inserted (or determined), the user can reconnect the current physician entity to it. Similarly, if the new physician has no visit yet, the user can insert a new null entity to the \textit{Visit} entity type and connect it to the physician in the \textit{treating} relationship. Later on, when the new physician treats his/her first patient, the user will reconnect that treating relationship to the proper patient’s visit.

\textit{The Integrity_Delete Method}

The \texttt{integrity_delete} method for an entity type symbol \textit{E}, and a legal key value \textit{v} for \textit{A} of \textit{E} in a database instance \textit{D}, involves the following operations: If \textit{A : v} indeed identifies an existing entity in \textit{ED}, then the entity is deleted, and the deletion effect is propagated. The propagation involves observation of all relationships that include the removed entity. These relationships can be instances of relationship types \textit{R} of \textit{E.rels}. For each such relationship, the user can choose one of four possibilities: Reject the update, Nullify the references to the removed entity, Reconnect the relationship, or Delete the relationship. In the Reconnect option, the user is asked to replace the deleted entity with another one, new or old. Insertion of a new entity might propagate as above. In the Delete option, the relationship is deleted, and the effect may propagate, either to further deletion of the related entity (for a binary relationship) or to further deletions of other relationships in the same relationship type (for a non-binary relationship). Note that it may not be possible to avoid propagation by deleting the relationships before the entities, since the deletion of a binary relationship is sanctioned if it violates the minimum constraints set on the related entities.

\textbf{Example} (see Figure 1): In order to delete the \textit{Physician} entity with license number ph12345 from the database, \texttt{Physician.integrity_delete (license-no : ph12345)} is applied. Integrity preservation requires that several constraints be checked before the real deletion takes place. First, we need to check that this physician is indeed in the database. Second, we
need to decide what to do with each relationship in which this entity participates. As explained above, we can choose among **Reject, Nullify, Reconnect** or **Delete**. For example, due to the cardinality constraints on the treating relationship type, the physician entity must have at least one treating relationship. If we decide to reconnect the relationship, it might be to an existing physician or to a new physician. For an existing physician, we need to check whether it does not exceed the maximum of 20 visits. For a new physician, it should be inserted, connected, and the effect of the insertion propagated.

If we decide to delete the **treating** relationship, the minimum cardinality constraints on the **Visit** entity type should be checked. In general, if a **Visit** entity could be related to several **Physician** entities, then it might have been possible to delete the relationship without violating the cardinality constraints. However, here, since every visit must be related to exactly one physician, deletion of the relationship violates the database integrity and must invoke propagation of the deletion to the related visit. In order to avoid or terminate the propagation, the user might decide to **Nullify** the **Physician** entity in the **treating** relationship of that visit. (Note that in our example all possible solutions are problematic because of the requirement that a physician must have at least one visit—an unrealistic constraint.)

**Integrity Methods of Relationship Types**

The operations are **Connect**, **Disconnect**, and **Reconnect**. The **Connect** operation inserts a new relationship between existing entities; the **Disconnect** operation deletes a relationship; and the **Reconnect** operation replaces an entity in a relationship.

Our recommended policy is to restrict the **Connect** and **Disconnect** operations so that they do not propagate outside the relationship type under consideration. Hence, if these operations violate the integrity of the related entities (for binary relationship types only) they are rejected and can be replaced by other operations, such as **reconnect**, or **integrity_insert** or **integrity_delete**. The **Reconnect** operation, on the other hand, can propagate to the related entity types, when an entity in a relationship is replaced by a new entity. This propagation cannot be avoided, since the database integrity might block the independent insertion of the new entity.

**The Connect Method**

The **Connect** method for a relationship type symbol $R$, and a legal relationship $r = \{ R_{1}^{e_{1}}: e_{1}, \ldots, R_{n}^{e_{n}}: e_{n} \}$ for $R$ in a database instance $D$, involves the following operations: If $r$ does not already exist in $R^{0}$, and if all entities $e_{i}$ ($1 \leq i \leq n$) already exist, the new relationship is tested not to violate the cardinality constraints set on $R$. If the test succeeds, the new relationship is inserted to $R^{0}$.

**Example** (see Figure 1): In order to connect a **Physician** entity $p$ to a **Visit** entity $v$, $\text{treating.connect( \{ treating.physician : p, treating.visit : v \} )}$ is applied, (**treating.physician** and **treating.visit** are the schema provided role names for the **treating** relationship type). If both entities do exist in the database instance and can be connected without violating the maximum cardinality constraints set on **treating**, the relationship is inserted to the **treating** relationship type. Note that the maximum constraint on the **visit** entity $v$ is 1. That means that for the connection to be performed, $v$ must have been inconsistent prior to the connection. This can be the case, for example, if the current **treating.connect** operation was invoked from within **Visit.integrity_insert(Patient.ID : p123, date : 3.9.01)**, assuming that $v$ is identified by the key value (**Patient.ID**: $p123$, **date**: 3.9.01). No further updates are invoked since **treating** is binary.
In order to connect a Physician entity \( p \) to a Specialization entity \( f \), and a School entity \( s \), `study.connect( \{ \text{study.physician} : p, \text{study.specialization} : f, \text{study.school} : s \} )` is applied. If the entities do exist in the database instance and can be connected without violating the maximum cardinality constraints set on \text{study}, the relationship is inserted to the study relationship type. Since \text{study} is non-binary, an auxiliary method \text{study.make_consistent}( \{ \text{study.physician} : p, \text{study.specialization} : f, \text{study.school} : s \} ) should be applied. Assume, for example, that there are no other \text{study} relationships with the physician \( p \) and the school \( s \). Then, one such \text{study} relationship is still missing (there should be at least 2, and at most 3). Then the user can be asked to choose among \text{Reject}, \text{Nullify}, \text{Connect} or \text{Schema revision}. If \text{Nullify} is chosen, then a new null entity is inserted to the Specialization entity type and connected to the entities \( p \) and \( s \) to yield a new \text{study} relationship. If \text{Connect} is chosen, an existing or a new Specialization entity is connected to \( p \) and \( s \) to yield a new \text{study} relationship. If the Specialization entity is new, it should also be made consistent (by invoking \text{Specialization.make_consistent} on it).

The Disconnect Method

The Disconnect method for a relationship type symbol \( R \), and a legal relationship \( r = \{ RN_1 : e_1, ..., RN_n : e_n \} \) for \( R \) in a database instance \( D \), involves the following operations: If \( r \) exists in \( R^0 \), and if the deletion of the relationship does not violate minimum cardinality constraints set on the participating entity types in \( R \) (possible only if \( R \) is binary), then the relationship is disconnected, and the effect is propagated to other relationships in \( R^0 \), if needed. The method should treat differently binary relationship types and non-binary ones. For a binary \( R \), preserving the integrity of its cardinality constraints might require deletion of the related entities (since the constraints are imposed on the related entities). We feel that, in such cases, it is more reasonable to start with the deletion of the related entities and not with their relationship. Hence, violation of the cardinality constraints for a binary \( R \) should lead to rejection. For a non-binary \( R \), violation of its cardinality constraints might require reconnection or disconnection of other relationships in \( R^0 \). Disconnect is applied to every selection of \( n-1 \) entities from \( r \), to see whether they still meet the minimum cardinality specified for the \( n \)-th entity type symbol.

Example (see Figure 1): In order to disconnect the treating relationship \{treating.physician : \( p \), treating.visit : \( v \} \), the method `treating.disconnect( \{treating.physician : \( p \), treating.visit : \( v \} \)` is applied. If the Visit entity is maximally consistent (i.e., \( v \) is related only to the Physician entity \( p \)), disconnect should be rejected, since it violates the minimum cardinality constraints set on Visit. In any case, since treating is binary, no further updates are invoked.

In order to disconnect a \text{study} relationship \{\text{study.physician} : \( p \), \text{study.specialization} : \( f \), \text{study.school} : \( s \} \), the method `study.disconnect( \{\text{study.physician} : \( p \), study.specialization : \( f \), study.school : \( s \} \)` is applied. First, the relationship should be deleted from the \text{study} relationship type. Since \text{study} is non-binary, an auxiliary method `study.make_deleted_relationship_consistent(\text{study.physician} : \( p \), \text{study.specialization} : \( f \), \text{study.school} : \( s \)` should be applied. Assume, for example, that prior to the disconnection, there were exactly two \text{study} relationships with the physician \( p \) and the school \( s \). Then, following the disconnection, one such relationship is missing. The user can be asked to choose among \text{Reject}, \text{Nullify}, \text{Connect}, \text{Disconnect}, or \text{Schema revision}. If \text{Nullify} is chosen, then a new null entity is inserted to the Specialization entity type, and connected to the entities \( p \) and \( s \) instead of the \text{study} relationship that was just deleted. If \text{Connect} is chosen, an existing or
a new **Specialization** entity is connected to $p$ and $s$ to yield a new **study** relationship. If the **Specialization** entity is new, it should also be made consistent (by invoking **Specialization.make_consistent** on it). If **Disconnect** is chosen, then the remaining **study** relationship with physician $p$ and school $s$ is disconnected. If **Schema revision** is chosen, then the minimum bound on the **study.specialization** role of the **study** relationship type is decreased (from 2 to 1).

**The Reconnect Method**

This operation stands for a **Disconnect** operation that is followed by a **Connect** operation. However, under the integrity preservation policy, **Reconnect** is essential since otherwise, there would be no way to reconnect a relationship that includes an entity with mandatory participation, that participates in no other relationship. For example, if Fred is an employee that moves from the Pediatric Department to the Internal Department, disconnecting the (Fred, Pediatric Department) relationship is rejected due to the mandatory participation of **Employee** in the **works-for** relationship type. Hence, the move cannot be achieved as a sequence of **Disconnect** and **Connect**.

The **Reconnect** method for a relationship type symbol $R$, accepts three parameters: a legal relationship for $R$ in a database instance $D$, a role name $RN$ for $R$, and a legal key value $A : v$ for the super-entity type symbol identified by $RN$ in $R$ given by $R.E_of_RN$, with respect to $D$. For simplicity, we denote the relationship $r = \{ RN : e, R.E_of_RN : e_1, ..., R.E_of_RN : e_k \}$, where $RN$ is the role name whose entity $e$ should be replaced by the new entity that is identified by $A : v$. The **Reconnect** method involves the following operations: If $r$ exists in $R^0$, and if the deletion of the relationship does not violate minimum cardinality constraints on the replaced entity $e$ (possible only if $R$ is binary), then the relationship is disconnected, the effect is propagated to other relationships in $R^0$, and if the insertion of the new relationship does not violate maximum cardinality constraints set on $R$, the relationship is reconnected. This last test refers to the super-entity type symbol identified by the role name $RN$ in $R$ (given by $R.E_of_RN$). First, it is checked whether $A : v$ identifies an existing entity in $(R.E_of_RN)^0$. In that case, if this entity can be connected to the entities $e_1, ..., e_k$ to form a new relationship $r'$ in $R^0$, $r'$ is inserted to $R^0$. If not, the user is asked to replace $A : v$ by another legal key value. If $A : v$ does not identify an entity in $(R.E_of_RN)^0$, a new entity with the key value $A : v$ is inserted to $(R.E_of_RN)^0$, the new relationship is inserted to $R^0$, and the effect of the entity insertion is propagated in $D$.

**Example** (see Figure 1): In order to reconnect the **treating** relationship \{treating.physician : $p$, treating.visit : $v$\} to a new physician, identified by the key value license_no : ph5678 of **Physician**, the method **treating.reconnect**( \{treating.physician : $p$, treating.visit : $v$\}, treating.physician, license_no : ph5678 ) is applied. If $v$ is the only visit of the physician entity $p$, then after the deletion of \{treating.physician : $p$, treating.visit : $v$\}, $p$ becomes inconsistent as it must have at least one visit. The user can be asked to compensate for the missing **treating** relationship of $p$ either by connecting it to a **Visit** entity (a new one—real or null, or an already existing one), or by deleting $p$. After that, the physician $p'$, identified by the license number ph5678 can be inserted to **Physician** (if it is not already there), the new **treating** relationship \{treating.physician : $p'$, treating.visit : $v$\} can be connected, and $p'$ should also be made consistent if it is new.

In order to reconnect the **study** relationship \{study.physician : $p$, study.specialization : $f$, study.school : $s$\} to a new physician identified by the key value license_no : ph5678 of **Physician**, the method **study.reconnect**( \{study.physician : $p$, study.specialization : $f$, study.school : $s$\}, study.physician, license_no : ph5678 ) is applied. Since **study** is
non-binary, the deletion of the relationship \{study.physician : p, study.specialization : f, study.school : s\} does not affect any entity inconsistency. Then the physician \(p'\), identified by the license number \(ph5678\) is inserted to Physician (if it is not already there), the new study relationship \{study.physician : \(p'\), study.specialization : f, study.school : s\} can be connected, and \(p'\) should also be made consistent if it is new. Since study is non-binary, the effect of the deleted study relationship should also propagate to the rest of the study relationships.

Consistency Preserving Property of the Integrity Methods

The integrity methods suggested in this study are valuable since they can preserve the integrity of a consistent database instance. That is, if \(D\) is a consistent data base instance of an EER schema \(ER\), and \(D'\) results from \(D\) by the application of an integrity method, then \(D'\) is a consistent up to null entities instance of \(ER\). That is, \(D'\) satisfies all key relationship construct and sub-typing constraints and all cardinality constraints set on real entities. If \(D'\) includes no null entities, then it is a consistent instance of \(ER\).

INTEGRITY METHODS OF ATTRIBUTES

The attribute operations are defined for an entity or a relationship type symbol \(T\), and an attribute \(A\) of \(T\). They accept the same parameters as their corresponding primitive methods. In principle, they can be extended to accept any number of attribute-value pairs. The attribute operations that we suggest are: \(T.Integrity\_modify( v, A, val )\), \(T.Integrity\_remove( v, A, val )\), and \(T.Integrity\_get( v, A, )\).

SUMMARY

In this study, we briefly described an extension of the EER data model with integrity-sensitive update methods. For that purpose, we first classified the cardinality constraints and formalized their semantics. The integrity methods can be fully defined by the cardinality constraints, using the primitive update methods. Hence, our approach enables the development of a tool that automatically creates the integrity methods for a given EER diagram.

We are currently extending the conventional EER-to-OO schema mappings to map the newly added methods. We have proved that the method mapping preserves their semantics. The method mapping is defined in terms of the primitive methods alone. The mapping of the more complicated integrity methods is directly obtained from the primitive methods mapping. Alternative mappings of the EER schema into different logical database schemas can be extended similarly by faithful translation of the primitive methods.

The contribution of this research is in capturing the intended meaning of the cardinality constraints as an integral part of the EER schema. Moreover, since the EER schema language is an integral part of the UML language, and since the latter does not account for active integrity maintenance of the associated constraints, the suggested enhancement actually extends the static part of UML with integrity-preserving methods. The data-model independence of our approach is, therefore, essential, since a UML conceptual schema should not be hard-wired to a specific logical database model.
ENDNOTE


REFERENCES


INTRODUCTION

Chapter II

Ternary Relationships: Semantic Requirements and Logically Correct Alternatives

Trevor H. Jones
Duquesne University, USA

Il-Yeol Song
Drexel University, USA

Conceptual data modeling is a backbone of most major systems development projects. Within this arena, one of the most widely used techniques is the entity-relationship (ER) or extended entity-relationship model (EER, henceforth also referred to as ER), introduced by Chen (1976). However, there are multiple competing models and notations, each having distinct strengths and weaknesses. Many of the model definitions and underpinnings continue to undergo practical and theoretical development. The abilities of each of the model structures and notations to fully represent the semantics of any given situation are constantly compared, with many issues open to argument. More specifically, certain arguments revolve around the inclusion of binary or N-ary representation of relationships in ER models. A central argument stems from the superior ability of N-ary modeling to reflect the true semantics of any given situation, whereas a binary model provides the simplest constructs for expressing information systems’ logical design and is equivalently represented in a relational database management system (DBMS) (McKee & Rodgers, 1992).

The purpose of conceptual models is twofold: to provide a semantically correct, conceptual representation of the organizational data, as well as to provide a platform from which to develop a logical implementation schema. Consequently, the superior methodology for model construction is to adopt the semantically superior form and provide some
heuristic set to allow transformation to a result, which can be implemented in the more desired format. This course of action has been widely recognized and well researched in the form of developing relational schema from ER diagrams; rule sets as well as automated tools have been developed that offer to guide the process of translation (e.g., Jajodia, Ng & Song, 1983; Ling, 1985; Markowitz & Shoshani, 1992; Elmasri & Navathe, 2000).

Within these methodologies, one area continues to be formally investigated. N-ary relationships in ER models continue to be constructs that are misunderstood by educators, difficult to apply for practitioners, and problematic in their interpretation to relational schemas. These problems are due to causes ranging from a difficulty in identifying legitimate ternary relationships in practical situations to the lack of understanding of the construct in relation to the basis of normalization upon which the relational model is grounded. Song, Evan, and Park (1995) provide a comparative analysis of conceptual modeling notations. While all of the notations had some allowance for ternary modeling, none of the CASE tools included in the study allowed for the use and translation of ternary relationships. This indicates the recognition of ternary relationships as having semantic significance, but a practical difficulty of implementing them beyond the equivalent logical level. Very little research has been completed on the theoretical underpinnings of N-ary relationships, and that which exists is generally created as passing references in search of other research solutions (Krippendorf and Song, 1997; Song, Rowan, Medsker & Ewens, 2001).

N-ary relationships are substantially different from binary, in that they create a particular set of dynamics inherent to the relationship, which do not exist otherwise. With binary relationships, the construct is concerned with the cardinality between two participating entities. However, with relationships of a higher order, we encounter the problem of subsets within the original relationship, which may affect the logical derivation of the database structure subsequently (Dullea and Song, 1998). This complexity accounts for differing notations to address the relationships (Song et al., 1995) and other problems associated with these constructs. McAllister (1996, pp. 257) states, “…different approaches for modeling n-ary relationships differ not only in terms of appearance and placement, but also in terms of meaning.”

This chapter serves to provide insight to these problems. It seeks to formally analyze the dynamics of having three entities participating in a relationship simultaneously. This is done from two perspectives:

1. The theoretical approach to understanding this type of conceptual construct and the subsequent analysis of logical and relational models founded on these theories.
2. The practical aspects of using these constructs in entity-relationship modeling and how the various construct combinations can be mapped to the logical/physical model.

The second aspect is partly founded on the first, because the potential decomposition of N-ary constructs and their final representations can be derived from theoretical analysis of the implicit relationships.

There will, therefore, be a particular effort to explain the simultaneous existence of N-ary and binary relationships that share the same participating entities and are semantically related; this viewpoint has never been raised in previous research and leaves several questions unanswered. It is possible that an N-ary relationship may contain, or have imposed on it, a binary relationship between two of its participating entities that is semantically related to, and therefore potentially constrains, the ternary. Jones and Song (1996) have previously analyzed the question of which semantically related N-ary and binary relationship combinations can logically co-exist simultaneously. In their work, they have shown that only certain combinations of ternary/binary
cardinalities may simultaneously co-exist and have provided a set of rules and notations that provide for conceptually correct modeling.

In extending the previous work of Jones and Song (1996), and providing an explanation of these implicit dynamics of N-ary structures, this work allows a further investigation of decomposition and restructuring of N-ary relationships to multiple binary structures, based on relational theory.

Many of the foregoing theoretical arguments lead to more utilitarian questions. The conceptual understanding of the N-ary construct remains difficult for practitioners. Various notations exist for representing the construct in modeling, each with its own strengths and weaknesses. One of the most difficult problems is identifying exactly when an N-ary relationship should be used or when its binary equivalent is available. No prior work offering rules or heuristics can be found dealing with these questions. Typically, questions associated with ternary relationships are discussed within the context of 4NF and 5NF without regard for the specifically conceptual modeling problems. Since some of the solutions to these problems can be addressed with the previously mentioned theoretical background, each direction of analysis can contribute to the strength of the other in terms of clarity and relevancy. This work seeks to address both the theoretical and practical issues surrounding N-ary relationships in conceptual modeling. Using a theoretical analysis of the construct, it seeks to provide practical answers to the understanding and use of those same constructs.

BACKGROUND

Before proceeding to the substance of the chapter, we first present certain terminology used throughout. Since some terms in this field can be interpreted differently, this section provides a solid foundation for the ensuing discussions.

In order to simplify complex arguments associated with higher orders of N-ary relationships and particularly the dynamics of multiple subsets, we restrict our discussion and analysis in this paper to ternary relationships: that is, a relationship of degree three. This level of N-ary relationship is sufficient to demonstrate the issues of relationship attribute subsets, potential contradictions between cardinalities, and relationship decomposition to lower order relationships. Secondly, we use Chen’s modeling notation. The nature and exact interpretation of the format is fully explained in the following section. Furthermore, we do not address the issue of participation since the interpretation of Chens’ structure requires that, in defining cardinality, pair wise groupings of entity values are associated with the remaining (third) entity value. Without this third entity value, the relationship cannot exist. Consequently, the corresponding participation for each cardinality definition must be total (required). Since the rule here is that, under Chen’s notation, for all entity relationships participating in the ternary, all participation constraints are total, we do not address further the lack of participation definition in these constructs.

Ternary Relationship

A ternary relationship is a relationship that contains three participating entities. Cardinalities for ternary relationships can take the form of 1:1:1, 1:1:M, 1:M:N or M:N:P. The cardinality constraint of an entity in a ternary relationship is defined by a pair of two entity instances associated with the other single entity instance. For example, in a ternary relationship R(X, Y, Z) of cardinality M:N:1, for each pair of (X, Y) there is only one instance of Z; for each pair of (X, Z) there are N instances of Y; for each pair of (Y, Z) there are M instances of X.
Semantically Constraining Binary (SCB) Relationship

A Semantically Constraining Binary (SCB) relationship defines a binary constraint between two entities participating in a ternary relationship, where the semantics of the binary relationship are associated with those of the ternary and therefore affect potential ternary combinations of entity instances. They are differentiated from Semantically Unrelated Binary (SUB) Relationships, where a binary relationship exists between two entities that also participate in a ternary relationship but where the semantic of the binary relationship is unrelated to that of the ternary. A full explanation is provided in Jones and Song (1996). An example of this model type follows. We use the notation introduced by Jones and Song for the SCB relationships which consists of a broken line as opposed to a solid line used for the more common, independent, relationships.

Consider a ternary relationship between entities Teacher, Course and Section. The relationship has a cardinality of M:1:N respectively and models the sections associated with courses teachers are currently involved. Suppose we now wish to impose the constraint that a Teacher may only teach a single Course (which is not defined by the ternary relationship). The constraint is associated with the ternary relationship to the extent it restricts potential combinations. The modeling of this situation is shown in Figure 1. The binary relationship (A) between Teacher and Course would then be a SCB relationship. Notice that this model also shows an independent relationship (B) between Teacher and Course that might reflect, for example, which courses teachers are capable of teaching.

In considering functional dependencies within ternary relationships that have imposed binary relationships, we should remember that a binary functional dependency (FD) simply identifies an existence constraint within the ternary nature of the structure. The minimal determinant for a ternary relationship must be at least a composite of two entity identifiers.

DECOMPOSITIONS OF TERNARY RELATIONSHIPS

In this chapter, we discuss the decomposition of ternary relationships into binary relationships that are lossless, functional-dependency preserving, and update preserving.

Figure 1: M:1:N ternary relationship with SCB binary relationship (A)
Jones and Song (1993), in their analysis of ternary-binary combinations, have identified that if at least one binary constraint exists within a ternary relationship structure, the ternary relationship can be losslessly decomposed to a binary relationship structure. In this section, we review the fundamental basis and dynamics of ternary relationships leading to this conclusion. However, simple lossless decomposition is not sufficient to provide complete equivalency at the conceptual, subsequent logical and physical database levels. We explore this question of equivalency and derive the allowable decompositions that provide true and complete equivalencies.

The decomposition of any modeling structure assumes that the resulting structure(s) possesses all the implicit attributes of the original structure. That is, the alternatives are at least equal to the original structure, and may be more acceptable when considering processing and data extraction overhead issues. When considering ternary relationships, we have identified three areas that we investigate to identify whether this equivalency is preserved. These are:

1. Whether the decomposition is lossless;
2. Whether the decomposition preserves functional dependencies; and
3. Whether the decomposition can equivalently deal with the practical requirements of the constraint enforcement in the physical database resulting from the model (i.e., insertions, deletions), which we call update preservation constraints.

Lossless Decompositions

In Jones and Song (1996), we find an analysis providing the fundamental basis for constraining ternary relationships through binary impositions (single and multiple constraints). Their analysis provides guidance on how the cardinalities of ternary relationships govern the allowance of additional constraints between the participating entities and the potential for further decomposition.

Implicit Ternary Cardinalities

Consider the example of a ternary relationship with cardinality of 1:1:1, where each entity has a single instance; the ternary representation is X1:Y1:Z1. No other combinations of the identified entity instances are possible, and this combination of instances satisfies the cardinality requirements. If the number of instances of each entity is now increased to two, a possible combination satisfying the cardinality requirements of 1:1:1 may appear as follows:

\[
\begin{align*}
X1 & : Y1 : Z1 \\
X2 & : Y2 : Z2
\end{align*}
\]

The relation is maintained as 1:1:1 since for (X1,Y1) there is a single instance Z1; for (X1,Z1) there is a single instance Y1; for (Y1,Z1) there is a single instance X1, and so forth.

Additionally in this relation, all binary relationships between X, Y and Z are also maintained as 1:1. Given this set of instance combinations, no other possible combinations of the identified entity instances can exist without violating the cardinality restrictions. For example, adding the tuple X1:Y1:Z2 to the relation violates the ternary cardinality of 1:1:1 by creating 1:M between the pair of (X,Y) and the single entity Z, resulting in a ternary cardinality of 1:1:M.

If the number of instances for the three entities is now legitimately increased to three, it can be demonstrated that with certain instance combinations, the binary relationships
possible between entities changes from cardinalities of 1:1, to cardinalities of M:N. The following is a relation still satisfying the 1:1:1 ternary cardinality:

\[
\begin{align*}
X1 &: Y1 &: Z1 \\
X2 &: Y2 &: Z2 \\
X1 &: Y2 &: Z3
\end{align*}
\]

Here the binary relationship between entities X and Y already is established as M:N. Relationships X:Z and Y:Z are both 1:M. Yet the ternary cardinality for the overall relation is maintained as 1:1:1. While this relation does not represent the full range of possibilities of the instances available, it does demonstrate that any of the binary relationships embedded in the ternary relationship tend towards a M:N basis.

If we now consider the relationship with no restriction on the number of entity instances, it should be relatively easy to deduce that all binary relationships within the ternary relationship can be established as M:N.

Table 1 shows a series of instances with a ternary relationship satisfying the 1:1:1 cardinality requirement. Note also that all binary relationships within the ternary relationship have now been established as M:N.

Having shown the most restrictive case (1:1:1), it should be easy to show that all other, less restrictive cardinalities demonstrate the same tendency for their implicit binary cardinalities to become M:N. Exhaustive explanations of all ternary cardinalities, with examples, are provided in Jones and Song (1996).

Following the heuristics previously demonstrated, we can conclude that the specification of a ternary cardinality constrains only the combination of entity instances, and not the cardinalities of the embedded binary relationships. Further, given that there are no explicit restrictions on the number of instances that can occur, we must allow that the possible implicit relationships between any two entities will always be M:N. We reinforce the point that the M:N relationships occurring and included in the ER diagrammatic notation are implicitly required by the ternary relationships. From this analysis, it is important to reiterate here a fundamental rule the Implicit Binary Cardinality (IBC) Rule (Jones & Song, 1996), which states: “In any given relationship, regardless of ternary cardinality, the implicit cardinalities between any two entities must be considered M:N, provided that there are no explicit restrictions on the number of instances that can occur.”

This means, fundamentally, that any unconstrained ternary relationship can not have an equivalent, binary decomposition structure (and also why unconstrained ternary relationships are not included in the analysis of decompositional equivalents in the remainder of the paper).

Table 1: 1:1:1 cardinality with all M:N binary relationships

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
</tr>
<tr>
<td>X2</td>
<td>Y2</td>
<td>Z2</td>
</tr>
<tr>
<td>X1</td>
<td>Y2</td>
<td>Z3</td>
</tr>
<tr>
<td>X4</td>
<td>Y4</td>
<td>Z3</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
X1 &: Y1 &: Z1 \\
X2 &: Y2 &: Z2 \\
X1 &: Y2 &: Z3 \\
X4 &: Y4 &: Z3
\end{align*}
\]
Restriction of Binary Relationship Imposition

Further exploring the implicit, fundamental dynamics of ternary relationships, Jones and Song (1996) demonstrate that binary constraint imposition on ternary relationships can take place only under certain situations. Consider a ternary relationship $R(X, Y, Z)$, having cardinality $M:N:P$. Additionally, consider that a binary relationship of $1:1$ is required to be imposed between entities $X$ and $Z$. A minimal set of tuples fulfilling the requirements of the $M:N:P$ ternary relationship are reproduced as follows:

\[
\begin{align*}
X_1 : Y_1 : Z_1 \\
X_1 : Y_1 : Z_2 \\
X_1 : Y_2 : Z_1 \\
X_2 : Y_1 : Z_1
\end{align*}
\]

From this set of instances, we can immediately determine that the requirements of the ternary relationship are in conflict with the requirements of the requested binary relationship imposition. The ternary requires the unique pair of instances $X_1:Y_1$ to have multiple instances of entity $Z$. Inherent in this structure is the requirement for both entities $X$ and $Y$ to individually have a relationship with $Z$ that is $1:M$. Logically then, we can not comply with the establishment of the binary requirement of $1:1$ between $X$ and $Z$.

Another example may help illustrate this finding. Can we impose a binary relationship with cardinality $M:1$, between $X$ and $Y$ that are parts of a ternary relationship having $1:M:N$ ($X:Y:Z$) cardinality? The ternary relationship ($1:M:N$) can be minimally represented as follows:

\[
\begin{align*}
X_1 : Y_1 : Z_1 \\
X_1 : Y_1 : Z_2 \\
X_1 : Y_2 : Z_1
\end{align*}
\]

Since this is a minimal representation of the $1:M:N$ cardinality, we can not remove any tuple. Note that the relationship $X:Y$ is of cardinality $1:M$; but the imposition of the binary requires that the relationship between $X$ and $Y$ be $M:1$. The binary imposition is therefore disallowed because the requirement for a single instance of $Y$ for each instance of $X$ has already been violated during construction of the ternary relationship. In other words, the cardinality of entity $Y$ in the ternary relationship could not be reduced to accommodate the binary imposition. The imposed binary relationship is in conflict with the implicit requirements of the ternary relationship.

We reiterate, then, another primary rule for testing potential combinations of ternary and binary relationships—the Explicit Binary Permission (EBP) Rule (Jones & Song, 1996): “For any given ternary relationship, a binary relationship cannot be imposed where the binary cardinality is less than the cardinality specified by the ternary, for any specific entity.”

Cardinality Constraints Following Ternary-Binary Combination

Finally, Jones and Song (1996) consider the imposition of binary relationships on ternaries that do not violate the EBP. Consider a ternary relationship having $1:1:1$ cardinality. We have established that the IBC rule implies that any binary relationship within this ternary can be considered $M:N$. By imposing a binary relationship between the two entities $X$ and $Y$ of $1:1$, for example, we must effectively restrict the number of possible instance
combinations to those abiding by the binary specifications. For example, the following relation adheres to the 1:1:1 ternary requirements:

\[ X_1 : Y_1 : Z_1 \]
\[ X_2 : Y_2 : Z_2 \]

If we now attempt to add a further tuple \((X_1, Y_2, Z_3)\), it does not comply with the imposed \(X:Y\) binary requirements of 1:1, because we have a 1:M relationship between \(X\) and \(Z\). Therefore, this combination of instances must be disallowed. The imposition of the binary is effectively restricting the possible addition of tuples to those not violating the 1:1 relationship between \(X\) and \(Z\). The cardinality of the relationship between the two entities is therefore controlled by the binary relationship and is no longer unrestricted (or M:N) as determined by the IBC Rule.

We now add one additional rule—the Implicit Binary Override (IBO) Rule (Jones & Song, 1996): “Given the imposition of a permitted binary relationship on a ternary relationship, the cardinality of the binary relationship determines the final binary cardinality between the two entities involved.”

Having now reviewed and defined the fundamental implicit dynamics of ternary relationships as they interact with binary requirements, we begin to discover the potential for further decomposition to a binary state. Jones and Song (1993) have previously investigated and determined that certain binary-ternary combinations can be losslessly decomposed to binary equivalents. In presenting the proofs, they establish the Constrained Ternary Relationship (CTD) Rule: “Any given ternary relationship cardinality can be losslessly decomposed to two binary relationships, provided that at least one 1:1 or 1:M constraint has been explicitly imposed between any two of the participating entities.”

Table 2 (Jones & Song, 1993) provides an exhaustive summary of ternary/binary combinations and lossless decomposition strategies for the various cardinality combination outcomes. The outcome cardinalities shown represent all possible results that can be obtained through any allowable combination of binary/ternary cardinalities. We note that in accordance with the CTD rule, all constrained ternary structures have a potential lossless decomposition except M:N:P.

This section, and the fundamental analysis contained, provides a basis to understanding only the issue of lossless decompositions. The following sections then, extend these arguments to include other issues paramount to the determination of full equivalence between ternary relationships and binary equivalents in both theoretical and practical environments.

**Functional Dependency Preserving Decompositions**

When we consider the decomposition of a ternary relationship into a binary relationship equivalent, we are actually considering whether the equivalent model has the ability to represent and enforce all constraints that were present in the original structure. The desired constraints in entity relationship modeling are explicitly identified through the cardinalities associated with each relationship or set of relationships. Consequently, we can test the equivalency of ternary formats against binary formats simply by comparing the implicit and explicit functional dependencies (which are derived from the cardinalities) found with each. Since the implicit cardinalities and functional dependencies may not necessarily be reflected in lossless decompositions, we should extend the investigation of decomposition to test whether functional dependency preservation is found in the lossless decompositions identified in Table 2.
We now complete this analysis and demonstrate that it is possible to have a lossless decomposition of a ternary relationship without the corresponding functional dependency preservation. The analysis of Case #1, Table 2, for functional dependency equivalence follows.

Consider a ternary relationship $R(X, Y, Z)$ with cardinality 1:1:1 and an imposed binary constraint of $M:1$ between $X:Y$. Using the notation identified previously, we can diagrammatically represent this structure, as shown in Figure 2.
The explicit constraint between X/Y implicitly suggests the following set of functional dependencies:

\[ XY \rightarrow Z, \ XZ \rightarrow Y, \ YZ \rightarrow X, \ X \rightarrow Y, \ X \rightarrow Z, \ (X \rightarrow Z \text{ is implicitly derived from } X \rightarrow Y \text{ and } XY \rightarrow Z). \]


According to Table 2 and the CTD rule, we may use a binary decomposition of (XY)(XZ), as shown in Figure 3.

Also consider a set of instances complying with the same cardinality requirements (Example 1).

This relation and population may be losslessly decomposed to Example 2.

---

**Figure 2: 1:1:1 ternary relationship with M:1 (XY) constraint**

![Diagram](image1.png)

**Figure 3: Suggested decomposition for Figure 2**

![Diagram](image2.png)

**Example 1: Ternary relationship instance set**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
</tr>
<tr>
<td>X2</td>
<td>Y1</td>
<td>Z2</td>
</tr>
<tr>
<td>X3</td>
<td>Y2</td>
<td>Z1</td>
</tr>
</tbody>
</table>
Example 2: Decomposed ternary storage structure

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
</tr>
<tr>
<td>X2</td>
<td>Y1</td>
</tr>
<tr>
<td>X3</td>
<td>Y2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>X</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Z1</td>
</tr>
<tr>
<td>X2</td>
<td>Z2</td>
</tr>
<tr>
<td>X3</td>
<td>Z1</td>
</tr>
</tbody>
</table>

As previously identified (Table 2), this decomposition is lossless. But are the two storage structures equivalent from the perspective of FD preservation? Consider the same decompositional scenario from a functional dependency perspective. We have previously defined the set of functional dependencies that were present in the original ternary structure. In the decomposed structure, we identify the following functional dependency set:

- $X \rightarrow Y$, $X \rightarrow Z$, $XY \rightarrow Z$ (augmentation), $XZ \rightarrow Y$ (augmentation)

We note that in this decomposition strategy, we lose the functional dependency of $YZ \rightarrow X$, and there is no way to recover it through reconstruction based on Armstrong’s Axioms (Armstrong, 1974). This supports the observation that a binary decomposition is not always able to enforce functional dependency constraints, even though the decomposition may be lossless.

In scrutinizing all potential cardinality outcomes and applying the same functional dependency and decomposition analysis, we find that three additional combinations (Cases 1, 6, 11) cannot be decomposed without losing some level of functional constraint enforcement. Table 3 shows all possible combinations together with their decompositional status regarding functional dependency preservation. This table is then an extension of Table 2, where only the join and losslessness were considered and every combination qualified. We also notice in comparing the two sets of outcomes, that some of the alternate decompositions allowed in Table 2 do not preserve all functional dependencies and are therefore disqualified in the more restrictive Table 3.

**Update Constraint Preservation**

So far, in investigating dependencies within these alternate structures, we have considered only the ability to decompose static structures from a lossless and FD preserving perspective. That is, we have looked only at the ability to preserve a specific set of instances during decomposition and re-composition. We have not considered the ability of a binary model (as opposed to a ternary structure) to equivalently handle insertions and deletions of ternary relationships (dynamic decompositions), which may be reflected in the preservation of functional dependencies. We have simply identified that certain static structures have equivalent lossless forms. The additional consideration of updates, which are of significant importance to creating realistic database models, are investigated in this section.

**Insertion Constraints**

Let us consider the way the alternative storage structures (ternary vs. binary) allow insertion of similar tuples. We should keep in mind that we are comparing the ability of the
### Table 3: Lossless and FD preserving decompositions

<table>
<thead>
<tr>
<th>Case #</th>
<th>Ternary Cardinality (X:Y:Z)</th>
<th>Binary Impositions</th>
<th>Potential Lossless Decomposition</th>
<th>Potential FD Preserving Decompositions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:1:1</td>
<td>(X:Y) = (M:1)</td>
<td>(XY)(XZ)</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>1:1:1</td>
<td>(X:Y) = (1:1)</td>
<td>(XY)(XZ) -or- (XY)(YZ)</td>
<td>(XY)(XZ) -or- (XY)(YZ)</td>
</tr>
<tr>
<td>3</td>
<td>1:1:1</td>
<td>(X:Y) = (M:1) (Z:Y) = (M:1)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
</tr>
<tr>
<td>4</td>
<td>1:1:1</td>
<td>(X:Y) = (M:1) (X:Z) = (1:1)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
</tr>
<tr>
<td>5</td>
<td>M:1:1</td>
<td>(X:Y) = (M:1)</td>
<td>(XY)(XZ)</td>
<td>(XY)(XZ)</td>
</tr>
<tr>
<td>6</td>
<td>M:1:1</td>
<td>(Y:Z) = (M:1)</td>
<td>(XY)(YZ)</td>
<td>None</td>
</tr>
<tr>
<td>7</td>
<td>M:1:1</td>
<td>(Y:Z) = (1:1)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
</tr>
<tr>
<td>8</td>
<td>M:1:1</td>
<td>(X:Y) = (M:1) (Y:Z) = (1:1)</td>
<td>(XY)(YZ) -or- (XZ)(ZY) -or- (XY)(XZ)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
</tr>
<tr>
<td>11</td>
<td>M:N:1</td>
<td>(X:Z) = (M:1) (Y:Z) = (M:1)</td>
<td>(XY)(XZ) -or- (XY)(YZ)</td>
<td>None</td>
</tr>
<tr>
<td>12</td>
<td>M:N:P</td>
<td>Not Allowed</td>
<td>None</td>
<td>None</td>
</tr>
</tbody>
</table>

structures to ensure enforcement of all constraints present (typically identified by the implicit functional dependencies).

Consider the act of requesting an insertion of tuple R1(X4, Y1, Z1) into the database. In the original format of structuring the ternary relationship (Figure 2, Example 1, and Case #1, Table 3), this insertion would be disallowed because it violates the constraint that a pair of YZ values must be associated with a single value for X via FD Y1, Z1 → X1. In the decomposed structure (Figure 3 and Example 2), the tuple would be decomposed and inserted in the form (X4, Y1) and (X4, Z1). These insertions into the binary equivalents would be accepted without violating any constraints of the binary structures. Checking the constraints
YZ → X in the binary equivalents requires joins between two decomposed tables. We therefore have an obvious difference between the way the two structural representations would allow insertions.

The discussion here deserves two important observations. The first issue concerns the recognition and identification of a constraint at schema level. That is, while in the ternary format the functional dependency YZ → X can be observed and identified at the schema level, thus improving the modeling power; in the binary equivalents this constraint is not easily observed and identified. There is less opportunity for designers to identify the constraint in the binary equivalents. The second aspect concerns the enforcement of the constraint. Checking the constraint YZ → X does not require a join in the ternary structures, but requires a join in the binary equivalents. The binary equivalent format could cause inconsistency during insertion if not checked at all or degrade performance when checking the constraint using a join.

The conclusion is that not every ternary relationship can be decomposed to binary equivalents without losing the constraint and modeling power of original ternary relationships.

**Deletion Constraints**

Let us now move on to consider the same arguments with respect to binary alternatives correctly providing for deletion of tuples. Consider the ternary relationship R(X, Y, Z) with cardinality M:1:1 (Case #8), and binary impositions M:1 (X, Y) and 1:1 (Y, Z). According to Table 3, this relation can be lossless and FD preserving, decomposed to its binary equivalent of S(X, Y) and T(Y,Z). An example of the original table and decomposition, with appropriate instances, is shown in Example 3 and Example 4, below. Example 3 shows data as stored in the structure derived from a ternary relationship. Example 4 shows the data stored in the decomposed, functional dependency preserving alternative structure (see also Table 3). If we now attempt to delete tuple R1(X1, Y1, Z1), the ternary storage structure dictates that we have tuples R2(X2, Y1, Z1) and R3(X4, Y3, Z2) remaining. However, if we delete the same semantic combination of data, R1(X1, Y1, Z1), from the equivalent binary storage format, we must delete tuples S1(X1, Y1) and T1(Y1, Z1), based on the projections of R1. We have now deleted the only reference to T1(Y1, Z1). This tuple is in fact required to reconstruct the remaining relationship R2(X2, Y1, Z1). In other words, after the binary deletions, we are unable to reconstruct the expected tuple R2(X2, Y1, Z1).

Another simple example helps to underscore this point. Let us assume we wish to delete tuple R3(X4, Y3, Z2). In Example 3, this is done by removing the complete tuple, with no additional consideration to any associated affects on the data. In the decomposed form, the same tuple is stored in its binary components. In this format, however, we cannot simply delete the two binary projections of R3 without first checking to see if either is required by additional binary components elsewhere in the schema. That is, we cannot delete (X4, Y3) from Relation S without first checking to see if multiple values of Y3 exist in Relation T, not just the tuple (Y3, Z2). In our example here, since there is only a singular reference to Y3 as a foreign key, tuple (X4, Y3) could be deleted. However, if there were additional references to Y3 in Relation T, tuple (X4, Y3) could not be deleted as it would indicate that an additional ternary relationship tuple, consisting of (X4, Y3, Z#), may be required and could not be reconstructed without the preservation of (X4, Y3).

We therefore observe that, although the decomposition is equivalent from the point of view of lossless and FD-preserving constraints, from a practical standpoint, the binary structure does not always have the implicit semantic constraint checking properties of the ternary structure in the case of deletions. We have no ability to preserve the necessary binary
Example 3: M:1:1 ternary relationship with binary impositions (Relation_R)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td>Z1</td>
</tr>
<tr>
<td>X2</td>
<td>Y1</td>
<td>Z1</td>
</tr>
<tr>
<td>X4</td>
<td>Y3</td>
<td>Z2</td>
</tr>
</tbody>
</table>

Example 4: Lossless & FD preserving decomposition of Example 3 (Relation_S and Relation_T)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X1</td>
<td>Y1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X2</td>
<td>Y1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X4</td>
<td>Y3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

relationships without some additional mechanism by application programs to check reference requirements between tables. Additionally, even if we were to use additional constraint checking, since the original ternary relationship data now exists only as a set of binary combinations, we have no reference point to verify that any other ternary combinations should exist. For example, in the above case, how do we know in the future that tuple R₁ may even need to be reconstructed? The binary structure does not automatically provide the storage requirements to allow deletions without potential loss of data or semantic integrity. In fully analyzing which ternary/binary cardinality combinations fully preserve constraint checking on deletions, we find there are a total of 8 cases (see Table 4).

Exhaustive Analysis

Table 4 (which extends Table 3) provides an exhaustive analysis of all ternary-binary combinations and the full results of which are lossless, FD preserving, and update preserving with insertion and deletion constraints. We observe from this analysis that given all combinations of binary/ternary cardinality combinations, only four of the combinations and subsequent potential binary decompositions provide losslessness, functional dependency preservation, and update constraint (insert and delete) preservation. Consequently, only these four structures can be considered fully equivalent to the structure derived from the ternary relationship.

CONCLUSION

Ternary relationships are an established and accepted construct within the realm of (N-ary) conceptual modeling. However, very little research has been completed that investigates the logical complexity of the construct, particularly when combined with additional, internal
Table 4: Implicit semantic constraint enforcement

<table>
<thead>
<tr>
<th>Case #</th>
<th>Ternary Cardinality (X:Y:Z)</th>
<th>Binary Impositions</th>
<th>Potential Lossless Decomposition</th>
<th>Potential FD Preserving Decomposition</th>
<th>Enforces Semantic Constraints on Insertions</th>
<th>Enforces Semantic Constraints on Deletions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:1:1 (X:Y) = (M:1)</td>
<td>(XY)(XZ)</td>
<td>None</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>1:1:1 (X:Y) = (1:1)</td>
<td>(XY)(XZ) -or- (XY)(YZ)</td>
<td>(XY)(XZ) -or- (XY)(YZ)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1:1:1 (X:Y) = (M:1) (Z:Y) = (M:1)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1:1:1 (X:Y) = (M:1) (X:Z) = (1:1)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
<td>(XY)(XZ) -or- (XZ)(ZY)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>M:1:1 (X:Y) = (M:1)</td>
<td>(XY)(XZ)</td>
<td>(XY)(XZ)</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>M:1:1 (Y:Z) = (M:1)</td>
<td>(XY)(YZ)</td>
<td>None</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>M:1:1 (Y:Z) = (1:1)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>M:1:1 (X:Y) = (M:1) (Y:Z) = (1:1)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
<td>(XY)(YZ) -or- (XZ)(ZY)</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>M:1:1 (X:Y) = (M:1) (Y:Z) = (1:M)</td>
<td>(XY)(YZ) -or- (XY)(XZ)</td>
<td>(XY)(YZ)</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>M:N:1 (X:Z) = (M:1)</td>
<td>(XY)(XZ)</td>
<td>(XY)(XZ)</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>M:N:1 (X:Z) = (M:1) (Y:Z) = (M:1)</td>
<td>(XY)(XZ) -or- (XY)(YZ)</td>
<td>None</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>12</td>
<td>M:N:P</td>
<td>Not Allowed</td>
<td>None</td>
<td>None</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

constraints, which are termed SCB relationships. There is a valid question of whether ternary constructs that have been semantically constrained have fully equivalent binary decompositions. We have shown in this paper, that logical (fully equivalent) decompositions do exist for certain combinations of ternary / binary cardinalities, but the majority do not have fully (logical and practical) equivalents.

This work, in addition to providing a basis for manipulating ternary relationships, provides important facts concerning the implicit nature of the constructs, and therefore offers key insights into methodologies and mechanisms for dealing with them. Other works that extend the scope of the fundamental issues presented here include McAllister (1996) which attempts to provide a unification of notations through cardinality tables. This work is particularly important since it offers a proof and methodology of comprehending orders of relationships higher than 3. Dullea and Song (1999) discuss the issues of structural validity and contradictory constraints as they apply to recursive relationships, as well as the redundancy of relationships between binary structures. None of these papers readily addresses the issue of participation constraint inclusion as they are currently adopted and interpreted in the multiple notations available.
If we accept that ternary relationships are “absolute,” that is they possess some singular representation in data modeling, then true equivalents of the structure must follow those derivations provided by the contents of this chapter. We have shown that certain constraint combinations have no correct decomposition from a theoretical or practical perspective. We therefore deduce that arguments concerning “alternative” representations of such constructs are invalid or contain additional and superfluous overhead required to represent the inherent properties of the original N-ary form. In binary modeling, the creation of “entities,” with composite keys (“gerunds,” that substitute for ternary relationships) allows only partial representation of all possible semantics, since they do not allow graphical modeling of the intra-entity binary constraints (SCB relationships) which may be present within a ternary relationship. Furthermore, we have demonstrated that the simple preservation of functional dependencies in a decomposition does not always support the valid join of two projections if some level of update is involved between the decomposition and join (i.e., a join of two projections that preserve functional dependencies is only valid from an instance level, but cannot be extended to the database’s implicit constraint level).

Although the arguments concerning insertions and deletions may be categorized as view update problems, we maintain that view update problems are derived from the creation of logical and physical aspects (subsets) of a database. What we suggest here is that the logical interpretation of the conceptual model is affected by the physical structure chosen in the final design. This need for translation reflects problems of maintaining functional dependencies between attributes in a model but potentially destroying the semantics of a particular (ternary) structure. We have demonstrated that the same sets of constraints that are associated with the semantics are not necessarily reflected in the (theoretical) preservation of the functional dependencies.

In conclusion, we suggest that if indeed the ternary relationship is considered a bone fide structure, modeling notations, and any derivations of the conceptual models containing ternary relationship structures, must recognize and fully implement the correct logic and semantics implicit to these types of structures. This includes modeling notations and the heuristics built into automated (CASE) tools. Towards this objective, this chapter has identified the practical and theoretical underpinnings for this position and provides a decompositional framework (Table 4) to assist practitioners who may have to deal with these issues.

REFERENCES


Redwood City, CA: Benjamin/Cummings.


Chapter III

Object-Oriented Database Benchmarks

Jérôme Darmont
Université Lumiére, France

Michel Schneider
Université Blaise Pascal, France

We present in this chapter an overview of the benchmarks aimed at evaluating the performances of Object-Oriented Databases (OODBs). We particularly focus on the Object-Clustering Benchmark (OCB), which is both generic and originally designed to specifically evaluate the performances of clustering policies within OODBs. OCB is generic because its sample database may be customized to fit any of the databases introduced by the main previous benchmarks, e.g., Object Operation 1 (OO1) or OO7. The first version of OCB was purposely clustering-oriented due to a clustering-oriented workload, but OCB has been thoroughly extended to be able to suit other purposes. Eventually, OCB’s code is compact and easily portable. OCB has been validated through two implementations: one within the O2 OODB and another one within the Texas persistent object store. The performances of two different clustering policies implemented in Texas, Dynamic, Statistical, Tunable Clustering (DSTC) and Detection & Reclustering of Objects (DRO), have also been compared with OCB.

The need to evaluate the performances of Object-Oriented Database Management Systems (OODBMSs) is important both to designers and users. Performance evaluation is useful to designers to determine elements of architecture, choose between caching strategies, and select Object Identifier (OID) type, among others. It helps them validate or refute hypotheses regarding the actual behavior of an OODBMS. Thus, performance evaluation is an essential component in the development process of efficient and well-designed object stores. Users may also employ performance evaluation, either to compare the efficiency of different technologies before selecting an OODBMS or to tune a system.

The main work presented in this chapter was initially motivated by the evaluation of object clustering techniques. The benefits induced by such techniques on global performances are widely acknowledged and numerous clustering strategies have been proposed.
As far as we know, there is no generic approach allowing for their comparison. This problem is interesting for both designers (to set up the corresponding functionalities in the system kernel) and users (for performance tuning).

There are different approaches used to evaluate the performances of a given system: experimentation, simulation, and mathematical analysis. We focus only on the first two approaches. Mathematical analysis is not considered because it invariably uses strong simplification hypotheses (Benzaken, 1990; Gardarin, Gruser, & Tang, 1995) and its results may well differ from reality.

Experimentation on the real system is the most natural approach and a priori the simplest to complete. However, the studied system must have been acquired, installed, and have a real database implanted in it. This database must also be significant of future exploitation of the system. Total investment and exploitation costs may be quite high, which can be drawbacks when selecting a product.

Simulation is casually used in substitution or as a complement to experimentation. It does not necessitate installing nor acquiring the real system. It can even be performed on a system still in development (a priori evaluation). The execution of a simulation program is generally much faster than experimentation. Investment and exploitation costs are very low. However, this approach necessitates the design of a functioning model for the studied system. The reliability of results directly depends on the quality and the validity of this model. Thus, the main difficulty is to elaborate and validate the model. A modeling methodology can help and secure these tasks.

Experimentation and simulation both necessitate a workload model (database and operations to run on this database) and a set of performance metrics. These elements are traditionally provided by benchmarks. Though interest for benchmarks is well recognized for experimentation, simulation approaches usually use workloads that are dedicated to a given study, rather than workloads suited to performance comparisons. We believe that benchmarking techniques can also be useful in simulation. Benchmarking can help validate a simulation model as compared to experimental results or support a mixed approach in which some performance criteria necessitating precision are measured by experimentation and other criteria that does not necessitate precision are evaluated by simulation.

There is no standard benchmark for OODBs, even if the more popular benchmarks, OO1, HyperModel, and OO7 are de facto standards. These benchmarks are aimed at engineering applications (Computer-Aided Design, Manufacturing, or Software Engineering). These general purpose benchmarks feature quite simple databases and are not well suited to the study of clustering, which is very data dependent. Many benchmarks have been developed to study particular domains. A fair number of them are more or less based on OO1, HyperModel, or OO7.

In order to evaluate the performances of clustering algorithms within OODBs, a new benchmark was designed: OCB (Darmont, Petit, & Schneider, 1998). It originally had a generic object base and was clustering oriented through its workload. It actually appeared afterwards that OCB could become more general, provided the focused workload was extended, as described in this chapter.

Full specifications for a new version of OCB are provided in this chapter. More precisely, we address the following points: the generalization of the OCB workload so that the benchmark becomes fully generic, a comparison of OCB to the main existing benchmarks, and a full set of experiments performed to definitely validate OCB. These performance tests were performed on the O2 OODB (Deux, 1991), the Texas persistent object store (Singhal, Kakkad, & Wilson, 1992), and the DSTC (Bullat & Schneider, 1996) and the DRO
(Darmont, Fromantin, Régnier, Gruenwald, & Schneider, 2000) clustering techniques, that are both implemented in Texas. The results obtained are discussed in this chapter.

We are aware of the legal difficulties pointed out by Carey, Dewitt, and Naughton (1993) and Carey, Dewitt, Kant, and Naughton (1994). Indeed, OODBMS vendors are sometimes reluctant to see benchmark results published. The objective of our effort is rather to help software designers or users evaluate the adequacy of their product in a particular environment. OCB should also prove useful at least for researchers, to benchmark OODB prototypes and/or evaluate implementation techniques.

The remainder of this chapter is organized as follows. The de facto standards in object-oriented benchmarking are presented (OO1, HyperModel, and OO7; as well as the Justitia benchmark, which is interesting due to its multi-user approach). Next, the OCB benchmark is described and compared to the other benchmarks. Experiments performed to validate OCB are also presented. The chapter is concluded with future trends.

STATE OF THE ART

The OO1 Benchmark

OO1, also referred to as the “Cattell Benchmark” (Cattell, 1991), was developed early in the 1990s when there was no appropriate benchmark for engineering applications. OO1 is a simple benchmark that is very easy to implement. It was used to test a broad range of systems including object-oriented DBMS, relational DBMS, and other systems such as Sun’s INDEX (B-tree based) system. The visibility and simplicity of OO1 provide a standard for OODB benchmarking. A major drawback of this tool is that its data model is too elementary to measure the elaborate traversals that are common in many types of object-oriented applications, including engineering applications. Furthermore, OO1 only supports simple navigational and update tasks and has a limited notion of complex objects (only one composite hierarchy).

**OO1 Database**

OO1’s database (Figure 1) is built upon two classes: Part and Connection. The parts are composite elements that are connected (through Connection objects) to three other parts. Each connection references two parts: the source (From) and the destination part (To).

*Figure 1: OO1 database schema*
The database is generated as follows:
1. Create all the Part objects and store them into a dictionary;
2. For each part, randomly choose three other parts and create the associated connections. Locality of reference (objects are often linked to relatively close objects) is simulated by a reference zone; i.e., Part #i is randomly linked to parts whose Id are in the interval \([Id-RefZone, Id+RefZone]\). The probability that the links are determined this way is 0.9. Otherwise, the linked parts are chosen totally at random.

**OO1 Operations**

OO1 performs three types of operations. Each of them is run 10 times. Response time is measured for each run.

- **Lookup**: access to 1000 randomly selected parts.
- **Traversal**: randomly select a root part, then explore the corresponding part tree (in depth first) through the Connect and To references, up to seven hops (total of 3280 parts, with possible duplicates). Also perform a reverse traversal by swapping the To and From directions.
- **Insert**: add 100 parts, and the corresponding connections, to the database. Commit the changes.

**The HyperModel Benchmark**

The HyperModel Benchmark (Anderson, Berre, Mallison, Porter, \& Sceider, 1990), also referred to as the Tektronix Benchmark, is recognized for the richness of the tests it proposes. HyperModel possesses both a richer schema and a wider extent of operations than OO1. This renders it potentially more effective than OO1 in measuring the performances of engineering databases. However, this added complexity also makes HyperModel harder to implement, especially since its specifications are not as complete as OO1’s. The presence of complex objects in the HyperModel Benchmark is limited to a composition hierarchy and two inheritance links. The scalability of HyperModel is also not clearly expressed in the literature, whereas other benchmarks explicitly support different and well-identified database sizes.

**HyperModel Database**

The HyperModel Benchmark is based on an extended hypertext model. Hypertext is a generic graph structure consisting of nodes and links (Figure 2). The main characteristic of this database is the various relationships existing between classes: inheritance (the attributes of a Node object may be inherited from another Node object), aggregation (an instance of the Node class may be composed of one or several other instances), and eventually association (two Node objects may be bound by an oriented link).

**HyperModel Operations**

The benchmark consists of 20 operations. To measure the time to perform each operation, the following sequence is followed.

1. **Setup**: prepare 50 inputs to the operations (the setup is not timed);
2. **Cold run**: run the operation 50 times, on the 50 inputs precomputed in the setup phase; then, if the operation is an update, commit the changes once for all 50 operations;
3. **Warm run**: repeat the operation 50 times with the same input to test the effect of caching; again, perform a commit if the operation was an update.
The 20 possible operations belong to seven different kinds:

- **Name Lookup:** retrieve one randomly selected node;
- **Range Lookup:** retrieve the nodes satisfying a range predicate based on an attribute value;
- **Group Lookup:** follow the relationships one level from a randomly selected starting node;
- **Reference Lookup:** reverse Group Lookup;
- **Sequential Scan:** visit all the nodes;
- **Closure Traversal:** Group Lookup up to a predefined depth;
- **Editing:** update one node.

### The OO7 Benchmark

OO7 (Carey et al., 1993) is a more recent benchmark than OO1 and HyperModel. It reuses their structures to propose a more complete benchmark and to simulate various transactions running on a diversified database. It has also been designed to be more generic than its predecessors and to correct their weaknesses in terms of object complexity and associative accesses. This is achieved with a rich schema and a comprehensive set of operations.
However, if OO7 is a good benchmark for engineering applications, it is not for other types of applications such as financial, telecommunication, and multimedia applications (Tiwary, Narasayya, & Levy, 1995). Since its schema is static, it cannot be adapted to other purposes. Eventually, the database structure and operations of OO7 are nontrivial. Hence, the benchmark is quite difficult to understand, adapt, or even implement. Yet, to be fair, OO7 implementations are available by anonymous FTP.¹

**OO7 Database**

OO7’s database is based on a conceptual model that is very close to the HyperModel Benchmark’s, though it contains a higher number of classes (Figure 3). Four kinds of links are also supported (IS-A, 1-1, 1-N, M-N).

**OO7 Operations**

The range of transactions offered by OO7 is also close to HyperModel’s. Three main groups may be identified:

- *Traversals* browse the object graph using certain criteria. These traversals are very close to OO1’s. There are ten different operations that apply depending on the database characteristics (basically, its size);

---

Figure 3: OO7 database schema

![Figure 3: OO7 database schema](image-url)
• Queries retrieve objects chosen in function of various criteria. There are eight kinds of queries;
• Structural Modification Operations deal with object insertion and deletion (two operations).

The Justitia Benchmark

Justitia (Schreiber, 1994) has been designed to address the shortcomings of existing benchmarks regarding multi-user functionality, which is important in evaluating client-server environments. Justitia is also aimed at testing OODB capacity in reorganizing its database.

Because Justitia’s published specifications lack precision, the author’s work cannot be easily reused. Furthermore, taking multiple users into account renders the benchmark quite complex. Justitia is fairly tunable and supposed to be generic, but it still uses structures that are typical of engineering applications. Its database schema is more limited than those of HyperModel or OO7. Though the object types are diverse, inter-class relationships are very few. The inheritance graph is substantial, but other types of references are limited to composition.

Justitia Database

Justitia’s database structure is a tree of bi-directional 1-N relationships interleaved with an inheritance graph (Figure 4). The basic classes that compose it are inspired by engineering application data structures:
• Dynamic-sized objects (DO) have a random size, which immunizes the benchmark against OODBs performing optimizations regarding a given object size;
• Static-sized objects (SO) represent typical objects; their size is available to the system, that can use it for optimization purposes;
• Container objects (CO) are nodes pointing each to a ring of references toward primitive objects (PO) that represent complex, internal objects.

Figure 4: Justitia database schema
**Justitia Operations**

All the benchmark operations are based on four basic transactions that can be combined to form composite operations. The basic operations use elementary access methods and model simple accesses.

- **Simple read**: Access a set of objects and read their date attribute.
- **Simple update**: Access a set of objects and update their date attribute.
- **Extended read**: Access a set of objects, read their date attribute, and execute an operation depending on their class.
- **Extended update**: Access a set of objects, update their date attribute, and execute an operation depending on their class.

The composite operations are defined by the user through a series of parameters. Their aim is to test the efficiency of the multi-user mode.

**THE OBJECT-CLUSTERING BENCHMARK**

Originally, the purpose of OCB was to test the performances of clustering algorithms within object-oriented systems. OCB is structured around a rich object base including many different classes and numerous types of references allowing the design of multiple interleaved hierarchies. This database is wholly generic. The OCB workload, once clustering oriented, has been extended with relevant, significant, and reproducible transactions. Thus, the workload became fully generic.

The flexibility of OCB is achieved through an extensive set of parameters. Many different kinds of object bases can be modeled with OCB as well as many different kinds of applications running on these databases. This is an important feature since there exists no canonical OODB application. OCB can indeed be easily parameterized to model a generic application or dedicated to a given type of object base and/or application. OCB is also readily scalable in terms of size and complexity resulting in a wide range of object bases. Usage time can be set up as well to be rather short or more extensive. Moreover, OCB’s parameters are easy to set up.

OCB’s code is very compact and easily implemented on any platform. OCB is currently implemented in C++ to benchmark O2 and Texas. Both versions are freely available. The C++ code is less than 1,500 lines long. OCB has also been ported into QNAP2 and C++ simulation models. QNAP2 is a simulation software that supports a non object-oriented language close to Pascal. The QNAP2 code dealing with OCB is shorter than 1,000 lines.

The next version of OCB, which is currently in development, shall support multiple users viewed as processes in a very simple way to test the efficiency of concurrency control. As far as we know, Justitia is the only benchmark to have actually addressed this problem, though OO7 also has a multi-user version in development. O01 was designed as multi-user, but the published results only involve a single user. One of our research objectives is to provide clear specifications for our benchmark so that others can readily implement it and provide feedback to improve it.

**OCB Database**

The OCB database is highly generic because it is rich, simple to achieve, and very tunable. It is made of a predefined number of classes (NC) derived from the same metaclass (Figure 5). A class has a unique logical identifier, Class_ID, and is defined by two parameters: MAXNREF, the maximum number of references in the class’...
instances; and BASESIZE, an increment size used to compute the InstanceSize after the inheritance graph is processed at database generation time. In Figure 5, note that the UML « bind » clause indicates that classes are instantiated from the metaclass using the parameters between brackets.

Since different references can point to the same class, 0-N, 1-N, and M-N links are implicitly modeled. Each of these CRef references has a type, TRef. There are NTREF different types of references. A reference type can be, for instance, a type of inheritance, aggregation, or user association. Eventually, an Iterator is maintained within each class to save references toward all its instances.

Objects in the database (instances of class OBJECT) are characterized by a unique logical identifier OID and by their class through the ClassPtr pointer. Each object possesses ATTRANGE integer attributes that may be read and updated by transactions. A string of size InstanceSize, the Filler, simulates the actual size the object should occupy on disk.

After instantiating the database schema, an object O of class C points through the ORef references to at most MAXNREF objects. These objects are selected from the iterator of the class referenced by C through the corresponding CRef reference. For each direct reference identified by an ORef link from an object oi toward an object oj, there is also a backward reference (BackRef) from oj to oi.

The database generation proceeds through three primary steps:
1. Instantiation of the CLASS metaclass into NC classes: Creation of the classes without any reference, then selection of the classes referenced by each class. The type of the references (TRef) can either follow the DIST1 random distribution or be set up a priori.

Figure 5: OCB database schema
The referenced classes belong to the $[\text{Class\_ID} - \text{CLOCREF}, \text{Class\_ID} + \text{CLOCREF}]$ interval that models a certain locality of reference, as introduced by OO1, but at the class level. The class reference selection can either follow the DIST2 random distribution or be set up a priori. NIL references are possible.

2. Database consistency check-up: Suppression of all the cycles and discrepancies within the graphs that do not allow them, e.g., inheritance graphs or composition hierarchies.

3. Instantiation of the NC classes into NO objects: Creation of the objects, without any reference—their class follows the DIST3 random distribution, then random selection of the objects referenced by each object. The referenced objects belong to the $[\text{OID} - \text{OLOCREF}, \text{OID} + \text{OLOCREF}]$ interval that models a certain locality of reference at the instance level. The random selection of object references follows the DIST4 random distribution. Reverse references (BackRef) are instantiated when the direct links are instantiated.

The random numbers are generated by the Lewis-Payne random generator (Lewis & Payne, 1973), which is one of the best pseudorandom number generators currently available. The database parameters are summarized in Table 1.

**OCB Operations**

The core of the workload is organized around several transactions, the *traversals*, that are able to explore the effects of clustering. Several operations that do not benefit from any clustering effort have been re-introduced, e.g., creation and update operations. A full description of the benchmark’s operations follows.

- **Random Access**: Access to NRND objects following the DIST5 random distribution.
- **Sequential Scan**: Browse the instances of a class following the DIST6 random distribution (Simple Scan). A Range Lookup additionally tests the value of NTEST attributes in each instance.
- **Traversals**: Traversal operations are divided into two types: Set-Oriented Accesses (or Associative Accesses) and Navigational Accesses, which have been empirically found by McIver and King (1994) to match breadth-first and depth-first traversals, respectively. Navigational Accesses are further divided into Simple, depth-first traversals, Hierarchy Traversals that always follow the same type of reference, and finally Stochastic Traversals that randomly select the next link to cross. Stochastic

<table>
<thead>
<tr>
<th>Parameter name</th>
<th>Parameter</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>Number of classes in the database</td>
<td>50</td>
</tr>
<tr>
<td>MAXNREF (i)</td>
<td>Maximum number of references, per class</td>
<td>10</td>
</tr>
<tr>
<td>BASESIZE (i)</td>
<td>Instances base size, per class</td>
<td>50 bytes</td>
</tr>
<tr>
<td>NO</td>
<td>Total number of objects</td>
<td>20,000</td>
</tr>
<tr>
<td>NREFT</td>
<td>Number of reference types (inheritance, aggregation, etc.)</td>
<td>4</td>
</tr>
<tr>
<td>ATTRANGE</td>
<td>Number of integer attributes in an object</td>
<td>1</td>
</tr>
<tr>
<td>CLOCREF</td>
<td>Class locality of reference</td>
<td>NC</td>
</tr>
<tr>
<td>OLOCREF</td>
<td>Object locality of reference</td>
<td>NO</td>
</tr>
<tr>
<td>MAXRETRY</td>
<td>Maximum number of retries when linking objects</td>
<td>3</td>
</tr>
<tr>
<td>DIST1</td>
<td>Reference types random distribution</td>
<td>Uniform</td>
</tr>
<tr>
<td>DIST2</td>
<td>Class references random distribution</td>
<td>Uniform</td>
</tr>
<tr>
<td>DIST3</td>
<td>Objects in classes random distribution</td>
<td>Uniform</td>
</tr>
<tr>
<td>DIST4</td>
<td>Objects references random distribution</td>
<td>Uniform</td>
</tr>
</tbody>
</table>

Table 1: OCB database parameters
traversals effectively simulate the access patterns caused by real queries, according to Tsangaris and Naughton (1992). An object bears at most \( \text{MAXNREF} \) references numbered from 1 to \( \text{MAXNREF} \). At each step of a stochastic traversal, the probability to follow reference number \( N(\text{[1, MAXNREF]}) \) is \( p(N) = \frac{1}{2^N} \). Each type of traversal proceeds from a root object following the \( \text{DIST7} \) random distribution and up to a predefined depth depending on the traversal type. All these transactions can be reversed to follow the links backward, “ascending” the graphs.

- **Update**: Update operations are also subdivided into different types. *Schema Evolutions* deal with individual insertion and deletion of *Class* objects. The class to be deleted follows the \( \text{DIST8} \) random distribution. *Database Evolutions* deal with individual insertion and deletion of objects. The object to be deleted follows the \( \text{DIST9} \) random distribution. Eventually, *Attribute Updates* allow attribute changes, either of random-accessed objects (Random Update of \( \text{NUPDT} \) objects following the \( \text{DISTA} \) random distribution) or of instances of a class following the \( \text{DISTB} \) random distribution.

The execution of transactions by each client (the benchmark is to be multi-user) is organized according to the following protocol:

1) **cold run** of \( \text{COLDN} \) transactions whose types are determined randomly according to predefined probabilities. The purpose of this step is to fill in the cache in order to observe the real, stationary behavior of the clustering algorithm implemented in the benchmarked system;

2) **warm run** of \( \text{HOTN} \) transactions.

A latency time \( \text{THINK} \) can be introduced between each transaction run. Furthermore, the whole benchmark execution may be replicated so that the same set of transactions is executed on different randomly generated object bases. This feature allows the computation of mean values and confidence intervals, which are typically more significant than a single measurement. The OCB workload parameters are summarized in Table 2.

The metrics measured by OCB are basically:

- database response time (global and per transaction type) and throughput. In a client-

<table>
<thead>
<tr>
<th>Parameter(s) name(s)</th>
<th>Parameter(s)</th>
<th>Default value(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{NRND} )</td>
<td>Number of objects accessed in Random Accesses</td>
<td>50</td>
</tr>
<tr>
<td>( \text{NTEST} )</td>
<td>Number of attributes tested in Range Lookups</td>
<td>1</td>
</tr>
<tr>
<td>( \text{SETDEPTH}, \text{SIMDEPTH}, \text{HIEDEPTH}, \text{STODEPTH} )</td>
<td>Depth: Set-oriented Access, Simple Traversal, Hierarchy Traversal, Stochastic Traversal</td>
<td>3, 3, 5, 50</td>
</tr>
<tr>
<td>( \text{NUPDT} )</td>
<td>Number of updated objects in Random Updates</td>
<td>50</td>
</tr>
<tr>
<td>( \text{DIST5}, \text{DIST6}, \text{DIST7}, \text{DIST8}, \text{DIST9}, \text{DISTA}, \text{DISTB} )</td>
<td>Random distribution law: Random Access objects, Sequential Scan classes, Transaction root objects, Schema Evolution classes, Database Evolution objects, Random Update objects, Sequential Update classes</td>
<td>Uniform</td>
</tr>
<tr>
<td>( \text{PRND}, \text{PSCAN}, \text{PRANGE}, \text{PSIMPLE}, \text{PHIER}, \text{PSTOCH}, \text{PCINSERT}, \text{PCDEL}, \text{POINSERT}, \text{PODEL}, \text{PRNDUP}, \text{PSEQUP} )</td>
<td>Occurrence probability: Random Access, Simple Scan, Range Lookup, Set Access, Simple Traversal, Hierarchy Traversal, Stochastic Traversal, Class Insertion, Class Deletion, Object Insertion, Object Deletion, Random Update, Sequential Update</td>
<td>0.1, 0.05, 0.05, 0.2, 0.2, 0.2, 0.1, 0.005, 0.005, 0.02, 0.025, 0.025</td>
</tr>
<tr>
<td>( \text{COLDN} )</td>
<td>Number of transactions executed during the cold run</td>
<td>1,000</td>
</tr>
<tr>
<td>( \text{HOTN} )</td>
<td>Number of transactions executed during the warm run</td>
<td>10,000</td>
</tr>
<tr>
<td>( \text{THINK} )</td>
<td>Average latency time between two transactions</td>
<td>0</td>
</tr>
<tr>
<td>( \text{CLIENTN} )</td>
<td>Number of clients</td>
<td>1</td>
</tr>
<tr>
<td>( \text{RSEED} )</td>
<td>Random generator seed</td>
<td>Default seed</td>
</tr>
</tbody>
</table>
server environment, times must be measured on the client side with standard system primitives like time() or getrusage() in C++. The replication of the transactions compensates for the possible inaccuracy of these functions. If the number of transactions is sufficiently large, the absence of such system functions can be compensated by a manual timing, as it is specified for OO1:

- number of accessed objects (both globally and per transaction type). The computation of these usage statistics must be included in the benchmark’s code;
- number of Input/Output (I/Os) performed. The I/Os necessary to execute the transactions and the I/Os needed to cluster the database (clustering overhead) must be distinguished. I/O usage can be obtained through the C++ getrusage() function or by statistics provided by the DBMS. For instance, O₂ provides such statistics.

Comparison of OCB to the Existing Benchmarks

Genericity of OCB

Since we intend to provide a generic benchmark, our tool must be able to model various types of databases and applications. In addition, it must also be able to imitate the demeanor of previous object-oriented benchmarks. Schreiber (1994) claims Justitia bestows this property provided the benchmark is properly parameterized. However, he does not provide any solid evidence to back up his claim.

We have shown that the OCB database is generic by comparing it to the object bases from existing benchmarks (Tables 3 and 4). In terms of workload, however, the demonstration of genericity is more difficult to achieve. OO7 especially offers a wide range of complex transactions. Some of them have been discarded when designing OCB, because they added complexity without providing much insight. Still, the transactional behavior of OO1, HyperModel, and Justitia can easily be imitated. Furthermore, some of OCB’s operations, if combined, can be equivalent to OO7’s complex operations.

Table 3: OCB tuning to imitate OO1 and HyperModel object bases

<table>
<thead>
<tr>
<th>OCB parameter</th>
<th>OO1</th>
<th>HyperModel</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>MAXNREF (i)</td>
<td>Parts: 3</td>
<td>5 (Parent/Children)</td>
</tr>
<tr>
<td></td>
<td>Connections: 2</td>
<td>+ 5 (PartOf/Part)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ NO (RefTo/RefFrom)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+ 1 (Specialization)</td>
</tr>
<tr>
<td>BASESIZE (i)</td>
<td>Parts: 50 bytes</td>
<td>Node: 20 bytes</td>
</tr>
<tr>
<td></td>
<td>Connections: 50 bytes</td>
<td>TextNode: 1000 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FormNode: 20008 bytes</td>
</tr>
<tr>
<td>NO</td>
<td>20000 parts + 60000 connections</td>
<td>3906 Nodes + 125 FormNodes + 15500 TextNodes</td>
</tr>
<tr>
<td>NREFT</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>CREFLOC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>OREFLOC</td>
<td>RefZone</td>
<td>Level k + 1 in the Parent/Children hierarchy</td>
</tr>
<tr>
<td>DIST1</td>
<td>Constant (non-random)</td>
<td>Constant (non-random)</td>
</tr>
<tr>
<td>DIST2</td>
<td>Constant (non-random)</td>
<td>Constant (non-random)</td>
</tr>
<tr>
<td>DIST3</td>
<td>Constant (non-random)</td>
<td>Constant (non-random)</td>
</tr>
<tr>
<td>DIST4</td>
<td>Uniform</td>
<td>Uniform</td>
</tr>
</tbody>
</table>
Table 4: OCB tuning to imitate OO7 and Justitia object bases

<table>
<thead>
<tr>
<th>OCB parameter</th>
<th>OO7</th>
<th>Justitia</th>
</tr>
</thead>
<tbody>
<tr>
<td>NC</td>
<td>10</td>
<td>6</td>
</tr>
</tbody>
</table>
| MAXNREF (i) | Design object: 0
Atomic part: 20
Connection: 18
Composite part: \( \text{NumAtomicPerComp} + 8 \)
Document: 1
Manual: 1
Assembly: 2
Complex assembly: \( \text{NumAssmPerAssm} + 2 \)
Base assembly: \( \text{NumComPerAssm} \times 2 + 1 \)
Module: \( \sum_{i=0}^{\text{NumAssmLevels}} \text{NumAssmPerAssm}^i \) | Database Entry: 0
Node: 2
CO: 3
PO: \( \text{PO_ATT_SIZE} + 3 \) |
| BASESIZE (i) | Design object: 18 bytes
Atomic part: 12 bytes
Connection: 14 bytes
Composite part: 0
Document: \( \text{DocumentSize} + 44 \) bytes
Manual: \( \text{ManualSize} + 48 \) bytes
Assembly: 0
Complex assembly: 0
Base assembly: 0
Module: 0 | Database entry: 4 bytes
PO: 0
Node: 4 bytes
CO: 0
DO: \( \text{DO_ATT_SIZE} \) bytes
SO: \( \text{SO_ATT_SIZE} \) bytes |
| NO | \( \text{NumModules} \text{ modules} + \sum_{i=0}^{\text{NumAssmLevels}-1} \text{NumAssmPerAssm}^i \text{ complex assemblies} + \sum_{i=0}^{\text{NumAssmLevels}} \text{NumPerAssm}^i \text{ base assemblies} + \text{NumCompPerModule} \text{ composite parts} + \text{NumCompPerModule} \text{ documents} + \text{NumAtomicPerComp} \times \text{NumCompPerModule} \text{ atomic parts} + \text{NumAtomicPerComp} \times \text{NumCompPerModule} \times \text{NumCompPerModule} \text{ complex connections} \) | SECTION . MAXWIDTH . MAXLEVEL |
| NREFT | 12 | 3 |
| CREFLOC | NC | NC |
| OREFLOC | NO | NO |
| DIST1 | Constant (non-random) | Constant (non-random) |
| DIST2 | Constant (non-random) | Constant (non-random) |
| DIST3 | Constant (non-random) | Constant (non-random) |
| DIST4 | Constant + Uniform | Constant (non-random) |

Comparison with Gray’s Criteria

Gray (1993) defines four primary criteria concerning the specification of a good benchmark:
1) **relevance**: it must concern aspects of performance that appeal to the largest number of potential users;
2) **portability**: it must be reusable to test the performances of different OODBs;
3) **simplicity**: it must be feasible and must not require too many resources;
4) **scalability**: it must be able to be adapted to small or large computer systems, or new architectures.
When designing OCB, we mainly intended to palliate two shortcomings in existing benchmarks: their lack of genericity and their inability to properly evaluate the performances of object-clustering techniques. To achieve this goal, we designed a fully tunable benchmark, allowing it either to be generic or to be specialized for a given purpose. The consequences of this choice on Gray’s criteria are the following:

- **relevance**: as previously stated, all the transactions from existing benchmarks have been included in OCB except the most intricate operations from OO7;
- **portability**: OCB has been used to evaluate the performances of the O₂ and the Texas systems. Both these implementations have been made in C++. OCB has also been included in simulation models written in QNAP2 and a simulation package called DESP-C++. OCB’s code is short and simple in all these cases;
- **simplicity**: complete specifications for our benchmark are provided in this section in order to support understandability and ease of implementation;
- **scalability**: OCB is a very flexible benchmark due to an extensive set of parameters. Its object base can take different sizes and complexity levels and its various transactions can model a fair number of applications.

The characteristics of the existing benchmarks and OCB according to these criteria are summarized in Table 5.

### VALIDATION EXPERIMENTS

We present in this section performance evaluations performed with OCB on the O₂ OODB, the Texas persistent object store, and the DSTC and DRO clustering techniques, which are implemented in Texas. Our research objective did not include a comparison of the performances of O₂ and Texas. This would have been troublesome since our versions of these systems did not run on the same platform. Furthermore, O₂ and Texas are quite different in their philosophy and functionalities. O₂ is a full OODB-supporting concurrent and secure accesses, while Texas is positioned as an efficient persistent store for C++. We only intended to show that OCB provided valid performance evaluations.

Since we recommended the use of a complex object base, the feasibility of our specifications has been checked by measuring the database average generation time function of the database size (number of classes and number of instances). For schemas containing 10, 20, and 50 classes, the number of instances $NO$ was varied from 5,000 to 50,000. The actual database size was also measured for all these configurations.

Next, the object base configuration was varied: number of classes $NC$, number of instances $NO$, number of inter-object references $MAXNREF$. Four database configurations were obtained using $NC$ values of 20 and 50, and $MAXNREF$ values of 5 and 10. Then, the

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Relevance</th>
<th>Portability</th>
<th>Simplicity</th>
<th>Scalability</th>
</tr>
</thead>
<tbody>
<tr>
<td>OO1</td>
<td>--</td>
<td>++</td>
<td>++</td>
<td>--</td>
</tr>
<tr>
<td>HyperModel</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>--</td>
</tr>
<tr>
<td>OO7</td>
<td>++</td>
<td>+</td>
<td>-</td>
<td>--</td>
</tr>
<tr>
<td>Justitia</td>
<td>--</td>
<td>--</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>OCB</td>
<td>++</td>
<td>+</td>
<td>+</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 5: Comparison of existing benchmarks to OCB

*Strong point : +  Very strong point : ++  Weak point : –  Very weak point : --*
number of instances in the database was varied from 500 to 20,000 for each configuration. The scope of this study is limited to raw performance results, i.e., the average response time and the average number of I/Os necessary to execute the operations.

The efficiency of the DSTC and DRO clustering techniques has been compared by measuring the performances achieved by Texas before and after object clustering. To achieve the comparison, we used a mid-sized OCB database composed of 50 classes and 100,000 objects, for a size of about 62 MB. The other OCB parameters defining the database were set to default. A series of 10,000 standard OCB transactions was executed on this database, before and after object clustering. System performance was measured in terms of I/Os, response time, and relative performance improvement due to clustering. Only the results concerning I/Os are presented here because response time plots present exactly the same tendencies and do not bring additional insight. Eventually, these experiments have been performed in several memory configurations. Since Texas makes an intensive use of virtual memory, it was interesting to see how the system behaved when the ratio main memory size / database size varied. The whole process was reiterated 100 times so that mean tendencies could be achieved. In each iteration, the same random seed was selected for the DSTC and DRO experiments so that they were rigorously identical.

**Results for O₂**

**Material Conditions**

The O₂ server (version 5.0) was installed on an IBM RISC 6000 43P240 biprocessor workstation. Each processor was a Power PC 604e 166. The workstation had 1 GB ECC RAM. The operating system was AIX version 4. The O₂ server cache size was 16 MB by default.

**Object Base Generation**

Figure 6 displays the database generation time function of the number of classes and the number of instances in the base. It shows that generation time increased linearly when the schema was made of 10 or 20 classes. The increase was more accentuated with 50 classes, because when the O₂ client cache was full, which happened with the biggest databases, an execution error occurred. To fix this problem, the generation process has been marked out with commits. These multiple commits were more costly than a single validation at the end of the generation process. The feasibility of OCB was also demonstrated, since in the worst case generation time was less than one hour. Moreover, a given object base could be saved and reused multiple times so that the generation process could be avoided each time.

Figure 7 shows how the size of the randomly generated database linearly evolved with the number of classes and instances. Hence, it was easy to foresee the final size of a database when setting the NC and NO parameters. The default OCB database (50 classes, 20,000 instances) had a mean size of 30 MB, which is average for a benchmark. For instance, the large database in OO1 has a size of 40 MB. However, we showed that larger databases are possible.

**Object Base Usage**

In Figure 8, we plotted the mean number of I/Os globally necessary to execute the transactions function of the number of instances in the object base (NO) for our four database configurations. We did the same in Figure 9 for the mean response time.
**Figure 6: Database generation time ($O_2$)**

![Database generation time graph](image)

**Figure 7: Actual database size ($O_2$)**

![Database size graph](image)

**Figure 8: Mean number of I/Os ($O_2$)**

![Mean number of I/Os graph](image)
We can see that the performances of $O_2$ logically decreased in the three following cases.

- **NC** increase—This was due to the structure of the OCB schema. The more classes it contained, the deeper the inheritance graph was. Since information is inherited at each level from the upper level, leaf classes in the inheritance graph have bigger instances than root classes. Hence, a higher number of classes induced bigger object sizes, so the database occupied more disk pages.

- **MAXNREF** increase—The number of objects accessed by transactions that browsed all the references increased.

- **NO** increase—The database got bigger and objects were distributed over more disk pages.

The evolution of our two performance criteria was quite similar. This result was expected, since most treatments performed by the system when running OCB deal with loading objects from disk.

**Results for Texas**

**Material Conditions**

Texas version 0.5 was installed on a PC Pentium-II 266 with a 64 MB SDRAM. The host operating system was Linux 2.0.30. The swap partition size was 64 MB. Texas has been compiled with the GNU C++ compiler version 2.7.2.1.

**Object Base Generation**

Figure 10 displays the average time for database generation function of the number of instances and the number of classes in the database. It shows that generation time did not increase linearly. However, the longest generation times were approximately 10 minutes long, which was an acceptable rate.

Texas did not appear to have the same behavior as $O_2$ because average generation time was greater when the schema contained few classes. This result can be attributed to two phenomena:

- The graph consistency check for acyclic graphs was more complex when the number of classes was low. In these conditions, the interclass references were dispersed in a reduced class interval and formed very dense graphs; and.
• When the database did not fit wholly into the main memory, the system swapped, which was costly both in terms of I/Os and time. The actual size of the object bases generated with Texas was always less than 60 MB, as shown in Figure 12, allowing them to be stored in the 64 MB memory. Hence, the graph consistency check was prevalent, while in the case of O₂, swap was prevalent. This hypothesis has been checked with Texas by reducing the available memory under Linux to 16 MB.

Figure 11 displays the results of these tests, which confirmed our assumption.

**Object Base Usage**

In Figure 13, we plotted the mean number of I/Os globally necessary to execute the transactions function of the number of instances in the object base (NO) for our four database configurations. We did the same in Figure 14 for the mean response time.

In the case of Texas, the correlation between the mean number of I/Os and the mean

**Figure 10: Database generation time (Texas)**

![Figure 10: Database generation time (Texas)](image)

**Figure 11: DB generation time with 16 MB memory**

![Figure 11: DB generation time with 16 MB memory](image)
Figure 12: Actual database size (Texas)

Figure 13: Mean number of I/Os (Texas)

Figure 14: Mean response time (Texas)
response time appeared tighter than for O₂. O₂ indeed includes many more features than Texas (security, concurrency control, and others) that add an overhead that is not directly linked to disk accesses.

**Clustering Effect Evaluation**

**Results for DSTC**

Figures 15 and 16 show that clustering with DSTC indeed allows a significant gain in performance, especially when the amount of main memory available is small. Clustering is definitely more useful when the database does not fit wholly within the main memory, since its effects are felt as soon as the system swaps and not only at page load time. This assumption is neatly confirmed by the clustering gain factor graph in Figure 16. Clustering gain factor is equal to the number of I/Os necessary to execute the transactions after clustering divided by the number of I/Os necessary to execute the transactions before clustering.

**Results for DRO**

Figures 17 and 18 show that DRO bears the same overall behavior as DSTC. However,
the gain factor achieved with DRO looks much better. It is indeed about 15. The comparison is unfair, however, because we selected the optimal set of parameters for DRO clustering, while we could not do it for DSTC. Due to technical problems with big databases, we had to parameterize DSTC so that clustering was not the best possible. There was a threshold effect on a set of DSTC parameters. Below this “threshold,” everything worked out fine but clustering was average. Beyond the “threshold,” clustering units were too big for Texas to manage and the system invariably crashed.

Comparison of DSTC and DRO

To eventually compare DSTC and DRO on a fair ground, we used a smaller database so that DSTC could work properly. We used OCB’s default database (50 classes, 20,000 instances, about 20 MB) and ran two series of typical transactions that were likely to benefit from clustering: depth-3 hierarchy traversals (that always follow the same type of reference) and depth-2 simple traversals (depth-first traversals). The depth of traversals was reduced regarding OCB’s default parameters so that the generated clusters were not too big and the effects of clustering were clear. The traversals have been performed from 100 predefined root objects and each of them was executed 10 times.

Table 6 displays the mean number of I/Os concerning database usage before and after clustering. It shows that DSTC and DRO both achieve a substantial increase in performance.
(factor 6-7). DRO looks even better, though more tuning with DSTC should bring this method on the same level. Unfortunately, such tuning still provoked execution errors in Texas. The big difference between DSTC and DRO lies in clustering overhead (the number of I/Os necessary for an algorithm to cluster the database). DSTC induces a high overhead that renders it difficult to implement truly dynamically. Its authors actually advocate its triggering when the database is idle. On the contrary, DRO, which is much simpler, presents a lower overhead (about four times lower) and is certainly better suited to a dynamic execution.

**FUTURE TRENDS**

Future research about OODB benchmarking could take at least three directions. First, directly using a benchmark such as OCB with its default parameters for performance analysis of clustering and pre-fetching algorithms may produce misleading results, since such optimization methods are particularly sensitive to the object base traversal order, both within and between transactions. Indeed, there often exists dependencies between successive transactions. To handle this issue, a “dynamic” extension to the OCB benchmark called DOCB has recently been designed (He & Darmont, 2001). It features various inter-traversal dependencies and changes in access pattern.

Second, we only exposed the principles of a multi-user version of the OCB benchmark. The transition from the single-user version toward an operational multi-user version is not immediate and requires particular care. The aim of this evolution is to evaluate the efficiency of concurrency control and to see how systems react when faced with a more important and heterogeneous workload. Since OODBs normally operate in a concurrent environment, their performances cannot be gauged with a single-user benchmark.

Third, one very different aspect we did not consider here is the “qualitative” element that is important to take into account when selecting an OODB. Atkinson, Birnie, Jackson, and Philbrow (1992), Banerjee and Gardner (1995), Kempe, Kowarschick, Kießling, Hitzelgerger, and Dutkowski (1995) all insist on the fact that functionality is at least as important as raw performances. Hence, criteria concerning these functionalities should be worked out. Sheer performance could be viewed as one of these criteria. Concerning optimization methods, we could, for instance, evaluate if a clustering heuristic’s parameters are easy to apprehend and set up or if the algorithm is easy to use or transparent to the user.

Eventually, another point that can be considered is the adequacy of OCB to evaluate the performances of object-relational systems. OCB’s generic model can of course be implemented with an object-relational system, and most of the operations are relevant for such a system. Thus, OCB can allow the comparison of different logical or physical organizations (distribution of the objects into tables, implementation of associations by values or by pointers, distribution of tables into tablespaces, index…). OCB can be considered as a

<table>
<thead>
<tr>
<th>Table 6: Clustering efficiency comparison between DSTC and DRO (I/Os)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Pre-clustering usage</td>
</tr>
<tr>
<td>Post-clustering usage</td>
</tr>
<tr>
<td>Clustering gain factor</td>
</tr>
<tr>
<td>Clustering overhead</td>
</tr>
</tbody>
</table>
candidate benchmark for this type of systems, even if extensions are needed to take into account additional aspects, e.g., Abstract Data Types, in particular.

**CONCLUSION**

The full specifications for the OCB object-oriented benchmark have been presented. OCB’s main qualities are its richness, its flexibility, and its compactness. This benchmark indeed offers an object base whose complexity has never been achieved before in object-oriented benchmarks. Furthermore, since this database (and likewise the transactions running on it) are wholly tunable through a collection of comprehensive but easily set parameters, OCB can be used to model many kinds of object-oriented database applications. Eventually, OCB’s code is short, reasonably easy to implement, and easily portable.

We have shown this benchmark was merely feasible by measuring generation time for its random database. It appears that, in the worst case, an OCB object base is generated in less than one hour, which is quite acceptable. Furthermore, the largest databases can be saved for multiple uses.

We have also illustrated the genericity of the benchmark by showing how it could imitate both the schema and the operations of four existing benchmarks. The flaws identified in these previous benchmarks have been underlined, and an attempt was made to correct them. We eventually demonstrated that OCB could be used as a general purpose benchmark by evaluating the performances of the O, OODB and the Texas persistent object store. We also showed it could serve as a more specialized benchmark by comparing the effects of the DSTC and DRO clustering methods on the performances of Texas.

**ENDNOTES**

1 ftp://ftp.cs.wisc.edu/OO7
2 http://eric.univ-lyon2.fr/~jdarmont, Software section

**REFERENCES**


Chapter IV

FOOM—Functional and Object-Oriented Methodology for Analysis and Design of Information Systems

Peretz Shoval and Judith Kabeli
Ben-Gurion University, Israel

FOOM is an integrated methodology for analysis and design of information systems, which combines the two essential software-engineering paradigms: the functional- (or process-) oriented approach and the object-oriented (OO) approach. In FOOM, system analysis includes both functional and data modeling activities, thereby producing both a functional model and a data model. These activities can be performed either by starting with functional analysis and continuing with data modeling, or vice versa. FOOM products of the analysis phase include: a) a hierarchy of OO-DFDs (object-oriented data flow diagrams), and b) an initial object schema, which can be created directly from the user requirements specification or from an entity-relationship diagram (ERD) that is mapped to that object schema. System design is performed according to the OO approach. The products of the design phase include: a) a complete object schema, consisting of the classes and their relationships, attributes, and method interfaces; b) object classes for the menus, forms and reports; and c) a behavior schema, which consists of detailed descriptions of the methods and the application transactions, expressed in pseudo-code and message diagrams. The seamless transition from analysis to design is attributed to ADISSA methodology, which facilitates the design of the menus, forms and reports classes, and the system behavior schema, from DFDs and the application transactions.
INTRODUCTION

The Functional Approach and ADISSA Methodology

Many paradigms for system analysis and design have been proposed over the years. Early approaches have advocated the functional (or process) approach. Common methodologies that support this approach are Structured System Analysis (SSA) and Structured System Design (SSD) (DeMarco, 1978; Yourdon & Constantine, 1979). SSA is based on the use of data flow diagrams (DFD), which define the functions to be performed by the system, the data stores within the system, the external entities (usually user-entities, but sometimes also other types, e.g., time-entities), and the data flows among the above components. Early SSA and similar methodologies emphasized the functional aspects of system analysis, neglecting somehow the structural aspect of data model. This was remedied by enhancing those methodologies with a data model, usually the entity-relationship (ER) model (Chen, 1976), that is used to create a diagram of the data model, according to which the database schema of the application is designed.

SSD is based on the use of Structure Charts (SC), which describe the division of the system to program modules as well as the hierarchy of the different modules and their interfaces. Certain techniques have been proposed to create SCs from DFDs (see Yourdon & Constantine, 1979). The main difficulty of an approach where functional analysis is followed by structured design lies in the transition from DFDs to SCs. The translation is problematic because a DFD is a network structure, whereas a SC is a hierarchical structure. In spite of various guidelines and rules for conversion from one structure to the other, the problem has not been resolved by those methodologies (Coad & Yourdon, 1990).

Shoval (1988, 1991) developed the ADISSA methodology that solved this problem. It uses hierarchical DFDs during the analysis stage (similar to other functional analysis methodologies), but the design centers on **transactions design**. A transaction is a process that supports a user who performs a business function, and is triggered as a result of an event. Transactions will eventually become the application programs. The transactions are identified and derived from the DFDs. A transaction consists of elementary functions (namely, functions that are not decomposed into sub-functions) that are chained through data flows, and of data stores and external entities that are connected to those functions. Hence, a transaction consists of at least one elementary function and one external entity, which serve as its trigger. The process logic of each transaction is defined by means of structured programming techniques. Based on the hierarchical DFDs and the transactions, ADISSA methodology provides well-defined procedures to design the user-system interface (a menu-tree), the inputs and outputs (forms and reports), the database schema, and detailed transactions descriptions, which are eventually translated into application programs.

The sub-stage of interface design results in a menu-tree, that enables users to find and to fire desired transactions. The menu-tree is derived from the hierarchy of DFDs in a semi-algorithmic fashion, based on functions that are connected to user-entities. Briefly, a general function that is connected to a user-entity produces a menu, at some level, while an elementary function that is connected to a user-entity produces a menu item within the menu created from its super-function. Obviously, the hierarchy of menus is equivalent to the hierarchy of DFDs. Certain rules that take into account human-engineering factors enable us to modify the initial menu-tree (which is generated algorithmically). (More details can be found in the ADISSA references, and in Shoval, 1990.)
The sub-stage of the inputs/outputs design produces the forms and reports of each transaction, and is based on data flows from user-entities to elementary functions and from elementary functions to user-entities. Generally, a data flow from a user-entity to an elementary function produces an input screen (form), while a data flow from an elementary function to a user-entity produces an output screen or report. Sometimes, input and output screens that belong to the same transaction are combined into one interactive input-output screen.

In the sub-stage of database design, a relational database schema is created. This is based on the DFD data stores, where each data store is usually decomposed into one or more relations, based on analysis of dependencies among the data elements within the data stores, and the application of normalization rules. Alternatively, based on those data element dependencies, it is possible to create an ER diagram, and then to map it to normalized relations. In addition, the data flows from elementary functions to data stores and from data-stores to elementary functions serve as a basis for defining access-steps. An access-step (which is part of a transaction) is an update or a retrieval operation on a relation. Eventually, access-steps are translated into SQL statements that are embedded in the program code of the respective transaction.

The products of the design stages can be easily implemented using various programming environments.

The Object-Oriented Approach

The development of object-oriented (OO) programming languages gave rise to the OO approach. The OO approach maintains that, in order to develop information systems in such languages, it is recommended to perform object-oriented analysis and design. Many OO methodologies have been developed (e.g. Booch, 1991; Coad & Yourdon, 1990; Coad & Yourdon, 1991; Jacobson, 1992; Martin & Odell, 1992; Rumbaugh, Blaha, Premerlani, Eddy & Lorensen, 1991; Shlaer & Mellor, 1988; Shlaer & Mellor, 1992; Wirfs-Brock, Wilkerson, & Wiener, 1990). In the OO approach the world is composed of objects with attributes (defining its state) and behavior (“methods”), which constitute the only way by which the data included in the object can be accessed. When using the OO approach, a model of the system is usually created at the analysis stage in terms of objects—an object schema (or OO-schema, or class diagram). An OO-schema consists of different object classes with various structural relationships between them (e.g., generalization-specialization), and each object class having its attributes and behavior (functionality). During the design stage, implementation considerations are added and the result is a model that is supposed to enable OO programming.

Many advocates of the OO approach claim (with no substantial proof, however) that it is more natural to begin the analysis of a system by defining its object structure rather than by defining its functions. They support this with the claim that the real world is not composed of functions but rather of objects. They maintain that the OO approach simplifies the transitions in system development stages, enhances communication between users and developers, encourages code reuse and enables the creation of robust information systems that can be easily upgraded and maintained.

While there are no doubts about the advantages of the OO approach in programming, as it supports information hiding (encapsulation), software reuse and maintenance, there are doubts with respect to the effectiveness of the approach for analyzing business-oriented information systems (as opposed to real-time systems). OO methodologies tend to neglect
the functionality aspect of system analysis, and do not show clearly and systematically how to integrate the application functions (transactions) with the object schema. This is clearly a drawback since users usually express their information needs in terms of functions that the system ought to perform. Another difficulty with many OO methodologies is that they involve many types of diagrams and notations with no specific guidelines about which to use and in what order.

The multiplicity of diagram types in the OO approach has been a major motivation for developing the Unified Modeling Language (UML) (see, for example, Booch et al., 1999; Clee & Tepfenhart, 1997; Fowler, 1997; Larman, 1998; UML Rose, 1998; Maciaszek, 2001). UML was developed in order to produce a standard (“unified”) modeling language. It consists of several types of diagrams with well-defined semantics and syntax, which enable a system to be presented from different points of views. But UML does not offer a development methodology that guides the developer on how to do or what development process to follow. Moreover, some of the diagram types are redundant, thus it is not always clear if and when they should be used in system development (e.g., sequence diagrams vs. collaboration diagrams vs. activity diagrams, or class diagrams vs. object diagrams).

Nowadays, the practice is to either follows a “pure” object-oriented or a “pure” functional-oriented approach. However, our approach is to integrate the two paradigms. In our view, functions are as fundamental as objects, and complement each other. Hence, we propose a system analysis and design methodology that combines the two approaches.

**RESEARCH MOTIVATION AND GOALS**

Information systems development is a multi-phase process in which the analysis and design are of primary importance. Therefore, it is vital to examine which approaches and methods are appropriate to perform each of these phases. On the one hand, those who adopt the OO approach claim that using data abstraction at the analysis phase, producing a model of reality by means of classes, is preferable to producing a functional model because the real world consists of objects. However, as far as we know, no such study has shown that the OO approach is more effective than the functional/data approach in the development of business-oriented information systems. OO methodologies tend to neglect the functionality aspect of system analysis, and do not clearly show how to integrate the systems functions, or transactions, with the object schema. One sometimes gets the impression that the functionality of the system is expressed by means of methods that are encapsulated within objects, thus disregarding functional requirements that cannot be met by simple methods. In contrast, based on vast experience in performing functional analyses with DFDs, we have met with no problems as a means to express the functionality of the system; the only problem was how to continue from there to the following phases of development.

In our opinion, since process and object are both fundamental building blocks of reality, the analysis phase must cover both the functional and the data aspects. The functional approach, using DFDs, is suitable for describing the functionality of the system, while ERD or OO-schemas are suitable for modeling the data structure. Since the OO approach (as will be discussed later) is the one most appropriate for performing the design phase, it seems more effective to produce an initial OO-schema already at the analysis phase, and then to use it as input to the design phase. (The term initial OO-schema will be clarified later on.) We suggest performing data modeling by creating an initial OO-schema, or by first creating an ERD, and then translating it (quite automatically)
to an initial OO-schema. This method was shown to be easy and simple to implement and preferable by designers (Shoval & Shiran, 1997).

For the design phase, it is crucial to provide a smooth and seamless transition to system implementation. Since there is an agreement on the advantages of OO programming, it is also desirable to design the system with methods and tools that belong to the OO family. Therefore, we conclude that, in order to perform each of those development phases with its most appropriate method, there is a need to integrate the functional- and object-oriented approaches.

Dori’s Object-Process Methodology (OPM) (Dori, 1996; Dori, 2001) indeed integrates the two approaches. It utilizes a single graphic tool, Object-Process Diagram (OPD), at all development phases. However, since OPD defines a new notation that combines DFD and OO diagrams, it includes a great many symbols and rules. It seems to us that such diagrams are not so easy to construct and comprehend for large-scale systems, and that reality has to be modeled by means of simple notations that are easy to learn, comprehend and utilize. A single hybrid notation, like OPM, must be very rich in order to elicit all those points of view, thus leading to a complex, perhaps distorted model of reality. On the other hand, multiplicity of models and corresponding diagramming tools, as found in UML, may also be too complicated. Too many diagram types can hamper coherent understanding and lead to the production of erroneous models and systems.

We are looking for an optimal way to integrate the process and object approaches. Since users express their information needs in a functional and data manner, and not by means of an object structure and behavior, an appropriate (natural) method to carry out the analysis task is by functional/data analysis. On the other hand, the design should be made through the OO approach to facilitate the transition of the design to OO programming, which has proved itself to be a better approach to implement software. The integration of the two approaches is made possible because it applies principles and techniques taken from the ADISSA methodology, especially transactions design. This enables the transition from functional analysis DFDs to an OO model that consists of object and behavior schemas.

**OUTLINE OF FOOM METHODOLOGY**

The purpose of this paper is to present FOOM, an integrated methodology for the development of business-oriented information systems, which combines the functional and the OO approaches. The functional approach is used at the analysis stage (where its advantage lies) to define user needs, which is done by the creation of a hierarchical OO-DFD model, and an ERD (in some cases) as an interim product to produce an initial OO-schema. The OO approach is used at the design stage (where its advantage lies) to define the structure and behavior of the system, which is done by creation of full object and behavior schemas. The essential steps of the methodology are as follows.

**The Analysis Phase**

The analysis phase consists of two main activities: functional modeling and data modeling. They can be performed in any order, depending on factors that will be examined in further research. (Intuitively, we believe that it is better to begin with data modeling followed by functional modeling.) The products of this stage are a data model, in the form of an initial OO-schema, and a functional model, in the form of hierarchical OO-DFDs (supported by a data-dictionary).
The initial OO-schema consists of “data” classes (also termed “entity” classes), namely, classes that are derived from the application requirements and contain “real world” data. (Other classes will be added at the design stage.) Each class includes attributes of various types (e.g., atomic, multi-valued and tuples of attributes, keys, sets, and reference attributes). Association types between classes include “regular” (namely, 1:1, 1:N and M:N) relationships, with proper cardinalities, generalization-specialization (is-a, or inheritance) links between super- and subclasses, and aggregation-participation (is-part-of) links. (Note that in our model, relationships are signified not only by links between respective classes, but also as reference attributes to those classes.) The initial OO-schema does not include methods; these will be added at the design phase. An example of the OO-schema is shown in Figure 1. (Disregard the method names at the lower compartments of the classes; they do not belong to the initial OO-schema.)

The OO-DFDs model specifies the functional requirements of the system. Each OO-DFD consists of general or elementary functions, external entities—mostly user-entities, but also time and real-time entities, object-classes (instead of the “traditional” data-stores), and the data flows among them. Examples are shown in Figures 2-4. (The examples will be explained in a later section.)

When the performing order starts from functional modeling, the analyst analyzes the user requirements and based on that creates first “traditional DFDs,” which include (beside other components) data-stores. Then he/she can create an ERD from those data-stores and map the ERD to an initial OO-schema, and finally, upgrade the initial DFDs to OO-DFDs by replacing each data-store with a respective object class appearing in that OO-schema. Alternatively, the analyst can create OO-DFDs from the beginning (namely, based user requirements), by introducing classes, rather than data-stores, in the DFDs. Based on that, the analyst can continue by creating an initial OO-schema—based on the classes already appearing in the OO-DFDs. This means mainly defining proper class associations and attributes. The above performing order may be problematic because it may not be easy to define classes in the course of constructing the OO-DFDs, and many changes may be required in the OO-DFDs after the initial OO-schema is constructed.

So, the alternative performing order may be more effective; namely, it first creates the initial OO-schema (based on user requirements), then, since the classes are already defined, it creates the OO-DFDs. Creation of the initial OO-schema can also be done in two alternative ways: one alternative is to start by constructing an ERD, and then map it to an initial OO-schema. The other is to directly construct an initial OO-schema (not from an ERD). While the second alternative seems to be more direct, some analysts may prefer the “indirect” way, because it has been shown that production of an ERD (and mapping it to OO-schema, by following well-known mapping rules) provides more correct schemas (see, for example, Shoval and Shiran, 1997).

The alternative performing orders of the analysis activities are summarized in Table 1. We plan to investigate the pros and cons of the alternatives in experimental settings, hoping to get proof of a recommended order. At any rate, the analysis activities must not be carried out in sequence, as could perhaps be understood from the above discussion. Rather, they should be performed in parallel or in iterations. For example, the analyst can start by doing part of one activity (create a partial class diagram), continue with the other activity (produce a few OO-DFDs that relate to those classes), return to the first activity (update and extend the class diagram), and so on — until the analyst feels that the two products are complete, consistent and satisfy the user’s requirements.
Figure 1: The initial OO-schema

```
Index:
{}  Tuple of attributes
[]  Reference attribute

Listener Request
  ↩ ListenerId
  RequestedProgram [Music Program]
  RequestedPiece [Musical Piece]
  RequiredTimeInterval {from-date, to-date}
  InscriptionWords
  Status (Wait, Answered)
  ScheduledInProgram [Scheduled Program]
  Elementary Methods (Cons, Del, Set, Get)

Music Program
  ↩- ProgramName
  Season
  Duration {hours, min.}
  EditorName
  TechnicianName
  Set MusicTypes
  Set Time {Day, Hour}
  Set RequiredBy [Listener Request]

Musical Piece
  ↩ PieceId
  PieceName
  Set RightsOwner
  Set Composer
  Set Performer
  MusicType
  Length
  Set ScheduledInPrograms [Scheduled Program]
  Set BroadcastInPrograms [Scheduled Program]
  Set RequestedBy [Listener Request]
  Elementary Methods (Cons, Del, Set, Get)

Scheduled Program
  ↩- Program {BroadcastTime, [Music Program]}
  Status (Unscheduled, Scheduled)
  Set ScheduledPieces [{[Musical Piece], ScheduledTime}]
  Set BroadcastPieces [Musical Piece]
  Set ListenersAnswered [Listener Request]
  Elementary Methods (Cons, Del, Set, Get)
```
Figure 3: OO-DFD-4 schedule music program
The Design Phase

Defining Basic Methods

Basic methods of classes are defined according to the initial OO-schema. We distinguish between two types of basic methods: elementary methods and relationship/integrity methods. (More methods, for performing various users’ needs, will be added at the next stage.)
Table 1: Summary of alternative orders of performing the analysis activities

<table>
<thead>
<tr>
<th>A: Functional Model First</th>
<th>“Traditional” alternative</th>
<th>“Direct” alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1) create “traditional” DFDs</td>
<td>A1) create OO-DFDs</td>
<td></td>
</tr>
<tr>
<td>A2) create ERD</td>
<td>A2) create OO-schema</td>
<td></td>
</tr>
<tr>
<td>A3) map ERD to OO-schema</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4) change DFDs to OO-DFDs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B: Data Model First</th>
<th>“Indirect” alternative</th>
<th>“Direct” alternative</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1) create ERD</td>
<td>B1) create OO-schema</td>
<td></td>
</tr>
<tr>
<td>B2) map ERD to OO-schema</td>
<td>B2) create OO-DFDs</td>
<td></td>
</tr>
<tr>
<td>B3) create OO-DFDs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Elementary methods include: a) construct (add) object, b) delete (drop) object, c) get (find) object, and d) set (change) attributes of object. Elementary methods actually belong to a general (basic) class, from which all the classes inherit. The method “classname.const(object)” adds an object to the class; “object.del()” drops an object; “object.get(attribute)” returns the value of the object attribute; and the method “object.set(attribute,value)” changes the value of an attribute.

Relationship/integrity methods are derived from structural relationships between classes. They are intended to perform referential integrity checks, depending on the relationship types between the classes and on cardinality constraints on those relationships. For each relationship, which is also expressed in terms of reference attributes, the involved classes include appropriate integrity methods, that will fire whenever an object of a respective class is added, deleted or changed. For example, assume that Classes A and B are in (0,5): (1,1) relationship, such that an object of Class A must be connected to one object of B, and an object of Class B may be connected with up to five objects of Class A; if at run-time the application attempts to add an object of Class A, the integrity.add method of Class A will be invoked to check that the related object of Class B exists; if the application attempts to connect an object of Class B to an object of Class A, the integrity.connect method of Class B will be invoked to check that the number of connections of the A-type object does not exceed the limit of five. Generally, for each relationship between classes we can define an integrity method for operations of add, delete, connect, disconnect and reconnect. (For more details on integrity methods see Balaban & Shoval, 1999.) Figure 1 shows some of the basic and the relationship/integrity methods.
Top-Level Design of the Application Transactions

This stage is performed according to ADISSA methodology, where the application transactions are derived from DFDs (for more details see Shoval, 1988). Note that here the transactions include “read” or “write” data flows from or to classes rather than to data-stores. The products of this stage include the transactions diagrams, as extracted from the OO-DFDs, the top-level descriptions of the transactions, and a new class—“Transactions class.” This virtual class does not contain objects—just the transaction methods. (This will be elaborated in a later section.)

Figure 5 shows examples of two transaction diagrams: One (Figure 5a) is a “simple” transaction consisting of one elementary-function, one class and one user-entity; this transaction is derived from OO-DFD-0 (Figure 2). The other (Figure 5b) is a more “complex” transaction consisting of several elementary-functions, classes and user-entities; this transaction is derived from OO-DFD-4 (Figure 3).

A top-level transaction description is provided in a structured language (e.g., pseudo-code or flowchart), and it refers to all components of the transaction: every data-flow from or to an external entity is translated to an “Input from...” or “Output to...” line; every data-flow from or to a class is translated to a “Read from...” or “Write to...” line; every data flow between two functions translates to a “Move from... to...” line; and every function in the transaction translates into an “Execute function...” line. The process logic of the transaction is expressed by standard structured programming constructs (e.g., if... then... else...; do-while...). The analyst and the user, who presents the application requirements, determine the process logic of each transaction. This cannot be deducted “automatically” from the transaction diagrams alone, because a given diagram can be interpreted in different ways, and it is up to the user to determine the proper interpretation. The top-level descriptions of the above two transactions are shown in Figure 6.

The top-level transaction descriptions will be used in further stages of design, namely, input/output design and behavior design, to provide detailed descriptions of the application-specific class methods (which are in addition to the basic methods), as well as of the application programs.

Design of the Interface–The Menus Class

This stage is performed following the ADISSA methodology (Shoval, 1988, 1990). As already explained, a menu-tree interface is derived in a semi-algorithmic way from the hierarchy of OO-DFDs. An example of a menu-tree that is derived from our example OO-DFDs is shown in Figure 7. Note the correspondence of the menus and menu items to the respective general functions and elementary functions in the OO-DFDs. The menu-tree is translated into a new class—the “Menus class.” The instances (objects) of the Menus class are the individual menus, and the attribute values of each object are the menu items. Figure 8 shows the Menus class and its objects (it is quite similar to Figure 7). Note that some of the selections within

Figure 5a: Transaction 1
**Figure 5b: Transaction 4.3-4.7**

**Figure 6a: Top-level description of Transaction 1**

**Begin transaction_1**
/* Input details on Musical Pieces */
1. Input from E1: Musical Distributor – details from cassettes and CDs.
2. Execute Function 1 – Input Details of Musical Piece

**End transaction.**
Figure 6b: Top-level description of Transaction 4.3-4.7

Begin transaction 4.3-4.7

/* Find the program to schedule*/
1. Input from E3: Program Editor - Program name, broadcast date and time
2. Read from Class: Music Program - Program’s details
3. Read from Class: Schedule Program - Broadcast date and time
4. Execute Function 4.3 - Input Program to Schedule
5. Move from Function 4.3 to Function 4.4
   /* Retrives information about relevant musical pieces, previous scheduled programs and broadcast pieces, and relevant listeners’ requests */
6. Read from Class: Musical Piece - Details of Pieces suitable to the program
7. Execute Function 4.4 - Present Suitable Pieces for the Program
8. Output to E3: Program Editor - Suitable pieces to schedule
9. Move from Function 4.4 to Function 4.5
10. Read from Class: Music Program - Previous scheduled, broadcast pieces
    /*Retrieve pieces that were broadcast a month before today and also retrieve pieces that will be schedule to programs before and after the present Scheduled program*/
11. Execute Function 4.5 - Present Previous Scheduled and Broadcast Programs
12. Output to E3: Program Editor - Previous scheduled and broadcast pieces in same music program
13. Move from Function 4.5 to Function 4.6
14. Read from Class: Listener’s Request - Appropriate listeners’ requests
15. Execute Function 4.6 - Present Listeners’ Requests
16. Output to E3: Program Editor - Suitable listeners’ requests
17. Move from Function 4.6 to Function 4.7
18. Repeat
19. Input from E3: Program Editor - Selected piece for broadcast
20. Execute Function 4.7 - Schedule Musical Pieces for Program
21. Write to Class: Scheduled Program - Scheduled piece details
22. Repeat /*request for this piece*/
23. Input from E3: Program Editor – Selected Request for broadcast
24. Write to Class: Scheduled Program - the Listener request
25. Write to Class: Listener Request - Status = “Answered”
26. Output to E2: Listener - Message to the listener
    /* The message is that the request will be broadcast on the specified program*/
27. Until the editor finishes to select requests for the program
28. Until the editor finishes to select pieces for the program
29. Write to Class: Scheduled Program - Status = “Scheduled”

End transaction.
a given menu may call (trigger) other menu objects, signified by S (selection), while other selections may trigger transactions, signified by T. Transactions will be implemented as methods of the Transactions class (as will be detailed later). Hence, at run time, a user who interacts with the menu of the system actually works with a certain menu object. He/she may

Figure 7: Interface design–The menu tree

![Menu-0 object>
Schedule a Music Program

- 41: Input/update music programs for a season (T)
- 42: Display/print music programs for a season (T)
- 43: Schedule a program (T)
- 44: Produce report of a scheduled

Figure 8: The menus class and objects

![Menu-0 object>
The Music Programs System (main menu)

1: Input details of musical piece (T)
2: Produce catalogs of music pieces (T)
3: Input/update listeners' requests (T)
4: Schedule a music program (S)
5: Produce actual broadcasting report (T)
5.1: Produce a monthly hearings report (T)
select a menu item that will cause the presentation of another menu object, or invoke a transaction, which is a method of the Transactions class.

Figure 9 shows a possible implementation of the Menus class, and Figure 10 shows the Menus methods (“Display,” “Choose,” and “Set”), including the main program that starts the application. The main program calls method “Display” of Menu-0 (an instance of the Menus class) that displays the main menu and waits for the user’s selection. When the user selects a menu item, the method “Set” returns his/her selection. Depending on the selection and method, “Choose” displays a sub-menu or fires a respective Transactions method.

**Design of the Inputs and Outputs—The Forms and Reports Classes**

This stage is also performed according to ADISSA methodology and is based on the input and output lines appearing in each of the transaction descriptions. Hence, for each “Input from...” line, an input screen/form will be designed, and for each “Output to...” line an output screen/report will be designed. Depending on the process logic of each transaction, some or all of its input or output screens may be combined. Eventually, two new classes are added to the OO-schema: “Forms class” for the inputs, and “Reports class” for the outputs, as shown in Figure 11. Obviously, the instances (objects) of each of these class types are the input screens and output screens/reports, respectively. Such classes are usually defined in OO programming languages and can be reused. A possible implementation of an Input screen is shown in Figure 12.

**Design of the System Behavior**

In this stage, we have to convert the top-level descriptions of the transactions into detailed descriptions of the application programs and application-specific methods. A detailed description of a transaction may consist of procedures, which can be handled as follows: a certain procedure may be identified as a basic method of some class. Another procedure may be defined as a new, application-specific method, to be attached to a proper
Shoval & Kabeli

Figure 10: The interface methods

Sub Main ()
Menu-0.Display
Itemld = Menu-0.Set ( ) /*Display the main menu*/
Menu-0.Choose (Itemld) /*Return the user selection (clicking)*/
End Sub

Function Set () as integer
Set = user selection
End Function

Sub Choose (ItemId as integer)
Select Case ItemId
  Case 1: call transaction_1 ( ) /*Input details of a musical piece*/
  Case 2: call transaction_2 ( ) /* Produce catalogs of musical pieces*/
  Case 3: call transaction_3 ( ) /* Input/update a listener request*/
  Case 4: Menu-4.Display (*Display menu-4 - Schedule a music program*/
    Itemld = Menu-4.Set ( ) /* Save the user selection into variable “Itemld”*/
    Menu-4.Choose (Itemld) /*Call “Choose”*/
  End Case
  Case 41: call transaction_4.1 ( ) /*Input/update music programs for the season */
  Case 42: call transaction_4.2 ( ) /*Display/print music programs for the season */
  Case 43-47: call transaction_4.3-4.7 ( ) /*Schedule a music program*/
  Case 5: Menu-5.Display (*Display menu-5 - Update/report broadcast pieces*/
    Itemld = Menu-5.Set ( ) /* Save the user selection into variable “Itemld”*/
    Menu-5.Choose (Itemld) /* Call “Choose”*/
  End Case
  Case 51,52: call transaction_5.1-5.2 ( ) /* Input a broadcast piece*/
  Case 53: call transaction_5.3 ( ) /* Produce actual broadcasting report*/
  Case 54: call transaction_5.4 ( ) /* Produce a monthly hearings report*/
  Case 99: End /*Exit from system*/
End Select

Figure 11: The forms and reports classes

Forms Class

<table>
<thead>
<tr>
<th>ID</th>
<th>InputSource</th>
<th>InputType</th>
<th>InputLen</th>
<th>Description</th>
<th>Permissions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Object = Display( )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Some instances of Forms class:
1. Input a musical piece
2. Input a listener’s request
3. Input details on a music program

Reports Class

<table>
<thead>
<tr>
<th>ID</th>
<th>OutputMedia</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Permissions</td>
<td></td>
</tr>
</tbody>
</table>

Some instances of Reports class:
1. Output suitable musical pieces for a program
2. Output suitable listeners’ requests for a program

class. Remaining procedures (which are not identified as basic methods or defined as specific methods) will become a Transactions method, which is actually the “main” part of the transaction’s program. Hence, every transaction is represented as a Transactions method, which, once triggered by the user via proper menus selections, may call
We can categorize the application transactions according to their complexity – depending on how many classes and methods they refer to. For example, a **simple transaction** (e.g., one that finds a certain object and displays its state, or that updates attributes of an object) may be implemented as a small procedure (namely, a Transactions method) that simply sends a message to a basic method of a certain class. A **complex transaction** (e.g., one that finds and displays objects that belong to different classes, or that updates objects that belong to different classes or that both retrieve and update various objects) may be implemented as a more complex procedure that sends messages to basic methods and specific methods of various classes. Generally, an application may consist of many transactions with different levels of complexity. At any rate, every transaction is “represented” in the Transactions class as a Transactions method, which contains the “main” procedures of the transaction, which, once triggered, may trigger other methods of other classes, according to the process logic of the transaction. Note that at run-time, when a user wants to “run” any transaction, he/she actually approaches the Menus class and makes proper item selections within the menu objects, until a menu item that actually fires the desired Transactions method is selected. From here, the execution of a transaction depends on the process logic of the Transactions method and the other methods it calls.

The detailed description of a transaction is expressed in two complementing forms: **Pseudo-code and Message Diagram**. A pseudo-code (see examples in Figure 13) is a structured description that details the process logic of the Transactions method as well as any other class method. The transition from a top-level description of a transaction to its detailed pseudo-code description is done as follows: Every “Input from...” and “Output to...” line in the top-level description is translated to a message calling an appropriate method of the
Forms/Reports class. Every “Read from...” or “Write to...” line is translated to a message calling a basic method (e.g. “Get,” “Const,” “Set,” and “Del”) of the appropriate class. Every “Execute-Function...” line is translated to messages calling one or more basic methods of certain classes, or to new, specific methods that will be attached to proper classes, or to procedures that remain as part of the Transactions method. Note that since a specific method of a class may consist of calls to basic methods and other specific methods of the class, a “Read from...” or “Write to...” line of a top level description along with an “Execute...” line may all together be translated to a specific method. A transaction description must not be too “formal”; it should include explanations and comments that will clarify the transaction to the programmer who will implement it in the proper programming language.

A Message Diagram (see example in Figure 14) shows the classes, methods and messages included in a transaction, in the order of their execution. A message diagram is actually a partial class diagram that shows only the classes involved in the transaction (including Data, Menus, Forms, Reports and Transactions classes), the method names (and parameters) included in that transaction, and message links from calling to called classes. Message diagrams supplement the pseudo-code descriptions of transactions. They must not be created, especially for simple transactions.

To summarize, the products of the design phase are: a) a complete class diagram, including Data, Menus, Forms, Reports and Transactions classes, each with various attribute types and method names (and parameters), and various associations among the classes; b) detailed menu objects of the Menus class, each menu listing its items (selections); c) detailed form and report objects of the Forms and Reports classes, each detailing its titles and data fields; d) detailed transactions descriptions in pseudo-code; and e) message diagrams of non-trivial transactions.

System Implementation With OO Programming Software

At the implementation stage, the programmers will use the above design products to create the software with any common OO programming language, such as C++ or Java. (More details on this stage are beyond the scope of this chapter.)

Figure 13a: Pseudo-code of Transaction_1

```
Transaction_1()
{
    Repeat
        Piece=Input_Piece.Display       /*1, 2*/
        /* Input_Piece is a form, which enables the user to fill in details of a musical piece. When the user completes the form it returns the filled piece object */
        If Piece != NULL then            /* if the user selects to add a piece */
            Musical_Piece.Const(Piece)   /* Construct an object of Musical Piece */
            /*3*/
        End if
    Until Piece = NULL                /* the user selects to stop adding pieces */
}
```
Figure 13b: Pseudo-code of Transaction_4.3-4.7

```plaintext
Transaction_4.3-4.7( )
{
    Repeat /*repeat until the user selects not to scheduled more programs*/
        details = Input_Program_to_Schedule.Display /*Displays a form to fill the details about the scheduled program*/
        If details != NULL then /*if details were entered*/
            MusicProgram = Music_Program.GetObject (ProgramName, "="=", details.ProgramName) /*retrieves the appropriate music program object*/
            ScheduleProgram = Scheduled_Program.GetObject (BroadcastTime, "="=", details.Time, "and", MusicProgram, "="=", details.ProgramName) /*retrieves the appropriate scheduled program object*/
            SuitablePieces = Musical_Piece.GetObject (MusicType, "in", MusicProgram.MusicTypes) /*retrieves all the musical pieces that their type in SuitableMusicTypes*/
            Output_Musical_Pieces.Display (SuitablePieces) /*Displays a report that present all suitable musical pieces*/
            PreviousPieces = MusicProgram.PreviousPieceInProg.Get () /*retrieves all pieces that have been scheduled or broadcast*/
            Output_Previous_Pieces.Display (PreviousPieces) /*Displays a report that presents these pieces*/
            SuitableRequests = Listener_Request.GetObject (RequestedProgram, "="=", details.ProgramName, "and", RequiredTimeIntervals, "="=", details.BroadcastTime) /*retrieves requests that ask this program*/
            Output_Listener_Request.Display (SuitableRequests) /*displays a report that presents these requests*/
            Repeat /*repeat until the user selects not to schedule more pieces*/
                Piece = Input_Piece.Display /*displays a form to fill musical piece details*/
                If Piece != NULL then /*if details were entered*/
                    ScheduledProgram.PieceScheduled.Con (Piece) /*connects the musical piece to the scheduled program*/
                    Repeat /*repeat until the user selects not to schedule more requests for this piece*/
                        Request = Input_Request.Display (Piece, SuitableRequests) /*Displays a form to select suitable requests for the piece that as scheduled*/
                        If Request != NULL then /*if request was entered*/
                            ScheduledProgram.RequestScheduled.Con (Request) /*connects the request to scheduled program*/
                            Request.Set ( Status, "Answered") /*set the request status to "answered"*/
                            Output_Listener_Request (Request) /*send a message to the listener*/
                        End if
                    End repeat
                End if
            End repeat
        End if
    Until details = NULL
}
```
Figure 14a: Message diagram of menus items 1 and 4.3-4.7

Figure 14b: Message diagram of Transaction_1
Figure 14c: Message diagram of Transaction_4.3-4.7
EXAMPLE: THE RADIO MUSIC PROGRAMS SYSTEM

To demonstrate the application of our methodology through the analysis and design stages, we use a small example: Radio Music Programs. It calls for the development of an information system for a radio station that specializes in broadcasting music programs. The example is small enough to be worked out in some detail, yet it contains elements that are essential to demonstrate the methodology.

The Radio Music Programs System—User Requirements

The radio station has a store of musical pieces, which is updated when cassettes and compact discs arrive from distributors. The data on all musical pieces is stored in the database. Each musical piece gets a unique ID number. Data also includes the title, the owner of the creator rights, the composer, the performer, and the music type (e.g., pop, classic). The system enables the production of various catalogs of inventory (sorted by music types, composers, performers, etc). The catalogs are distributed to the music manager, program editors and distributors.

Some of the programs are intended to broadcast musical pieces according to listeners’ requests. An interested listener sends a postcard or an email message that contains the program name, the name of the piece of music, date intervals for broadcasting the requested piece, and a small narrative describing his request. Each request is stored and waits to be treated by the editor of the required program.

An essential task of the system is to aid the program editors in preparing the programs. The music manager of the radio is responsible for planning the music programs of the season. Each program has a title, duration (in hours/minutes), day(s) of the week and hours that it will be on the fly, music types to be scheduled, editor-in-charge, technician, and broadcast date. The planned music programs for the season are distributed to the music manager, the program editors, the technicians and the listeners.

A program editor schedules the musical pieces for each of the programs for which he/she is in charge. He/she inputs the program identification to be scheduled, and the system retrieves and presents information that helps the editor with scheduling. The system enables the editor to review the database of existing musical pieces according to certain criteria (e.g., music type), see previously played pieces in past programs, and review listeners’ requests that are relevant to this program. Based on this information, the editor selects the pieces to be played. A printed message about the broadcasting time is sent to each listener whose request is answered.

The radio station pays royalties to the owners of the creator rights in exchange for each playing. Therefore, the technician in charge of a program marks, at the actual broadcasting time, the pieces actually played. Different reports of the played musical pieces, arranged according to specified parameters (e.g., time intervals, programs, creator rights owners, composers, performers, editors, etc.) are produced and distributed to the music manager and to the musicians’ association (which represents the owners of the creator rights). The financial accounts with the owners of the creator rights are made by the financial system of the radio. Therefore, at the end of each month, the system provides a summary report of the played musical pieces through the month. It summarizes the numbers of broadcasts of each musical piece, sorted by the owners of the creator rights, programs and musical pieces. The report is sent to the musicians’ association and a corresponding file is sent to the finance department of the radio station, which is responsible for payments.
The Analysis Phase

In this example we demonstrate a performing order that starts with the creation of a data model—the initial OO-schema, followed by a functional model, creating OO-DFDs. (According to Table 1, the “Direct” alternative of the “B” approach.)

Data Modeling

Figure 1 (disregarding the method names listed at the bottom compartment of each class symbol) presents the initial OO-schema, which is the result of the data analysis of the above user requirements. It consists of four Data classes: “Music Program,” “Scheduled Program,” “Musical Piece,” and “Listener Request.” “Music Program” class consists of actual music program objects—scheduled or still unscheduled. “Scheduled Program” class consists of objects that detail each actual program (broadcasted at a specific date). “Musical piece” class objects contain data on all musical pieces stored in the radio station. “Listener Request” class stores requests to hear musical pieces in scheduled programs, arriving from listeners. (The “Employee” class, which contains data about program editors and technicians, is not shown here; we assume it is part of a separate/existing information system). Each class includes attributes of various types. For example, “Musical Piece” includes a key attribute “PieceId,” atomic attributes “PieceName,” “MusicType,” and “Length,” set attributes “RightsOwner,” “Composer,” and “Performer,” and reference attributes “ScheduledInProgram,” “BroadcastInPrograms,” and “RequestedBy” that contain references to objects of the respective classes. In this example, the three reference attributes are also sets because of the 1:N relationships. Note that every reference attribute has an inverse reference attribute in the related class, for example, “RequestedPiece [Musical Piece]” in class “Listener Request.”

Functional Analysis

Figures 2, 3 and 4 show the “OO-DFDs” of the system. As can be noted, they consist of classes, instead of data-stores as in “traditional” DFDs.

Figure 2 shows the top-level OO-DFD of the system. It consists of five functions, seven external/user entities (E1-E7) that are sources of inputs or target of outputs of the system, and the above four classes. Three functions (1, 2 and 3) are “elementary” while two (4 and 5) are “general” functions, each of which is decomposed into sub-functions and is detailed in a separate OO-DFD. Function 1, “Input Details of Musical Piece,” enables us to input data about musical pieces arriving in cassettes and CDs from distributors, and store them as musical piece objects. Function 2, “Produce Musical Pieces Catalogs,” enables us to produce various catalogs according to the parameters entered by the music manager. The catalogs are distributed to the music distributors, program editors and the music manager. Function 3, “Input/Update Listener Request,” handles requests from listeners. Before saving a request, the system checks if the required music program exists and if the required musical piece is in the radio store. Function 4, “Schedule Music Program,” and Function 5, “Update/Report Broadcast Pieces,” are general functions that will be detailed in separate OO-DFDs (Figures 3 and 4).

Figure 3 is an OO-DFD that describes general Function 4. It consists of eight elementary functions. Function 4.1 inputs and updates details of the music programs for the season. From the external class “Employee,” it reads details on editors and technicians and writes to “Scheduled Program” and “Music Program” classes the details entered by the music manager. Function 4.2 displays or prints the details of the music programs to the corresponding user entities. (Note that listeners may sometimes obtain this information from newspa-
Functions 4.3-4.7 belong to one major transaction (because they are directly linked one to the other) that treats the scheduling of musical pieces. Once the program editor triggers the transaction and inputs the program’s name and broadcast date and time, the system reads details from “Music Program” and “Scheduled Program” classes and displays the relevant information to the editor. Then the system reads from the “Musical Piece” class and displays information about appropriate pieces for the program, and also reads from the “Listener’s Request” class and displays appropriate requests. Function 4.7 enables the editor to mark the pieces he/she decided to schedule and the respective listener requests. (Note that it is not necessary to complete the program scheduling at once; the program editor may fire the same transaction any time, till the task is complete.) Function 4.8, which belongs to a separate transaction, enables us to produce reports about any scheduled program for the editors and technicians (to be used during program broadcasting).

Figure 4 is an OO-DFD that describes general Function 5. It consists of four elementary functions. Functions 5.1 and 5.2 belong to one transaction that records the musical pieces actually played. The details about these pieces and the scheduled programs are read from “Musical Piece” and “Scheduled Program” classes, respectively. Function 5.3 produces reports for the music manager and the musicians’ association about the musical pieces that were actually broadcasted—according to parameters that are entered by the music manager. It retrieves the relevant data from the “Music Piece” and “Scheduled Program” classes. Function 5.4 produces periodic reports about the musical pieces that were played each month. It is triggered at the end of every month. It reads from “Musical Piece” class details about broadcast pieces, composers and performers, then it reads from “Scheduled Program” details about musical pieces that were played this month, and finally it reports to the musicians’ association and the financial system (to be used for paying royalties).

The Design Phase

Defining Basic Methods

Figure 1 shows the class model after adding the basic methods. Although the elementary methods actually belong to a general class from which all the classes inherit, for demonstration purpose we show them within the specific classes as “Const” (add), “Del,” “Set,” and “Get.” The relationship methods are added to the proper classes depending on the relationship type of each class. These methods are overloaded according to their operations: “Add” for adding an object, “Del” for deleting, “Con” for connecting, “Dis” for disconnecting, and “Rec” for reconnecting. For example, ScheduledProgram.PieceScheduled.Con(PieceObject) checks if it is possible to connect the object PieceObject to the object ScheduledProgram when the editor decides to schedule it in the program; and ScheduledProgram.PieceScheduled.Dis(PieceObject) disconnects the object PieceObject from the object ScheduledProgram when the editor decides to delete it from the program.

Top-Level Design of the Application Transactions

The application transactions are derived from the OO-DFDs. In OO-DFD-0 (Figure 2), there are three transactions, each consisting of one elementary function, and the related classes, external-entities and data-flows. In OO-DFD-4 (Figure 3), there are four transactions: one consisting of function 4.1; one of function 4.2; one functions 4.3-4.7; and one - of function 4.8. In OO-DFD-5 (Figure 4), there are three transactions: one consisting of functions 5.1 and 5.2; one of function 5.3; and one of function 5.4.
Figure 5a shows the transaction diagram that consists of Function 1, and Figure 5b shows the transaction diagram that consists of Functions 4.3-4.7 (taken from OO-DFD-4). Note that sometimes (when component functions of a transaction belong to different OO-DFD) a transaction diagram may be taken from more than one OO-DFD (such a case does not appear in our example). Figures 6a and 6b demonstrate the top-level descriptions of these transactions, respectively. (The line numbers are not part of the descriptions; we will refer to them later on.) Note the “Input/Output” lines from/to external-entities, the “Read/Write” lines from/to object classes, the “Move from/to” lines between elementary functions, and the “Execute” lines for the elementary functions. The process logic of a transaction is expressed by means of “standard” programming patterns, namely, sequences, conditions and iterations (loops). Note that the process logic of a transaction is not derived “automatically” from the transaction diagram; rather, the analyst and user determine it.

**Interface Design—The Menus Class**

Figure 7 demonstrates three menus created for this system. The main menu contains five lines/items: three marked by “T” (for “trigger”), indicating lines that trigger transactions; and two marked by “S” (for “selection”) indicating lines that call other menus. The menu “Schedule a Music Program” consists of four “T” lines, each triggering a different Transactions method. (The numbers next to each line/item indicate the functions included in the transaction being triggered by the respective menu line/item.) The menu “Update/Report Broadcast Pieces” consists of three “T” lines, each triggering a different Transactions method.

Figure 8 presents the corresponding Menus class and its three menu objects. Note that each line/item in a menu becomes an attribute value of a respective menu object. The class has three main methods: “Display,” “Set,” and “Choose.” The method “Display” presents the respective menu object to the user. The method “Set” enables the user to set the value of the corresponding attribute value and return it. The method “Choose” gets the user’s selection (attribute value) as a parameter, and either sends a message to another menu object - if a “S” line is selected, or sends a message to a Transactions method - if a “T” line is selected (meaning that it fires a transaction). Figure 9 presents an implementation of the menu-tree of the System.

Figure 10 demonstrates the Menus methods including the main program. For example, when the program starts to run the main program is executed and it displays the main menu (Figure 9 - top window). If the user selects option 4, “Schedule a music Program,” he/she actually fires the “Set” method of Menu-0 object and number 4 is returned. This triggers the “Choose” Method with parameter 4 that makes the program display the middle window of Figure 9. Then, if he/she selects the third option, “Schedule a program,” the “Set” method of Menu-4 object is fired and returns 43 that triggered the “Choose” method with 43 as a parameter, which triggers the Transactions method “transaction_4.3-47()”. Had the user selected option 5, “Update/report broadcast pieces” (instead of option 4), the bottom window of Figure 9 would have been presented. Had the user selected option 1, “Input details of musical piece,” the transactions method “transaction_1()” would have been triggered, and Figure 12 would be presented.

**Input/Output Design**

In this stage the forms and the reports of each transaction are designed, based on the data items that flow on the data flows from external-entities to functions or from functions to external-entities. Then we add two object classes “Forms class,” and “Reports class.”
Figure 11 present these classes. Note that each input screen and output report is an instance (object) of one of these classes, respectively. The Forms and Reports classes have two methods: "Display" and "Help." The "Display" method of the "Forms" class displays an appropriate form on screen, enabling the user to enter values in the proper fields, and then saves them in a buffer. The "Help" method of the "Forms" class displays help on how to fill out the form. The "Display" method of the "Reports" class displays a report to the user, and the "Help" method presents more explanations on it.

Figure 12 is an example of a Forms object, "Musical Piece," that enables the user to input data on musical pieces. Selecting "Input a musical piece" from the main menu, Menu-0 object, triggers it. This user selection fires the method "Set" which returns number 1 and triggers the method "Choose(1)," which executes the Transactions method "transaction_1()" (see "Choose" method in Figure 10) that calls the "Display" method of the form "Musical Piece" and then calls the "Construct" method of class "Musical Piece." "Musical Piece" form consists of three text boxes (for the musical piece ID number, name and length of play), four combo boxes (for the music type, composers, performers and owners), and three commands buttons ("Add," "Help," and "Exit"). All these fields are attributes of the Data class: "Musical Piece" (see Figure 1).

Design of System Behavior

Figures 13a and 13b show pseudo-codes of the two transactions; Figure 14a shows the Message Diagram of the menus that triggers them, and Figures 14b and 14c show the Message Diagrams of the Transactions methods "transaction_1()" and "transaction_4.3-4.7()", respectively.

Figures 13a and 14b describe a "simple" transaction–Transaction 1–which was presented in Figure 5a, and top-level description–in Figure 6a. In essence, it involves methods of Forms and Musical_Piece classes.

Figures 13b and 14c describe a more complex transaction – Transaction 4.3-4.7 – whose diagram was presented in Figure 5b, and top-level description – in Figure 6b. Since it involves complex process logic, new specific methods are defined. An example of a specific method is "PreviousPieceInProgram.Get" of "Music Program" class, which retrieves all musical piece objects broadcast in the same kind of program a month before, or scheduled a month before or after the date of this program. Other examples are "GetObject (attributename, operator, attributevalue/s [,logicoperator, attributename, operator, attributevalue/s])" of each Data class, which retrieves objects that satisfy the condition within the passed parameters. For example, the method MusicalPiece.GetObject (MusicType, "in", Types) retrieves all musical piece objects whose music type appears in Types; it consists of a call to a specific method that translates the passed parameter to a condition and calls a basic method "Get" of this class that shows the attribute value of each of its object. Another example is the method of Schedule Program class: ScheduledProgram.GetObject(BroadcastTime, ",=", TimeValue, "and", MusicProgram, ",=", MusicProgValue), which retrieves the Scheduled Program whose Broadcast time equals to TimeValue and belongs to MusicProgValue.

The numbers within comment lines in Figures 13a and 13b refer to corresponding line numbers in the top-level descriptions of the transactions (see Figures 6a and 6b, respectively). For example, in Figure 13b we see that an Input/Output line translates to a Form/Report call (e.g., /*1*//*8*/), and a Read/Write line translates to a basic method call (e.g., /*2*//*25*/). The figures include detailed explanation remarks.
SUMMARY

The advantages of the integrated methodology presented in this chapter are: (a) System analysis (i.e., specification of user requirements) is performed in functional terms via OO-DFDs—a natural way for users to express their information needs, and in data terms via an initial OO-schema, or an ERD which is easily translated into an initial OO-schema. (b) System design follows the analysis and uses its products. The OO-schema is augmented with a Menus class that is derived from the menu-tree that was designed earlier from the OO-DFDs. Inputs and Outputs classes are also derived from the input forms and the outputs of the system (earlier products of the design stage). Basic methods are derived from the OO-schema structure, and the application programs, which are based on the transaction descriptions, are attached to the Transactions class. The Menus class enables the users to access and trigger any application transaction. (c) The end products of the design phase can be easily implemented with any OO programming environment.

Our further research and development agenda includes: development of a set of CASE tools to support the methodology; demonstration of the methodology in use by means of several real-world case studies; examination of the alternative performing orders of the analysis phase; and evaluation of the methodology by means of experimental comparisons with other methodologies. Comparisons are possible on various dimensions, e.g., comprehension of schemas by users, quality (i.e., correctness) of designed schemas, ease of learning the methods, etc. Participants in the experiments might be students in relevant programs or professionals in relevant development organizations.

ENDNOTE

1 An earlier version of this paper appeared in Journal of Database Management (2001), 12(1), 15-25.

REFERENCES


Chapter V

Object-Process Methodology Applied to Modeling Credit Card Transactions

Dov Dori
Israel Institute of Technology, Israel

Object-Process Methodology (OPM) is a system development and specification approach that combines the major system aspects--function, structure and behavior--within a single graphic and textual model. Having applied OPM in a variety of domains, this chapter specifies an electronic commerce system in a hierarchical manner, at the top of which are the processes of managing a generic product supply chain before and after the product is manufactured. Focusing on the post-product supply chain management, we gradually refine the details of the fundamental, almost “classical” electronic commerce interaction between the retailer and the end-customer, namely payment over the Internet using the customer’s credit card. The specification results in a set of Object-Process Diagrams and a corresponding equivalent set of Object-Process Language sentences. The synergy of combining structure and behavior within a single formal model, expressed both graphically and textually, yields a highly expressive system modeling and specification tool. The comprehensive, unambiguous treatment of this basic electronic commerce process is formal, yet intuitive and clear, suggesting that OPM is a prime candidate for becoming a common standard vehicle for defining, specifying, and analyzing electronic commerce and supply chain management systems.

BACKGROUND

Current object-oriented methods suffer from three major inter-related problems: the encapsulation problem, the complexity management problem, and the model multiplicity problem.

The encapsulation problem is a direct consequence of the OO encapsulation principle, which requires that any process be “owned” by some object, within which it is defined. While
being a helpful programming convention, a direct, unavoidable consequence of this encapsulation requirement is lack of explicit process modeling. Conforming to the OO encapsulation principle suppresses the dynamic aspect of the system and imposes an unnatural modeling of the real world, because processes usually involve more than one object class. Hence, while being a suitable programming paradigm, this unnecessary encapsulation constraint has been a source of endless confusion and awkward modeling of real-life situations.

The complexity management problem is rooted in the fact that OO methods cope with managing the complexity that is inherent in real-life systems by breaking it into various models, one for each aspect or facet of the problem: structure (the object/class model), dynamics (Statecharts), actors (use cases), etc. When the system is large and complex, no good tools are available to seamlessly present parts of the system at varying levels of complexity.

The closely related model multiplicity problem stems from the fact that the fundamental OO object/class model, which is at the basis of all OO methods, is inadequate for accommodating the functional and dynamic system aspects. OO methods must employ a host of models to specify the various aspects of the system. The currently accepted UML standard (Fowler, 1999; OMG, 2000) requires nine different models, including class diagram, use case diagram, message trace diagram, object message diagram, state diagram, module diagram, and platform diagram.

The model multiplicity problem refers to the need to comprehend and mentally integrate a variety of models of the same system and constantly take care of synchronizing among them.

This problem arises from the requirement to concurrently construct, maintain and consult several models that represent various system aspects. Some of the confusion caused by model multiplicity is expressed in the following excerpt (Kovitz, 1998) that discusses the best mix of using UML class diagrams (the static model) and collaboration diagrams (the dynamic model):

Class diagrams cannot stand alone. Neither can collaboration diagrams. They reinforce each other, and need to be developed concurrently with each other. Failure to develop these diagrams concurrently will result in dynamic models that cannot be supported statically, or static models that cannot be implemented dynamically.

We have empirically established (Peleg & Dori, 2000) that maintaining a clear and coherent image of the systems under development using such a plethora of models is a source of inherent difficulty. Comparing the major predecessor of UML–Object-Modeling Technique (Rumbaugh, Blaha, Permerlan, Eddy, & Lorsenson, 1991), to Object-Process Methodology (OPM), we prove that an approach that is capable of specifying systems with just one model is significantly better than a multi-model one.

Object-Process Methodology

Object-Process Methodology–OPM (Dori, 1995, 2000) is a systems development approach that responds to the challenges that problems with the aforementioned OO methods raise. Using a single, integrated graphic and natural language model, OPM caters to the natural train of thought developers normally apply while trying to understand and build complex systems that involve humans, hardware, and software. In such systems, it is usually the case that structure and behavior are intertwined so tightly, that any attempt to separate them is bound to further complicate the already complex description.

OPM achieves model integration by incorporating the three major system aspects—function, structure, and behavior—into a single model, in which both objects and processes
are adequately represented without suppressing each other. This approach counters contemporary object-oriented systems development methods, notably UML, which require several models to completely specify a system. OPM is therefore not yet another OO analysis and design method, as it recognizes the fact that separating structure from behavior while engaging in system modeling, which results in the model multiplicity problem discussed above, is counter-intuitive and therefore counter-productive.

To avoid model multiplicity, OPM incorporates the static-structural and behavioral-procedural aspects of a system into a single, unifying graphic-textual model. Founded on a concise and compact ontology, in which processes and state-exhibiting objects are the only building blocks, OPM generically models the structure and behavior of physical and informatical things. In the OPM ontology, objects are viewed as persistent, state-preserving things (entities) that interact with each other through processes—another type of things. Thing is a generalization of an object and a process. Processes are patterns of behavior that transform objects by transforming them. Transformation is a generalization of effect, consumption and generation. Hence, transforming objects implies affecting them (i.e., changing their states), or generating new objects, or consuming existing objects. The synergy of structure-behavior unification within a single model, combined with a dual graphic-textual model, results in a highly expressive modeling tool. The OPM approach combines the graphical and textual modalities.

**Graphics-Text Synergy in OPM Systems Development**

OPM uses Object-Process Diagrams (OPDs) for the graphic specification and Object-Process Language (OPL) for the textual specification. This combination of graphic and text may seem redundant from a pure information-content viewpoint. In fact, however, these two modalities complement each other from the user’s perspective, because they go hand-in-hand such that if the diagram reader encounters some unclear point on the graphic side, she or he can directly consult the analogous textual OPL specification. Conversely, if the text is not well understood at some point along the OPL script, the corresponding OPD sentence (a construct made of one or more OPD graphic symbols) can be examined to obtain clarification. This graphics-natural language combination is a major advantage of OPM for the target audience—the professionals for whom the system is being developed. However, the same graphics-text synergy is instrumental not only for the system specification readers, but also for the developers (system analysts and designers) already at the analysis and design phases.

The optimal scenario for quality systems development in terms of the professionals involved is a team comprised of one or more system architect and one or more domain experts. The domain expert knows his/her field best, but is usually not a software professional and is not supposed to be one. The system architect is proficient with systems theory and applications, but in general lacks deep knowledge of the domain within which the system is to be developed. Together, they gradually acquire knowledge about the current state of affairs surrounding the system under development. They record the knowledge accumulated using the combination of Object-Process Diagrams and Object-Process Language. When recording OPD symbols, immediate feedback is provided through OPL sentences. This enables real-time verification of the correctness of the intent and design. If the natural Formal English sentence does not reflect the designers’ intent, immediate rectification follows. The next section describes each of the graphic and text modalities and how they relate to each other.
Object-Process Diagrams (OPDs) and Object-Process Language (OPL)

OPM uses Object Process Diagrams (OPDs), drawn using OPCODE (Dori & Sturm, 1998) for expressing the objects of a modeled system and the processes that affect them. OPCODE responds to some of the challenges Jarzabek and Huang (1998) propose for current CASE tools. The OPDs are elaborate workflow-like hypergraphs that model the system or parts of it at various levels of detail. We present the notations and symbols OPDs use gradually as we show OPDs of supply chain management and the electronic commerce processes associated with them.

Objects and processes are connected by procedural links, which can be either enabling links or transformation links. These two different kinds of links are used in the OPD of Figure 1 to connect objects to processes, depending on the roles that these objects play in the process to which they are linked. Objects may serve as enablers—instruments or intelligent agents, that are involved in the process without changing their state. Objects may also be transformed (change their state, generated, consumed, or affected) as a result of a process acting on them.

An enabling link connects an enabler to the process that is enabled. Enabler is an enabling object that needs to be present in order for the process to occur, but it does not change as a result of the process occurrence. An enabling link can be an agent link or an instrument link. An agent link is denoted by a line with a black arrow at the process end, such as the one from the object Retailer to the process Post-Product Supply & Retailing. An agent link denotes that relative to the enabled process, the enabler is an intelligent agent—such as a human or an organizational unit that comprises humans, such as a department or an entire enterprise. An instrument link is an enabling link denoted by a white circle at the process end, which denotes that the enabler is an instrument—a non-human physical or informational object (machine, file, etc.) that must be present for the process to take place but is not affected by the process.

Figure 1 shows the underlying principle of how such bi-directional effect link is generated. Arrows denote transformation links, which can be effect links, consumption links and result links. Figure 1(a) shows how the process Retailing affects the object Product by transforming it from state manufactured to state retailed. In Figure 1(b), the states of Product are suppressed, and then there is no point in keeping the incoming and outgoing

Figure 1: Object states and effect links. (a) The process Retailing affects the object Product by transforming it from state “manufactured” to state “retailed.” (b) States of Product suppressed. (c) The incoming and outgoing links are merged to yield the bi-directional effect link.
effect links separate. In Figure 1(c) these two links are therefore merged into the bi-directional effect link.

The consumption link is a transformation link denoted as a unidirectional arrow from the consumed object to the consuming process.

Based on a constrained context-free grammar, a textual description in a natural-like Object-Process Language (OPL) can be automatically extracted from the diagrammatic description in the OPD set. Devoid of the idiosyncrasies and excess of cryptic details that feature current programming languages, OPL sentences are palatable to humans with no prior training, and can therefore be a prime candidate for becoming a business language for electronic commerce.

Applications of OPM

In this chapter we present an application of OPM to modeling the basic electronic commerce process of credit card transaction as a case in point to demonstrate OPM's semantics, systems, and breadth of scope. However, OPM is domain independent and, as we show below, has been applied to a large variety of domains. OPM does not depend on a particular domain of discourse but rather on fundamental definitions of an ontology that captures what an object and a process are. Due to this generic nature of OPM, it has been found to be most suitable for developing systems in a large variety of domains. In fact, modeling of any single domain, in which OPM has been attempted as a systems development tool, has produced enlightening results. Semiconductors manufacturing and sintering technology for metal-cutting tools manufacturing are two domains in which OPM has been effectively applied to produce large-scale operational systems.

Sintering technology entails mixing, pressing and “baking” rare metal powders in extremely precise conditions to obtain near-diamond hardness for metal cutting. Using OPM, a team of system analysts and designers specified all the technological processes involved in the manufacturing of inserts during a period of six months. They came up with a detailed, complete specification of the system to be developed, expressed as a set of Object-Process Diagrams. The OPM design document they produced served as a blueprint for the implementation of the system. The OPD set they produced was the document that constituted the basis for contracting with the software house that implemented the MANTI system. MANTI is currently operational and serves as the backbone of the technology know-how management at a world-leading insert manufacturer.

The Semiconductor Automated FAB Design Project involved ten system analysts who used the OPM as the framework for the analysis. OPM was used by 30 in-house and ten contractor programmers who developed the code for the system. The system was developed as an add-on to the WorkStream™ Manufacturing Execution System (MES) of the fab. The project included a detailed analysis of the fab information system. The system included as main objects the machines, the operators, the automated material handling system and the wafers. The main processes included the major manufacturing processes (i.e., etching, photolithography, diffusion, testing, etc.), the releasing process of wafers into the fab, the transfer of wafers from machine to machine, and all the transformations in the wafer status. As the project leader has noted, “The use of the OPM dramatically increased the effectiveness of the development process, since all parties (analysts, programmers and users) were able to use unified terminology that covered all aspects of the system.”

Other domains in which OPM has been successfully applied include studyware design (Dori & Dori, 1996), Computer Integrated Manufacturing (Dori, 1996a), image understanding
(Dori, 1996b), R&D management, (Meyersdorf & Dori, 1997), representation of control flow systems (Peleg & Dori, 1998), real-time systems (Peleg & Dori, 1999), and algorithm specification (Wenyin & Dori, 1999). In the area of document analysis and recognition, OPM was instrumental in works dealing with the Machine Drawing Understanding System and line detection (Wenyin & Dori, 1998).

The large variation of the domains listed above, in which OPM has already been successfully applied, demonstrates the generality of OPM, which makes it suitable for specifying systems independently of their domain of discourse. In particular, as we show in this chapter, modeling supply chain and electronic commerce systems with OPM provides complete and concise specification that is palatable to software and domain experts alike.

Credit Card Transactions

Figure 2, adapted from Textcor (1998), describes credit card processing in a free natural language, while an accompanying pictorial scheme provides the details of electronic shopping that precedes the credit card processing.

The description in Figure 2 concerns details of the Retailing process, which is the focus of the OPD depicted in Figure 3 and its corresponding OPL paragraph in Figure 4. The two modalities—the graphic specification through the OPD and the textual one through the OPL—complement each other and reinforce the clarity of the system specification. The structure and behavior of Retailing are self-explanatory to the extent that there is not much to be added without simply repeating the content of the OPL sentences. In what follows, we examine the system specification obtained so far to study several unique features of OPM.

Modeling Prose Specification

While modeling a system (or "problem statement") given in prose, some details are not modeled in the OPM model, while others that are not explicit in the prose statement are expressed in an explicit manner. Thus, in Figure 2, the word "Regretfully" in the sentence "Regretfully there is sometimes a further stage where the customer is dissatisfied and arranges for the transaction to be cancelled" is not modeled, since it is a state of mind that is not very relevant in the model. This is where Actor Network Theory, discussed below, could be useful.

Structural Links and Fundamental Structural Links

The word describes in the OPL sentence, "Catalog describes many End Products," of Figure 4 is written in the OPD of Figure 3 along an arrow with an open arrowhead, which symbolizes a (general, or tagged) structural link. Unlike procedural links, which connect a process with an object or object state, structural links connect one object to another or one process to another. The name of the relation, describes in our case, is written such that a legal natural English sentence is produced when the source object (e.g., Catalog), the relation name (tag), and the destination object (End Product) are listed in this order. The word many is reflected by the “m” next to the arrowhead in Figure 3, which symbolizes a one-to-many participation constraint. Aggregation, generalization, characterization and instantiation are fundamental structural relations.

Fundamental structural relations are the four structural relations which, due to their frequent usage, are denoted by special (triangular) symbols. Characterization, for example, is denoted by a black-on-white triangle, as shown in Figure 5 and translated to the OPL sentence, "End Customer features Satisfaction."
What is involved in credit card processing?
The steps in credit card processing are as follows.

Authorization
The merchant must first obtain authorization for the charge from the merchant’s credit card processing company. Authorization simply means that the card has not been reported stolen, and there is sufficient credit on the card. It results in the customer’s credit limit being temporarily reduced by the value of the transaction.

There are two ways in which authorization may be obtained:
1. Manual: The merchant downloads details of the sale from the computer that is acting as web server. The merchant then requests authorization using their normal method such as a point of sale (POS) terminal or PC program.
2. Automatic: The server software communicates directly with the credit card processing company computer and arranges authorisation online.

Clearly option 2 is preferred, but this is more complex and the costs are greater.

Capture
The final stage is for the credit card to be debited. This can happen at the same time as authorization provided the merchant guarantees that delivery will take place within a certain fixed time. Otherwise capture should take place when the goods are shipped.

If the merchant’s business is such that capture can take place immediately, then this can also happen automatically. Otherwise a second manual process is required.

Charge-back
Regrettably, there is sometimes a further stage where the customer is dissatisfied and arranges for the transaction to be cancelled. Because many Internet sales are made to overseas customers, many banks perceive that there is an increased risk of charge-backs. It has been reported that some merchants will not accept orders to Russia because of the frequency of charge-back.

Note that the fact that a payment has been authorized by the bank does not provide any protection against charge-back.
Figure 3: Zooming into the retailing process

Figure 4: The OPL paragraph of the OPD in Figure 3

Retailer handles Retailing.
Retailing zooms into Selecting & Adding, Credit Card Submitting and Credit Card Processing.
Retailer handles Selecting & Adding.
Selecting & Adding requires Catalog.
Catalog describes many End Products.
Selecting & Adding changes End Product from distributed.
Selecting & Adding changes Virtual Cart from empty to filled.
Checking Out occurs if Virtual Cart is filled.
End Customer handles Checking Out.
Checking Out yields Transaction Amount.
End Customer handles Credit Card Submitting.
Credit Card Submitting yields Credit Card Details.
Credit Card Processing requires Credit Card Details.
Credit Card Processing changes End Product to retailed.

Credit Card Processing is represented in Figure 3 as a sub-process of Retailing. To model the free text description of Figure 2, in Figure 5, we zoom into Credit Card Processing, and in Figure 6 we provide the corresponding OPL paragraph. As before, the combination of the OPD and OPL is almost self-explanatory.
Control Structures

Figure 5 demonstrated two basic control structures: if-then and if-then-else. The object Satisfaction and its states exemplify the if-then statement. The OPL sentence is Back-Charging occurs if Satisfaction is low.
The object Authorization and its states exemplify two if-then statements:
Capturing occurs if Authorization is granted, and Notifying occurs if Authorization is denied.

Since there are only two Authorization values, these two OPL sentences could be combined into the following single OPL sentence: Capturing occurs if Authorization is granted, else Notifying occurs.

A case statement is a generalization of these examples, where the number of states is not limited to two. In Peleg and Dori (1998) we show how other control structures, like loops and recursion, can also be explicitly modeled within OPDs.

Distributing Procedural and Structural Links

A link from an enabling object to the circumference of a zoomed-in process implies that the link is attached to each one of the sub-processes within the zoomed in process. Thus, for example, the instrument link from the object Transaction Amount to the abstract, higher-level process Credit Card Processing implies that Transaction Amount serves as instrument for all four lower-level processes—Authorizing, Capturing, Notifying and Back-Charging—that comprise Credit Card Processing and are depicted within its zoomed-in enclosing ellipse. The enclosing high-level process acts as a graphic shorthand notation that saves drawing many links from the enabling object to each one of the enabled processes.

This is analogous to a pair of parentheses in an algebraic expression that act to shorten the expression through the distributive law, where $a\hat{A}(b+c+d+e)$ is equivalent to $a\hat{A}b + a\hat{A}c + a\hat{A}d + a\hat{A}e$. In this case, $a$ is the object Transaction Amount, $\hat{A}$ is the instrument link, and $b, c, d$ and $e$ are the four processes—Authorizing, Capturing, Notifying and Back-Charging—that comprise Credit Card Processing. More generally, $\hat{A}$ can be any relation or operation that obeys the distributive law and $a, b, c, d$ and $e$ the proper operands.

The distributive nature of relations exists not only for procedural relations but also for structural ones. Thus, in Figure 8, the structural relation holds emanating from Web-Server is common to both Transaction Amount and Credit Card Details. Therefore, in Figure 9, instead of writing the two separate sentences, Web-Server stores Transaction Amount, and Web-Server stores Credit Card Details, we write shortly, in one sentence: Web-Server holds Transaction Amount and Credit Card Details.

It should be noted that this is a simplified model as it does not take into account the distinction between the card-issuing bank and the acquiring bank—the bank that has a business relationship with a merchant and receives all credit card transactions from that merchant (Rosenberg, 1993).

Our model does not include the Interchange Fee— a fee the acquiring bank pays to the credit card-issuing bank in order to process a credit card transaction involving a card holder’s account (Rosenberg, 1993). The Merchant Discount, which is a percentage of the retail sale the merchant pays as a fee to the acquiring bank for processing the credit card transaction, is not accounted for either. The more accurate model, based on Lamond (1996), appears in Figure 7. For the sake of brevity, we refer in this model to both the acquiring bank and the credit card-issuing bank as the Credit Card Processing Company.
Figure 5: Zooming into credit card processing of Figure 3

Figure 6: The OPL paragraph of the OPD in Figure 5

Retailer handles Credit Card Processing.
Credit Card Processing Company handles Credit Card Processing.
Credit Card Processing requires Transaction Amount.
Credit Card Processing requires Credit Card Details.
Credit Card Processing affects End Customer.
Credit Card Processing zooms into Authorizing, Capturing, Notifying and Back-Charging.
Authorizing yields Authorization.
Authorization can be granted or denied.
Notifying occurs if Authorization is denied.
Notifying yields Notification.
Capturing occurs if Authorization is granted.
Capturing affects Income.
Retailer earns Income.
End Customer exhibits Satisfaction.
Satisfaction can be high or low.
Back-Charging occurs if Satisfaction is low.
Back-Charging affects Income.
Paths, Use Cases and Threads of Execution

Figure 8 and Figure 9 deal with the Authorization process. In the narrative text that describes the system (Figure 2), we find that there is a manual option and an automatic one. There are two ways in which authorization may be obtained:

1. **Manual:** The merchant downloads details of the sale from the computer that is acting as web server. The merchant then requests authorization using his/her normal method, such as a point of sale (POS) terminal or PC program.

2. **Automatic:** The server software communicates directly with the credit card-processing company computer and arranges authorization online.

We denote these two alternatives as two different paths, marked **Manual** and **Automatic.** A path in an OPD is a collection of procedural relations. Together they denote a **use case** - a possible scenario or happening, or part of a scenario in the systems that needs to be differentiated from one or more use cases.

Boolean Objects

A Boolean object is an object with exactly two states (values): yes and no. Its name is phrased as a statement containing the reserved word “is” and ending with a question mark, which uniquely identifies a Boolean object. There are two Boolean objects in Figure 10: **Credit Card is Reported Stolen?** and **Credit Limit is Exceeded?** These have been introduced to model the system specification of Figure 2 that reads:

“Authorization simply means that the card has not been reported stolen, and there is sufficient credit on the card.”

The OPL sentence pair that refers to a Boolean object is phrased so as to make sense in a natural language. For the **Credit Card is Reported Stolen?** Boolean object, the two OPL sentences, which appear in Figure 11, are as follows:

- **Legal Use Denying** occurs if **Card is Reported Stolen.**
- **Credit Limit Checking** occurs if **Card is not Reported Stolen.**

Application Generation From the OPL Specification

Figure 12 specifies precisely the following system requirement denoted in Figure 2:

It (authorization) results in the customer’s credit limit being temporarily reduced by the value of the transaction.

Figure 12 and Figure 13 are the pair that specifies the **Limit Checking & Decreasing** process.

Summarizing the credit card transaction system specification, we see that we started with **Retailing**—a very broad system definition. Following a series of refinements through zooming into processes and unfolding the associated objects, we ended up with a very concrete, down-to-earth minute detailed process such as decreasing the credit limit of an end customer. Theses can be automated to generate code.

OPM and Actor-Network Theory

Actor-Network Theory (ANT) was born out of ongoing efforts within the field called social studies of science and technology and developed from studies in two related but distinct fields: the social practice of science and the introduction of new technologies. An early paper by Latour and Woolgar (1979) looks at struggles over scientific truth in a
**Figure 7:** A more detailed model of the authorization and capturing processes that includes the interaction between the acquiring bank and the card issuing bank

Merchant transmits the credit card data and sales amount with a request for authorization of the sale to its acquiring bank. The acquiring bank that processes the transaction routes the authorization request to the card-issuing bank. The credit card number identifies type of card, issuing bank, and the cardholder’s account. If the cardholder has enough credit in his/her account to cover the sale, the issuing bank authorizes the transaction and generates an authorization code. This code is sent back to the acquiring bank. The acquiring bank processes the transaction and then sends the approval or denial code to the merchant’s point of sale unit. Each point of sale device has a separate terminal ID for credit card processors to be able to route data back to that particular unit.

Laboratory, while one of Callon’s early studies (1986) considers fishermen and scallops as some of the stakeholders in a changing fishing industry. These examples already exhibit some of the main features of ANT.

The underlying idea of ANT is that business is never done in a total vacuum but rather under the influence of a wide range of surrounding factors. The actors may be humans, organizations, cultures, ideas, animals, plants or inanimate objects, and these are treated symmetrically, irrespective of their ontology. These actors have interests that are represented (in both the semiotic and political senses) by themselves and other actors. In line with its semiotic origin, actor network theory is granting all entities of such a heterogeneous network the same explanatory status (Akrich & Latour, 1992, p.259).
Figure 8: Zooming into authorizing of Figure 5

Figure 9: The OPL paragraph that corresponds to the OPD in Figure 8

Web-Server holds Transaction Amount and Credit Card Details.
Authorizing zooms into Manual Downloading & Keying, Automatic Data Transferring and Verifying.
Manual Downloading & Keying requires Web-Server and Credit Card Processing Company.
Automatic Data Transferring requires Web-Server and Credit Card Processing Company.
Credit Card Processing Company enables either Manual Downloading & Keying or Automatic Data Transferring.
Web-Server enables either Manual Downloading & Keying or Automatic Data Transferring.
Manual Downloading & Keying requires either Point POS or PC.
Verifying requires Transaction Amount and Credit Card Details.
Verifying yields Authorization.

ANT and OPM share in common breadth of scope that extends well beyond information systems in their traditional sense. Both ANT and OPM view the entire universe as the stage where existence and action take place. ANT actors are OPM objects, and ANT actions are OPM processes.

The act an ANT actor carries out and all of the influencing factors should be considered together. The actor-network is a shifting system of alliances and exchanges among the actors. It is the act linked together with all of its influencing factors (which again are linked), producing a network (Hanseth & Monteiro, 1998). An actor network consists of and links together both technical and non-technical elements. Not only the car’s motor capacity, but
Figure 10: Zooming into credit card processing of Figure 8

Figure 11: The OPL paragraph of the OPD in Figure 10

Verifying requires Credit Card Company.
Verifying zooms into Legal Use Verifying, Legal Use Denying, Credit Limit Checking, Credit Denying and Approving.
Legal Use Verifying requires Credit Card Details.
Legal Use Verifying determines whether Card is Reported Stolen.
Legal Use Denying occurs if Card is Reported Stolen.
Credit Limit Checking & Decreasing occurs if Card is not Reported Stolen.
Credit Limit Checking & Decreasing requires Transaction Amount and Credit Limit.
Credit Limit Checking & Decreasing determines whether Credit Limit is Exceeded.
Credit Denying occurs if Credit Limit is Exceeded.
Approving occurs if Credit Limit is not Exceeded.
Credit Denying yields denied Authorization along the credit thread.
Legal Use Denying yields denied Authorization along the theft thread.
Approving yields granted Authorization.
Notifying generalizes Alerting and Customer Notifying.
Customer Notifying requires denied Authorization along the credit thread.
Customer Notifying yields Notification.
Alerting requires denied Authorization along the theft thread.
Credit Card Company handles Alerting.
Customer Notifying yields Notification along the credit thread.
Capturing requires granted Authorization.
Figure 12: Zooming into credit limit checking

![Diagram showing credit limit checking process]

Figure 13: The OPL paragraph of the OPD in Figure 12

Limit Checking & Decreasing zooms into Comparing, Credit Limit Decreasing and Continuing.

Comparing requires Transaction Amount and Credit.
Comparing determines whether Limit is greater than Amount.
Credit Limit Decreasing occurs if Limit is not greater than Amount.
Continuing occurs if Limit is greater than Amount.
Credit Limit Decreasing affects Credit Limit.
Credit Limit Decreasing establishes that Credit Limit is not Exceeded.

also one’s driving training and conditions influence the driving. Hence, ANT talks about the heterogeneous nature of actor networks.

In OPM terms, an ANT network is the system defined by an OPM specification, which is expressed by two completely equivalent modalities: a set of Object-Process Diagrams—OPD set, and an equivalent collection of Object-Process Language sentences—the OPL script. Like ANT, OPM can incorporate non-technical objects and processes, including humans, political or industrial organizations, and any kind of inanimate item or living organism along with its structure and behavior. Behavior is the dynamics of each object and of the system as a whole. It is exhibited as a transformation in one or more objects in the system, which can be a change in an object’s state, its generation or its consumption.

Akrich and Latour (1992) claim several advantages for the ANT approach. It is symmetrical with respect to type of actor, it treats humans and machines equally. It is
symmetrical with respect to outcome: failures have the same types of explanation as successes. It is symmetrical with respect to causality: each actor influences and is influenced by other actors and the network as a whole.

The equal treatment of people and machines has been criticized, but Underwood (1998) suggests that it may be realistic in terms of power relations, and it prevents issues from becoming invisible when their representation is transferred (translated) to an actor of a different type. If, for example, some data collection functions are transferred from police informers to computer programs, it is still important to be able to talk about the power relations and motives of the collectors and their allies.

Like ANT, OPM provides the system developer with facilities to account for actors, which are the OPM enablers—agents and instruments. Unlike ANT, though, agents are humans or organizations with intelligence and intent, while instruments are any physical or informational objects that are non-human and, therefore, are not characterized by free will and intent. While it may be important to be able to talk about the power relations and motives of the collectors and their allies in the above example, the computer that stores that information is a mere instrument that cannot, on its own, make any political use of the information it stores. In OPM, a clear distinction is therefore made between the two enabling types. Moreover, while an OPM process transforms (affects, consumes or generates) at least one object, enablers do not change as a result of the occurrence of a process.

The interests of ANT actors are represented by scripts, usually imperative statements such as “shut the door,” “pay your taxes,” or “calculate the gross pay.” Akrich and Latour (1992) give a comprehensive set of definitions of script-related processes (such as inscription and conscription). These processes describe (amongst other things) the translation of scripts among actors, often involving a change of medium, for example, from conditioned response in a human to lines of code in a computer program.

ANT scripts are OPM processes in the sense that they specify what transformation an object undergoes. Thus, the script “shut the door” is like the Door Shutting” process in OPM, where some enabler (agent or instrument) changes the state of the object Door from “open” to “closed.” Likewise, the “pay your taxes” ANT script is the “Tax Paying” OPM process, where an agent (the object Citizen) changes the state of the object Tax from “unpaid” to “paid.” Gross Pay Calculating” is the OPM process of the ANT script “calculate the gross pay.” Here, an instrument (the object Computer) or an agent (the object Clerk) changes “Gross Pay” from “unknown” to “calculated.”

Of particular interest is ANT’s idea of description (de-scription), the discovery of the words behind the things or actions. This discovery is only possible in contrived, exotic or crisis situations, such as reengineering, consultancy or system failure, in a time such as IS development when nothing is taken for granted. Indeed OPM users experience a similar phenomenon. While trying to explicitly determine all the processes and objects with their states in the system, even the most experienced domain experts frequently find themselves in a situation where they need to define and invent names for things that are central to the domain of discourse. These are things with which the domain experts had been working for years without giving themselves or others account about the exact nature of those things.

Scripts are imperative but don’t have intentions; actors do. An actor can develop a “program of action” (Akrich & Latour, 1992), perhaps with the intention of maximizing the number of actors following a particular script. Some actors may avoid this by following an anti-program. A program of action can include the creation of new actors suitably inscribed. The inscription is most effective if it becomes irreversible, if the actor is, with respect to that script, a “black box” and the script becomes inaccessible to other actors.
ANT helps us to understand the course of a project or enterprise. We can asks questions such as “How did it come to turn out this way?” (through the changing alliances of actors), “Who is influencing it?” (who has been doing what scripting?) or “Why are some actors acting this way?” (what scripts are they carrying?). OPM goes a long way in pinpointing cause and effect. Consider the OPD of Figure 10. One can tell exactly why, for example, authorization for some transaction was denied.

Some of the more spectacular applications of ANT have been to the genealogy of now well-established scientific theories (Latour, 1987), the meaning of simple technical devices (Latour, 1992) or the acceptance of a new product (Bijker, 1992). More recently ANT has been applied to the development of information systems. Monteiro and Hanseth (1996) claim that ANT allows a finer-grained analysis of information systems than some other interpretive approaches which can treat all information systems as essentially similar.

The majority of IS development projects still follow, albeit loosely (Fitzgerald, 1997), methodologies derived from the systems approach to problem solving popularized by the RAND corporation in the 1950s (Optner, 1973). This approach takes us through the steps of problem definition, search for solutions, selection of the best alternative, implementation and evaluation. As with many methodologies, what starts as a description of how particular projects were done soon becomes a normative or prescriptive theory, which promises future success. This may be a reasonable transformation if the original projects were successful, but after 40 years of IS development, several factors encourage IS researchers to look for different theoretical bases for methodologies (Underwood, 1998).

Firstly, a large percentage of computer-based information systems are generally acknowledged to provide less than satisfactory service to end-users and to fall short of their original objectives. While some authors have attributed these failures to developers not following accepted theories, there is always the suspicion that the theories themselves may be at fault (Beath & Orlowski, 1994). OPM can be a concrete framework that serves as a unifying theory in itself, as well as a platform with which more abstract (but less strictly defined) theories, such as ANT, can be explicitly expressed and discussed.

To summarize this section, a comparison between OPM and ANT has revealed a host of common features between the two approaches. OPM can be a prime vehicle to express advanced ANT ideas and serve as a basis for developing and proving ANT claims and hypotheses.

**SUMMARY**

This paper has presented the Object-Process Methodology (OPM) as a viable approach for precise and explicit specification of complex systems, such as manufacturing, enterprise resource planning, electronic commerce, and supply chain. To demonstrate the relevance of OPM to electronic commerce, we first analyzed a broad, generic supply chain management system and its supporting electronic commerce infrastructure. Gradually, we narrowed our focus to the electronic commerce aspects of the system. In particular, we focused on the final stages of electronic commerce that takes place between the retailer and the customer. Within the retailing process, we further focused on credit card-based payment, which is a broadly practiced electronic commerce activity. While noting that business-to-business electronic commerce can also practice this method of payment, we restricted the analysis to the interaction between the end customer and the retailer with the involvement of the credit card-processing company.
OPM appeals to the human intuition as it combines system structure and behavior into a single model that caters to the natural train of thought humans apply while trying to understand a complex system. As shown throughout the chapter, the synergy of combining structure and behavior using the formal graphic representation along with a corresponding equivalent textual representation yields a system specification tool with high expressive power. Due to these virtues of formality on one hand and intuitiveness on the other hand, Object-Process Methodology is most suitable as an infrastructure for Internet-based systems engineering environment, within which inter-corporate business processes, such as ERP, and other electronic commerce activities can be seamlessly defined and conducted.

REFERENCES


Chapter VI

The Psychology of Information Modeling

Keng Siau
University of Nebraska–Lincoln, USA

Information modeling is the cornerstone of information systems analysis and design. Information models, the products of information modeling, not only provide the abstractions required to facilitate communication between the analysts and end-users, but they also provide a formal basis for developing tools and techniques used in information systems development. The process of designing, constructing, and adapting information modeling methods for information systems development is known as method engineering. Despite the pivotal role of modeling methods in successful information systems development, most modeling methods are designed based on common sense and intuition of the method designers with little or no theoretical foundation or empirical evidence. Systematic scientific approach is missing! This chapter proposes the use of cognitive psychology as a reference discipline for information modeling and method engineering. Theories in cognitive psychology are reviewed in this chapter and their application to information modeling and method engineering is discussed.

INTRODUCTION

Even though research in systems analysis and design has been going on for over 40 years, successful software development is still an art rather than a science. In the 1980s, Jones (1986) observed that a typical project was one year late and 100% over budget. Yourdon (1989) reported application backlogs of four to seven years or more. The maintenance phase typically consumed up to 70% of all programmer effort, and it was errors, not enhancements, that accounted for 40% of maintenance (Rush, 1985). Page-Jones (1988) wrote: “It looks as if traditionally we spend about half our time making mistakes and the other half of our time fixing them.”

We are, however, no better off coming towards the end of this century. The IBMs Consulting Group (Gibbs, 1994) released the results of a survey of 24 leading companies that had developed large distributed systems. The numbers were unsettling: 55% of the projects
cost more than budgeted, 68% overran their schedules, and 88% had to be substantially redesigned. A recent high-profile failure is the Denver Airport baggage-handling system, responsible for delaying the opening of the airport. The Standish Group research (Chaos, 1995) predicted that a staggering 31.1% of projects would be canceled before they ever get completed and 52.7% of projects would cost 189% of their original estimates.

In the early days of computerized information systems, technological failure was the main cause in the failure of business data processing systems (Avison & Fitzgerald, 1995). Today, the failure of information systems is rarely due to technology that is on the whole reliable and well tested. Failure is more likely to be caused by miscommunication and misspecification of requirements. Similar sentiments were echoed in the Standish Group’s report (Chaos, 1995) which listed incomplete requirements and specifications as the second most important factor that caused projects to be challenged and the top factor that caused projects to be impaired and ultimately canceled. A recent survey of hundreds of Digital’s staff and an analysis of the corporate planning database revealed that, on average, 40% of the requirements specified in the feasibility and requirements phase of the life cycle were redefined in the later phases. This cost Digital an average of 50% more than budgeted (Hutchings & Knox, 1995).

The process of investigating the problems and requirements of the user community, and building an accurate and correct requirement specification for the desired system is known as information modeling (Siau, 1999; Siau, Rossi, 1998; Siau, Wand, & Benbasat, 1997; Mylopoulos, 1992; Rolland & Cauvet, 1992; Kangassalo, 1990).

**INFORMATION MODELING**

*Information modeling* is the process of formally documenting the problem domain for the purpose of understanding and communication among the stakeholders (Siau, 1999; Siau, 1998; Mylopoulos, 1992). Information modeling is central to information systems analysis and design, and takes place in the early phases of the software development life cycle. The product of the information modeling process is one or more information models (e.g., data-flow diagrams, entity-relationship diagrams, use cases, activity diagrams, sequence diagrams). *Information model* provides a conceptual basis for communicating and thinking about information systems (Willumsen, 1993), and a formal basis for tools and techniques used in the design and development of information systems (Kung & Solvberg, 1986).

Information models are constructed using *information modeling method*, which can be defined as an approach to perform modeling, based on a specific way of thinking consisting of directions and rules, and structured in a systematic way (Brinkkemper, 1996). There is no shortage of information modeling methods in the field. In fact, it is a “methodology jungle” out there (Avison & Fitzgerald, 1995). Olle, Sol, and Verrijn-Stuart (1982) and Bubenko (1986) stated that the field was inundated by hundreds of different modeling methods. Recently, Jayaratna (1994) estimated that there were more than a thousand brand name methodologies worldwide. The quest to develop the next modeling method has been wittily termed the YAMA (Yet Another Modeling Approach) syndrome (Oei, van Hemmen, Falkenberg, & Brinkkemper, 1992) and NAMA (Not Another Modeling Approach) hysteria (Siau, Wand, & Benbasat, 1996). Even the new kid on the block, object-oriented approach, has more than a dozen variants. Despite the “impressive” number, miscommunication and misspecification continue (Chaos, 1995).

To reduce the chances of misunderstanding and miscommunication during information modeling, the use of natural and intuitive modeling constructs (e.g., entity, relationship,
object) in information modeling methods has been stressed and advocated (e.g., Chen, 1976; Coad & Yourdon, 1991). This, they claimed, would enable end-users to better understand the information depicted in the information model and to pinpoint incomplete or incorrect information in the model.

**METHOD ENGINEERING AND MODELING CONSTRUCTS**

*Modeling constructs* are semantic primitives that are used to organize and represent knowledge about the domain of interest (Sernades, Fiadeiro, Meersman, & Sernadas, 1989). Modeling constructs form the core of an information-modeling method. *Method engineering* is the process of designing, constructing, and adapting modeling methods for the development of information systems (Siau, 1999; Siau, 1998; Brinkkemper, 1996). To design, construct, and adapt methods, we need to understand the role and value of each modeling construct.

The importance of modeling constructs can be viewed from two perspectives: ontology and epistemology of information systems analysis and design. Ontology is concerned with the essence of things and the nature of the world (Wand & Weber, 1993; Avison & Fitzgerald, 1995). The nominalist position in ontology argues that “reality is not a given immutable ‘out there,’ but is socially constructed. It is the product of human mind” (Hirschheim & Klein, 1989). The choice of modeling constructs, therefore, directly influences what the modeling method regards as important and meaningful versus what it suggests as unimportant and irrelevant. For example, the use of the entity-relationship (ER) approach emphasizes entities and relationships but ignores the processes involved. The use of the object-oriented (OO) approach, on the other hand, emphasizes objects and the behavior of objects.

Epistemology relates to the way in which the world may be legitimately investigated and what may be considered as knowledge (Avison & Fitzgerald, 1995). The choice of modeling constructs constrains how one can know or learn about reality—the basis of one’s claim to knowledge (Klein & Lytyinen, 1983; Walsham, 1993). Users of the entity-relationship approach, for example, would focus on identifying entities and relationships, whereas users of data-flow diagrams (DFD) would emphasize the eliciting of processes, data flows, external entities, and data stores from the problem domain.

Despite the importance of modeling constructs, not much research has been done in this area. Most modeling constructs are introduced based on common sense, superficial observation, and the intuition of researchers and practitioners. Theoretical foundation and empirical evidence are either non-existent or considered non-essential. For example, Coad and Yourdon (1991) nicely summed up the practitioners’ scant concern:

*It would be intellectually satisfying to the authors if we could report that we studied the philosophical ideas behind methods of organization, from Socrates and Aristotle to Descartes and Kant. Then, based on the underlying methods human beings use, we could propose the basic constructs essential to an analysis method. But in truth we cannot say that, nor did we do it.* (p. 16)

With this laissez-faire attitude, one can not help but cast doubts on the usefulness and importance of some of these modeling constructs. It is probable that some of these constructs are not actually actors in the modeling drama, but merely incidental artifacts, created by researchers to help them categorize their observations. These artifacts may play no significant role whatsoever in modeling the real world. A reference discipline to guide the design, construction, and adaptation of modeling constructs for information-modeling methods is needed!
In this chapter, we propose the use of cognitive psychology as a reference discipline in the engineering of methods and in the studying of information modeling. Card, Moran and Newell (1983) wrote “advances in cognitive psychology and related sciences lead us to the conclusion that knowledge of human cognitive behavior is sufficiently advanced to enable its applications in computer science and other practical domains” (p.1). Moray (1984) also argued for the use of knowledge accumulated in cognitive psychology to understand and solve applied problems. Researchers in human-computer interaction have demonstrated that such an effort is valuable and essential in building a scientific understanding of the human factors involved in end-users’ interaction with computers. We believe that similar effort will be useful in information modeling and method engineering.

HUMAN INFORMATION-PROCESSING SYSTEM

To understand the representation and use of knowledge by humans, we need to approach it from a human information-processing perspective. The information-processing paradigm views thinking as a symbol-manipulating process and uses computer simulation as a way to build theories of thinking (Simon, 1979). It attempts to map the flow of information that a human is using in a defined situation (Gagne, Yekovich, & Yekovich, 1993) and tries to understand the general changes of human behavior brought about by learning (Anderson 1995).

According to Newell and Simon (1972), all humans are information-processing systems (IPS) and hence come equipped with certain common basic features. Although some of the processes used by the system may be performed faster or better by some than by others, the nature of the system is the same. One of the popular and most well-known human information-processing model is the Adaptive Control of Thought (ACT) proposed by Anderson (1983, 1995) (see Figure 1).

An ACT production system consists of three memories: working, declarative, and production. Working memory contains the information that the system can currently access, consisting of information retrieved from long-term declarative memory, as well as temporary structures deposited by encoding processes and the action of productions (Anderson, 1983). Declarative and production are long-term memory, the former being facts and the latter being processes or procedures that operate on facts to solve problems. Declarative knowledge is knowing that something is the case whereas procedural knowledge is knowing how to do something (Gagne et al. 1993).

Encoding deposits information about the outside world into working memory whereas performance converts commands in working memory into behavior. The storage process can create permanent records in declarative memory of the contents of working memory and can increase the strength of existing records in declarative memory. The retrieval process retrieves information from declarative memory into working memory. During the match process, data in working memory are put into correspondence with the conditions of productions. The execution process deposits the actions of matched productions into working memory. The whole process of production matching followed by execution is known as production application.

Working Memory

The working memory is activation based; it contains the activated portion of the declarative memory plus declarative structures generated by production firings and perception. Working memory is a temporary memory that cannot hold data over any extended
duration. Information in this memory store decays within about 10 seconds (Murdock, 1961) unless it is rehearsed. In addition to being limited duration, working memory is also of limited capacity. Miller (1956) claimed that working memory holds $7 + 2$ units of information while Simon (1974) claimed that it holds only about five units. Whatever the actual number, the important point is that it is small. Because of its small size, working memory is often referred to as the “bottleneck” of the human information-processing system.

**Declarative Knowledge**

There are two types of long-term memory—declarative and procedural. The long-term declarative memory is represented in the form of a semantic net. A basic unit of declarative knowledge in the human information-processing system is proposition and is defined as the smallest unit of knowledge that can possess a truth value (Anderson, 1983). Complex units of knowledge are broken down into propositions. Propositions have at least two parts. The first is called the *relation*. Verbs and adjectives typically make up the relations of a proposition. The second part of the proposition is called the *argument*, which is determined by the nouns in the proposition. Arguments are given different names depending on their role in the proposition. Arguments may be *subjects, objects, goals* (destination), *instruments* (means), and *recipients*. 
The declarative knowledge for the ER approach can be represented as propositions as shown below. Each proposition comprises a relation, followed by a list of arguments:

(i) represent, entity, rectangle
(ii) represent, relationship, diamond
(iii) comprise, ER, entity
(iv) comprise, ER, relationship

These four propositions can be depicted diagrammatically using Kintsch’s system as shown in Figure 2.

In ACT, individual propositions can be combined into networks of propositions. The nodes of the propositional network stand for ideas, and the linkages represent associations among the ideas (Anderson, 1983). Figure 3 shows the network of propositions for the ER approach.

Figure 2: Diagrammatic representation of propositions for ER approach
Procedural Knowledge

Unlike declarative knowledge, which is static, procedural knowledge is represented in the form of productions. Each piece of knowledge is called a production because it "produces" some bit of mental or physical behavior. Productions are formally represented as IF-THEN contingency statements in which the IF part of the statements contains the conditions that must exist for the rule to be fired and the THEN part contains the action that will be executed when the conditions are met. The productions are also known as condition-action pairs and are very similar to the IF-THEN statement in programming languages. For example, the following is the production rule for identifying a relationship construct in the ER model.

IF Figure is a diamond shape
THEN Figure represents a relationship construct

Productions can be combined to form a set. A production system, or set, represents all of the steps in a mental or physical procedure. The productions in the production systems
are related to one another by the goal structure. In other words, each production contributes in some way to achieve the final goal behavior. The use of goals and subgoals in productions creates a goal hierarchy that interrelates the productions into an organized set. For example, Table 1 shows a production system to understand an ER diagram.

**Domain-General Versus Domain-Specific**

Procedural knowledge can be discussed from two dimensions. The first dimension refers to the degree to which procedural knowledge is tied to a specific domain, with the anchor points of the continuum being termed domain-general and domain-specific (Gagne et al., 1993). Domain-general knowledge is knowledge that is applicable across domains and domain-specific knowledge is specialized because it is specific to a particular domain. The term “domain” refers to any defined area of content and can vary in its breadth.

**Degree of Automation**

The second dimension can be labeled as “degree of automation” with the end points of the continuum being called automated and controlled (or conscious) (Gagne et al., 1993). An automated process or procedure is one that consumes none or very few of the cognitive resources of the information-processing system. Controlled process, on the other hand, is knowledge that underlies deliberate thinking because it is under the conscious control of the thinker.

**IMPLICATION ON INFORMATION MODELING AND METHOD ENGINEERING**

Researchers develop methods, and methods can be reengineered. By contrast, we cannot change the design of human information-processing system. Although the human

### Table 1: A production system to understand an ER diagram

<table>
<thead>
<tr>
<th>P1</th>
<th>IF</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Goal is to understand ER diagram</td>
<td>Set subgoal of identifying meaningful chunks of information in ER diagram</td>
</tr>
<tr>
<td>P2</td>
<td>IF</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>Subgoal is to identify meaningful chunks of information in ER diagram</td>
<td>Set subgoal of identifying entity in ER diagram and set subgoal of identifying relationship in ER diagram</td>
</tr>
<tr>
<td>P3</td>
<td>IF</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>Subgoal is to identify entity in ER diagram and symbol is a rectangle</td>
<td>Symbol represents an entity</td>
</tr>
<tr>
<td>P4</td>
<td>IF</td>
<td>THEN</td>
</tr>
<tr>
<td></td>
<td>Subgoal is to identify relationship in ER diagram and symbol is a diamond</td>
<td>Symbol represents a relationship</td>
</tr>
</tbody>
</table>
subsystem is intelligent and adaptive, we cannot change the basic properties that define its strengths and weaknesses. If an information model is to be easy to understand and to function as an effective communication tool, then the information-modeling method must be compatible with our information-processing characteristics. It is, therefore, important for us to consider this constraint when engineering methods and modeling information.

**Limitation of Working Memory**

The magic number $7 + 2$ has important implications on information modeling and method engineering. Firstly, if there are more than seven chunks of information required to be absorbed by the readers at any one time, the working memory capacity might be exceeded, which means that some information might not be acquired. This is consistent with the recommendations by researchers and practitioners (e.g., Hawryszkiewycz, 1991) that there should be no more than seven processes on a data-flow diagram. If this is true, some sort of leveling technique, similar to the one employed by data-flow diagram, might be needed to limit the information to a reasonable amount at any one point in time. Alternatively, the information model should be designed and laid out in such a way that at any time no more than seven pieces of information need to be processed together.

Secondly, if an information-modeling method has more than seven modeling constructs, cognitive overload might occur. For instance, it would be difficult for a casual user to remember what each of the construct means if there are more than seven of them. The capacity of working memory serves as a threshold on the number of modeling constructs that can be incorporated into a modeling method. As such, the complexity of Unified Modeling Language (UML) and the number of different diagrams used in UML are causes for concern.

**Declarative Knowledge**

Declarative knowledge deals with facts. With respect to method engineering, declarative knowledge will consist of facts about the modeling constructs—what they are and what they represent. Since declarative knowledge is one type of long-term memory, the larger the number of constructs in a modeling method, the more time is required to learn them. Training time is something that end-users are very reluctant to invest. One of the reasons for the popularity of entity-relationship (ER) and object-oriented (OO) approaches is that a very small number of constructs is involved and that results in their simplicity. Also, using constructs that tap into existing declarative knowledge facilitates the transfer of knowledge and reduces the training time. For example, many researchers and practitioners claimed that entity-relationship and object-oriented approaches are intuitive and natural. Although research results vary, the constructs used by entity-relationship and object-oriented are undeniably simpler than a modeling method based on algebra or predicate logic, especially from the end-users’ perspective.

**Procedural Knowledge**

Procedural knowledge is knowledge about how to do something. This is one of the most problematic areas in information modeling. For example, the most common criticism of the object-oriented approach is the difficulty in identifying objects (e.g., Wand & Woo, 1993). The fuzziness of constructs is also a problem with entity-relationship modeling where one is often not sure when to use relationship, attribute, or even entity to represent something in the real world. For example, Goldstein and Storey (1990) found that users of an automated database design tool had difficulty distinguishing between relationships and attributes. Codd
(1990) wrote “one person’s entity is another person’s relationship.” It is, therefore, vital that when engineering methods, we need to precisely define the constructs and specify when and how to use a construct. Saying that the world is made up of objects does not help the analysts or the end-users in information modeling. Metamodelling, which describes the procedural and representational aspects of modeling methods, is a good way of documenting the procedural knowledge of a method. Forcing method engineers to perform metamodelling ensures that they think through and sort out the details involved in using a construct.

**Domain-Specific Versus Domain-General Knowledge**

Research has shown that experts in a specific domain have more and better conceptual or functional understanding of the domain, automated basic skills in the domain, and domain-specific problem-solving strategies. Domain experts, in contrast to novices, have the ability to perceive large meaningful patterns; highly procedural and goal oriented knowledge; less need for memory search and general processing; and specialized schema that drive performance. The possession of domain-specific knowledge, however, is a problem during information modeling. To facilitate end-users’ understanding of an information model, it is important to use intuitive constructs that the end-users can relate to and recall easily. This has been the argument put forth for the benefit of ER and OO approaches.

Another aspect that is related to method engineering is the advantage of using domain-general constructs in methods. Domain-general constructs facilitate the transfer of knowledge from one method to another. As the degree of overlap of the modeling constructs that underlie two methods increases, transfer also increases. Situation method, which is an information system development method tuned to the situation of the project at hand, might be a problem from this perspective unless it makes use of well-known and easy to understand modeling constructs.

**Degree of Automation**

Working memory limitation impacts end-users much more significantly than analysts. For analysts, the meaning of each construct is in the long-term memory, not the working memory. The knowledge has been internalized and automated by the analysts. Automated skills require little cognitive effort and allow the problem solver to perform necessary, routine mental operations without thinking much about them. On the other hand, remembering what each of the constructs stands for would be a controlled process for the end-users. They need to consciously and deliberately think about them. Controlled process requires cognitive effort and is subjected to the limitation of working memory. Thus, when engineering methods, we need to consider the effect on end-users who are not at the automated stage in using modeling methods and will probably never attain the automated stage. Modeling methods, which are convoluted and highly technical, might be an excellent tool for analysts at automatic stage, but will be a poor communication vehicle between analysts and end-users.

**CONCLUSION**

This research attempts to bring the wealth of knowledge in cognitive psychology to bear on the practical problems of information modeling and method engineering. The goal is to apply and adapt cognitive psychology theories and techniques for information modeling.
and method engineering research and help to span the gap between science and the practice of information modeling. In this chapter, we look at some cognitive psychology theories and a popular cognitive architecture, Adaptive Control of Thoughts, and discuss their implication on information modeling and method engineering.

ACKNOWLEDGMENT

This research is supported by a Research Grant-In-Aid funding from the University of Nebraska-Lincoln (LWT/06-185-92501).

REFERENCES


Section II

Refinement of the Contemporary Database Model and Technique
Chapter VII

A Case Study of the Use of the Viable System Model in the Organization of Software Development

Peter Kawalek
Manchester Business School, UK

David Wastell
University of Salford, UK

This chapter considers the usefulness of the Viable System Model (VSM) in the study of organizational adaptation. The VSM is a rigorous organizational model that was developed from the study of cybernetics and has been given considerable attention by management science research. The chapter presents a longitudinal case study that focuses upon a software development team. The VSM was useful in diagnosing the likely consequences of different organizational designs and in prescribing an alternative solution.

INTRODUCTION

This chapter focuses upon the issue of business adaptation, specifically describing a case of a software development enterprise. The theoretical basis for the analysis is drawn from systems theory, in particular the Viable System Model (Beer, 1979, 1985).

Systems theory has a broad relevance, as many different scientific disciplines grapple with their own issues of organization and adaptation (see Capra, 1996, for a useful synthesis). Information systems (IS) research has been particularly receptive to such “systems thinking” (e.g., Checkland, 1981; Checkland & Holwell, 1997), possibly as a result of its need to address people and technologies as an organized complexity. Recently, as managerial disciplines have sought to address the complex and turbulent conditions of global markets, systems thinking has once again begun to permeate popular business literature. Haeckel’s (1999) influential text is a good example. Amongst other things, he proposes that business should
seek “sense and respond” capability with strategy being “a design for an adaptive structure.” In this way, effective business management is secured through the effective modularization of the organization and the nurturing of a culture of commitment amongst the various units. Many well-established ideas, such as the need for powerful, top-down strategic plans, are redundant in Haeckel’s thesis.

Haeckel has presented a timely distillation and development of ideas that have their lineage in the broader halls of systems theory. This chapter draws from a similar heritage, but its focus is upon a particular branch of systems thinking, namely cybernetics. Cybernetics has been highly influential in the development of systems concepts across many different disciplines (Checkland, 1981) and continues to attract attention today (see again, Capra, 1996). Its focus is upon patterns of control and communication in systems of all kinds and is thus described as the science of organization (Beer, 1979, 1985). We are motivated by a general interest in developing cybernetic approaches to IS problems and projects. Earlier papers have described its use in software development and the development of groupware for a manufacturing company (see the related paper, Kawalek & Wastell, 1999). We propose that cybernetic models and theory may assist the study of patterns of communication in organization and make it possible to appraise the linkage between these communication patterns and the structure of the organization itself. Our particular focus is upon the Viable System Model (VSM). The VSM has been developed from cybernetic theory by Beer (1972, 1979, 1985) for application to human organizations in general. It has been given considerable attention by management scientists (e.g. Espejo & Harnden, 1989).

In this chapter, we are concerned with general organizational issues in relation to the software development process. The chapter will proceed as follows. First we will introduce the key elements of the VSM. An extended case study will then be presented in which the VSM is applied in a diagnostic fashion to understand the mutating patterns of organization structure in a large software production organization, Violet Computing. We will show how the organizational structure of VC, which was originally well-adapted to the circumstances of its business ecology, became increasingly dysfunctional as the environment became more turbulent and complex. The use of VSM, this time in prescriptive mode, will then be illustrated as a tool for designing a more appropriate organizational structure. We are therefore in alignment with the general issues motivating Haeckel’s work—that many businesses of many different kinds need to be designed to be adaptive. This concern with flexibility was itself a key motivation underpinning Beer’s development of the VSM.

**THE VIABLE SYSTEM MODEL**

Cybernetics is concerned with the identification and description of patterns of organization in complex entities. It has influenced the development of many different areas of research (e.g., the science of cognition, artificial intelligence, the study of ecosystems) and through ideas of self-organization continues to inform many debates today (Capra, 1996; pp. 51-71). The Viable System Model (VSM) is a complete organizational model that was developed from cybernetic theory by Beer. The rationale and features of the Viable System Model (VSM) are set out by Beer in a trilogy of books (1972, 1979, 1985). It is not intended that this chapter should serve as a tutorial of VSM; more modestly, our objective is only to highlight some of the main features of the VSM that are pertinent to the case study presentation later on. A fuller understanding of the VSM can be gained from a number of sources. The reader is referred to Beer’s highly individual texts and, in particular, to the second of the trilogy (Beer, 1979). There are, in addition, a number of other academic writings on the
subject (e.g., Espejo & Harnden, 1989; Flood & Jackson, 1991; Kawalek & Wastell, 1996; Kawalek & Wastell, 1999).

By viable system, Beer refers to a system (i.e. an organization of some kind) that is capable of maintaining separate existence, of surviving on its own (1979, p.113), in its environment. Organizations (at whatever level) are to be seen as systems (i.e., goal-directed entities made up of interacting parts, operating in an environment of some kind). The issue is: what form of internal “architecture” is required if these systems are to be viable?

Beer’s answer to this question is built upon a fundamental concept in cybernetic thinking, namely, the Law of Requisite Variety (Ashby, 1965). This law stipulates that the “variety” of the regulator must equal or exceed the variety of that which is being regulated (variety is defined as the number of possible states of whatever it is whose complexity we want to measure). It follows from this that organizations can be understood to be structures for handling variety. Therefore, the structure of an organization seeking to survive in a particular environment must be attuned to the variety of the environment. For example, as an environment becomes more complex so the organization will be required to adapt itself in order to manage this variety and to preserve its viability. This simple idea is depicted in Figure 1.

Thus far, the organization has been depicted as being responsive to a changing environment. In fact, the interaction between organization and environment is much more complex than this. Beer’s concepts of “variety attenuation” and “variety amplification” describe the patterns of adaptation whereby organizations seek to proactively manage their variety and that of their environment. Variety attenuation describes the process of reducing the number of states between an entity and a receiver (e.g., a hospital limits operations to the chronically sick so as to attenuate the variety of the population from its perspective). Variety amplification describes the process of increasing the number of states between an entity and its receiver (e.g., a hospital markets a new diabetic clinic, thereby amplifying its variety from the perspective of the local population). Therefore, fundamentally, the management of any organization is concerned with equating varieties through attenuation and amplification.

Beer proposes that viable systems be understood as recursive structures wherein cybernetic laws apply internally to the management of their operations as they do between the organization and its environment. At each level of analysis (i.e., “system-in-focus”) a viable system will consist of an operational system that performs productive work, and a meta-

Figure 1: An organization and its environment

![Diagram of an organization and its environment]

e.g. (1) local community & government agencies; (2) software consumers & quality certifiers

e.g. (1) local hospital; (2) software development team
system that is the means of regulating the operational system. The operational system is known as System One. Four further systems make up the rest of the VSM architecture. Each is briefly described below. The five sub-systems of the VSM are then displayed in Figure 2, using the diagrammatic conventions established by Beer.

- **System One** is composed of a collection of operational systems, each comprising an area of operational activity (operational unit) and the local management structure responsible for directly managing the operational unit (operational management).

- **System Two** provides a coordination service to System One. Without it, System One would be potentially unstable. System Two is the element that “dampens” the instability caused by conflict between parts of System One, and its sole function is anti-oscillatory.

- The role of **System Three** is to steer the organization towards its current objectives. It interprets the policy decisions of higher management and manages the operations of System One on a resources-results basis.

- **System Four** is an intelligence-gathering and reporting function that seeks useful information about the environment of the current system-in-focus. It provides a model of the organization and the environment which serves as a basis upon which hypotheses are explored and changes proposed to the organization as a whole (e.g., “move into new markets”).

- **System Five** sets policy. The values and beliefs espoused through System Five should be shared with all other elements of the organization. An important part of this role is thus to arbitrate between the functional emphasis on the status quo (System Three) and change (System Four). System Five should also be open to the other elements of the viable system, e.g., through mechanisms such as the algedonic signal, i.e., a special communication pathway which bypasses all the normal channels, reaching directly from System One to System Five. Its function is to alert System Five to a sudden crisis (e.g., “new product specification cannot be met”).

Carrying out a VSM study of an existing organization involves creating a mapping between the VSM architecture and the individuals/teams/departments that the modeller identifies in the organization. This mapping allows the modeller to identify omissions (i.e., threats to viability) in the organization and to critique its capability to carry out the different functions effectively. In this way, the use of the VSM can assist a diagnostic of organizational viability.

**A CASE STUDY OF VIOLET COMPUTING**

The following case study considers a well-established software development organization and its ongoing struggle to adapt to an increasingly turbulent business environment. The changes to the environment affected the way the team was organized and the development process that they deployed. The subject of our study is Violet Computing (VC), a large operating systems development team of over fifty staff. VC has achieved a reputation for high quality. Its product, the Violet Operating System (VOS) is well known and highly regarded in its market sector. Its efforts to engender a culture of quality amongst staff had gained recognition through a number of prestigious quality awards.

The organization of VC has been rationalized and revised over a number of years and at the time of the study had reached a particularly critical juncture. A further set of change
initiatives had been proposed for VC by its senior management and managers from its parent company. These changes were contentious and had provoked some opposition from within VC. As well as being concerned about the changes themselves, some team members were concerned that VC was seemingly in a perpetual state of flux. If the process continues long enough, they argued, the organization will return to the models of operation that it deployed ten years ago. Clearly, there was a need for a more formal approach to organizational development. The team needed a framework through which it could reason and learn about its models of operation and their relation to the environmental characteristics of the time.

Our study sought to assess the likely impact of the proposed changes and to identify whether they were likely to have any implications for the ability of the organization to continue to successfully develop software. In doing this we undertook a retrospective study in order to describe the organization of VC in its formative years and to contrast the findings with the present day structure. This investigation of the organizational genealogy of VC proved to
be a fruitful tactic, for it shed light on a pattern of events whereby the organization had reacted tactically to ongoing environmental changes. At each stage in its development, VC had sought to maximize its viability. However, in responding tactically to changes in the environment, VC continued to run the risk of making changes that were detrimental to its well-being in the long term. A summary of the pattern of changes is given in Table 1. The following narrative will help to clarify both the different positions described and the terminology used.

The Traditional Organization of VC

In its early years, VC was organized around a “technical commission,” some additional management functions and a series of function teams—each of which was responsible for the development of a defined part of the VOS product. The most prominent feature of this arrangement was the role played by the technical commission. This entity, headed by a single individual, was made up of a number of team members who were selected for their technical and personal skills. The principal goal of the technical commission was to maximize product quality and integrity. In the language of the VSM, the commission dominated System Three, Four and Five functionality and to a great extent made long-term technical values preordainant. The prime responsibility of the commission was to prepare and update the functional specification for the product. This was done through the monitoring of technical possibilities and requirements (System Four), the evaluation of their potential benefit vis à vis current practice (Systems Three and Four) and the arbitration of conflicts between the two (System Five). Control of the functional specification certainly afforded the technical commission considerable power in the organization, however, the VSM exercise suggested that the uncompromising way in which the commission was able to wield this power actually depended upon a transitory set of environmental conditions that existed at that time.

In order to ensure that its technical standards were adhered to, the commission took a strategic role in the organization by providing the interface between the business environment (i.e., customers, salesmen, etc.) and the function teams (i.e., detailed designers, programmers, testers). From this vantage point, the technical commission was able to work

Table 1: Different positions and terminology

<table>
<thead>
<tr>
<th>Environmental Conditions</th>
<th>Organization and development method</th>
<th>Threats to continued viability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lower variety (less competition, less customer knowledge, greater customer loyalty)</td>
<td>Powerful meta-system control. Rigid development process.</td>
<td>Increasing variety of environment (increasing competition, increasing customer knowledge, decreasing customer loyalty)</td>
</tr>
<tr>
<td>Mid-position</td>
<td>Mid-position.</td>
<td>Pathological autopoiesis</td>
</tr>
<tr>
<td>Higher variety (high competition, high customer awareness, less customer loyalty)</td>
<td>Little meta-system control. Looser process.</td>
<td>Danger of loss of control. NB. Loss of holistic management of technical issues</td>
</tr>
</tbody>
</table>
in two interdependent ways (illustrated in Figures 3 and 4). First, it was able to provide an environment of relative stability for the function teams by insulating them from the worst fluctuations of market demands (e.g., changing requirements, uncertainty of customer expectation). This made it easier for the function teams to work to the specification and to conform to a rigidly prescribed development process that was laid down by the technical commission. In this way, the function teams were encouraged to focus only upon the tasks required of them to achieve high quality. This was particularly important in the early phases of VOS development when the product was less well understood by the team and greater effort was needed in order to ensure that it functioned correctly.

Secondly, the technical commission also acted as a filter that blocked those market requirements that threatened to undermine technical strategy. This was done by putting an alternative technical case to potential customers or even rejecting some demands. This role was crucial. It meant that as the technical commission was able to attenuate market variety from the point of view of the function teams, so too it was able to amplify the technical variety from the point of view of market. However, the VSM study suggested that their ability to provide this amplification was only possible because of some temporary features of the business environment. In particular, in the market segment occupied by VC, there was little competition at this time. For many purchasers there was actually no practicable alternative to the Violet system and many of these purchasers were relatively uninformed about technical possibilities. These conditions enabled the technical commission to act as a powerful advocate to customers. It was able to postpone or proscribe some customer requirements and, using its superior technical knowledge, the commission could argue that the sort of product that the customers really wanted was the sort of product that VC was building already.

Level one of the VSM created for the retrospective study is shown below at Figure 5. The technical commission is shown throughout the meta-system of System 3, 4 and 5. Four function teams are shown. These were each responsible for specific areas of product development (e.g., kernel, comms., peripherals etc.). The reader should refer to Figure 2 for a key to the primitives of a VSM diagram.

Since these early days of VOS development, there has been a series of ongoing changes to VC. These are concomitant with changes to its business environment. Competition and technical awareness amongst customers have been increasing whilst product loyalty has shown a marked decrease. Although our study did not subject this intervening period of VOS development to a formal VSM analysis, anecdotal accounts supplied by team members described how VC continually sought to adapt itself to its environment. Although the effect of these adaptations was not always consistent, over a period of time the technical commission was reduced in size and influence. Instead, greater authority was given to the function teams through the reformulation of the resource bargain between System Three and System One. The function teams were allowed to be more directly involved with the customers of VC and, in this way, could be more responsive to the business problems and issues of their clients. In order to achieve this, the function teams exploited their new authority to make some adaptations to the development method in order to make it more flexible and better suited to the tailoring of products for the new market-place. Thus, the traditional model of organization in VC was eventually usurped by a new design that shifted power from the technical commission to the function teams. However, pressure for change remained and over time VC discovered that it had solved one set of problems only to find itself in a new set of difficulties which we will now discuss in detail.
Figure 3: The technical commission’s role of attenuation

**Attenuation of environment to VC**

- **The Environment**
  - Customers
  - Product & Marketing
  - Salesmen

- **The VC Team**
  - Development by Function Teams
  - Technical Commission (TC)
  - Validation by Function Teams

**Uncertainty of environment is threatening to function teams seeking to develop and validate the product.**

**Major variety attenuation.**
TC interfaces to the environment and filters out excessive variety (e.g., by refusing some customer requirements).

**Relative stability in the development sub-system allows technical teams to focus on the task in hand.**

---

Figure 4: The technical commission’s roles of amplification

**Amplification of VC to the environment**

- **The Environment**
  - Customers
  - Product & Marketing
  - Salesmen

- **Technical Commission (TC)**

**Variety Amplification.**
TC interacts with the environment in order to shape its expectations of VOS.
The Current Organization of VC and Proposed Changes

The recent review that VC undertook of its organizational structure was motivated by two principal concerns. The first was the wish of senior management to continue the improvements in customer responsiveness in the face of an ever more demanding product market. The second concern was that the number of product enhancements should be controlled. This latter concern was especially acute. Despite the maturity of the product, even
the best understood and most proven product functions continued to undergo a greater number of enhancements than had been anticipated. This was felt to endanger product quality and cost. Using the VSM we were quickly able to identify the reason why this was the case.

As the technical commission had lost power, the function teams had gained the authority to propose enhancements to their part of the product. This enabled each function team to ensure that its part of the product was able to satisfy renewed technical and market expectations. The remaining powers of the technical commission were shared amongst a strategy function and a VOS product manager (“VOS Manager” see Figure 6). The responsibility for arbitrating over which of the enhancements proposed by the function teams should be given priority/resource was clearly vital. In practice, however, such meta-systemic arbitration was nearly impossible. The degree of specialization needed by each of the function teams, the fact that the VOS product was very mature, and the fact that the function teams had access to customers served to promote an internal technocracy within VC. If a function team argued that it needed to enhance its part of the product, the management was unable to launch an effective counter-argument to it. Thus, the function teams were now highly influential, both over the day-to-day attainment of quality and the direction of longer term product strategy. It became obvious that these teams were exploiting this new power to propose enhancements in order to generate work for themselves. In other words, even where there was insufficient technical or market justification, the function teams still argued that their part of the product should be enhanced.

This is an example of what Beer regards as “pathological autopoiesis” (Beer, 1979; p.412). It was in the interests of the function teams to behave in this way so that they were able to secure resources from management and, ultimately, to ensure that the team continued to exist. Figure 6 summarizes the organizational structure extant at the onset of our study. Note that the technical commission’s domination of the meta-system has been replaced by a senior management function, a strategy and business intelligence function, and a VOS Manager.

At the time of our study, the self-serving behavior of the function teams was becoming a matter of increasing concern to senior management in VC. A strategy was being implemented to strike radically at the root of the problem by reducing the power of the function teams whilst also dispensing with the final legacy of the technical commission (i.e., the notional power of arbitration held by the meta-system). This was to be achieved by putting greater emphasis on a project structure of organization. This meant that in the future, resources would be allocated only in response to customer requirements that were grouped and organized into defined work packages (i.e., projects). Although the function teams would continue to exist and now would have sole responsibility for the technical standards within their remit, critically, they would lose their power to argue for product enhancements on technical criteria alone.

These changes were part of the program of organizational change that was under way as our VSM study was carried out. Although many parts of the program were contentious, the VSM diagnostic helped to reveal that they were a rational response to changing market conditions and, as a result, we were generally supportive of them. However, we were concerned about the change to a project structure of organization. Of particular issue was the fact there would now be no meta-systemic ownership of technical issues. The management at this level of concern was to be focused only on market issues. As a result, it would be unable to arbitrate between conflicting objectives at the technical level and, in the face of sudden market opportunities or threats, it would be unable to organize any radical reworking of the product. This shortcoming was felt to threaten organizational viability.
The rational analysis facilitated by the VSM revealed the precarious nature of this new arrangement. The ownership of technical issues was entrusted to the function teams. However, none of these had a holistic view across different parts of the product and all were prohibited from proposing enhancements on technical criteria alone. Clearly then, the market would become the sole source of technical enhancements and yet, if ever these enhancements were wide-ranging, there was no mechanism to provide holistic management of them.
Product quality would be endangered by this. At the same time VC would have no mechanisms for promoting enhancements on technical criteria or, perhaps still more critically, in *anticipation* of market benefit. Thus, technical management would be fragmented and reactive.

**DISCUSSION AND CONCLUSIONS**

The history of VC that we have recounted is an interesting and instructive one. In essence it can be seen as a struggle for appropriate organizational structure in the face of changes in environmental uncertainty and complexity. The narrative clearly reveals that as the ecological setting of VC changed so it was necessary to change the organizational structure. This in turn had ramifications for the way VC carried out projects. Initially, VC had operated with the technical commission acting as a filter between the customers and the function teams. At this stage the function teams conformed to the specification developed by the technical commission and followed a rigid development route. As customers developed a higher variety of demands and as competition intensified, VC found it necessary to empower the function teams and to disengage the technical commission’s role as a “variety attenuator.” Ultimately, VC proposed to increase the accountability of the function teams by allocating resource solely on the basis of customer requirements.

The VSM provides important insights into the way in which businesses can design adaptive organizational structures. There are clear parallels with the newer body of work that shares this emphasis (we choose Haeckel as a particularly prominent example). In drawing attention to the influence of environment on organizational form, we would also cite a long tradition in organizational theory. The well-known work of Lawrence and Lorsch (1967) has been instrumental in establishing the critical linkage between environmental turbulence and organizational structure. Their work has highlighted the need to strike the right mix of internal differentiation and integration in relation to the stability/flux of the external environment. They contend that a more turbulent business environment requires greater organizational differentiation (i.e., the division of tasks amongst a greater number and/or variety or organizational units). In cybernetic terms this correlates to Ashby’s basic thesis that the variety of the regulator must equal or exceed the variety of the regulated. But as the degree of internal differentiation increases, there emerges a need for more sophisticated integrative mechanisms to maintain internal coherence.

In this light we can see the demise of the technical commission of VC as consistent with a trend towards greater differentiation (i.e., tasks became divided amongst the function teams). The proposed move to a project structure continues this trend, i.e., tasks allocated to function teams are in effect sub-divided again amongst projects. However, no new integrative structures were established to ensure coordination and consistency in this increasingly fluid and chaotic internal milieu. This was the crux of our concern.

Our historical analysis has demonstrated the retrospective power of the VSM for understanding detailed changes in the organizational structure of (software) enterprises as they struggle to sustain viability in a potentially predatory environment. But we require more of VSM than description and interpretation. We wish to use the model prescriptively as a meta-organizational tool enabling the design of more effective software processes and structures. To this end, the reader will recall that our study sought to assess the likely impact of the proposed changes and to identify whether they were likely to have any implications for the ability of the organization to continue to successfully develop software. Whilst we
could be generally supportive of the reasoning behind the changes, the VSM analysis did suggest that VC was once again in danger of replacing one threat to viability (i.e., pathological autopoiesis) with another (i.e., loss of control of technical issues).

Clearly, VC needed to devise a way of ensuring that the function teams have sufficient power whilst preventing them from becoming too powerful and promoting product enhancements compromising its overall technical integrity. The VSM study indicates that to dispense completely with meta-systemic arbitration of product enhancements could be seriously detrimental. VSM goes on to provide the general outline of a solution to this problem. We saw the locus of the solution to lie in the interaction between Systems One and Three (i.e., the nature of the resource bargain). Some ownership of technical issues by the metasystem is clearly required. The proposed solution is illustrated in Figure 7, which indicates two mechanisms for the distribution of resources to the function teams: either on the basis of customer requirements (the project stream) or technical issues (the technical stream).

Figure 7 indicates that the balance between these streams should be determined by System Three acting in concert with the demands of Systems Four and Five. By sharing the responsibility for meta-systemic control of technical issues amongst the function team leaders and other technically expert staff, this arbitration role should be afforded sufficient power without becoming alien to the function teams. We further recommended that this two stream approach should be facilitated by a resource pool acting in a System Two capacity to attach staff to function teams as needs arose. Through this, each function team could be reduced to nominal status when there was no work to be done in that area. At other times, the function team could be reactivated and staff placed within it. This design was felt to address the need for management of technical strategy whilst avoiding the pitfall of “pathological autopoiesis.”

In summary, the case history of VC recounted in terms of the VSM cogently demonstrates in a software context the descriptive and forensic power of the model for understanding patterns of adaptation of organizational structure to environmental conditions, and for identifying weaknesses that threaten continuing viability. We have also seen the VSM in action as a tool for organizational redesign; the concrete proposals that we articulated have now been presented to senior managers in VC and a further review of the organizational structure is now planned. Our concern that a model of twin-sponsorship of projects be established (Figure 7) amounts to a proposal that a new integrative mechanism be established as a counter-weight to the trend towards increased differentiation. This is a point of important general significance. It allows us to understand the proliferation of methodological diversity as an augmentation of internal variety being applied to an increasingly complex and demanding environment. It goes on to underline that such diversity must be balanced by new integrative structures to safeguard technical integrity.

VC typifies the situation faced by many organizations in many sectors; customers are becoming more sophisticated, the range and diversity of demands are increasing, and competition from rivals is intensifying. This increase in external variety can be met in two ways, either by attempting to attenuate it (for instance, by sticking to one immutable product—a common but doomed strategy in the longer term) or through an increase in internal variety, as VC has done. The VSM emphasizes that this must be counter-balanced by increased attention to more sophisticated integrative mechanisms. We believe, therefore, that the VSM contributes to the current discourse on business flexibility in general. It provides a detailed architectural model of organizational structure, based on a systematic set of cybernetic principles that can be applied at any level of abstraction as a tool for organizational modeling, analysis and design.
Figure 7: A model of twin sponsorship of product development

Key

Financial resources

Manpower resources

ENDNOTE

1 Violet Computing (VC) and Violet Operating System (VOS) are pseudonyms

REFERENCES

Chapter VIII

Modeling of Business Rules For Active Database Application Specification

Youssef Amghar, Madjid Meziane and André Flory
The National Institute of Applied Sciences, France

Active database applications require the classic cycle of analysis, design, prototyping and implementation. During analysis and design steps of the information system engineering process, modeling behavior is an important task. This task is both essential and crucial when information system is centered on active databases, which allow the replacement of parts of application programs with active rules. For that reason, the specification of business rules during analysis and design steps becomes an actual requirement. Business rules ensure the well-functioning of information system. They are descriptive (integrity constraints) or functional (derivation rules and active rules). To relieve programmers from using either traditional or ad hoc techniques to design active databases, it is necessary to develop new techniques to model business rules. These techniques have to enhance the specification of dynamic aspect through a high-level description language able to express precisely and completely rule semantic. In this chapter, we propose a uniform approach to model business rules (active rules, integrity constraints, etc.). To improve the behavior specification we extend the state diagrams that are widely used for dynamic modeling. This extension is a transformation of state transitions according to rule semantics. In addition, we outline new functionalities of Computer-Aided System Engineering (CASE) to take into consideration the active database specificities. In this way, the designer can be assisted to control, maintain and reuse a set of rules.

INTRODUCTION

To allow designers to exploit the specificities and features of active databases, it is important to build prototyping and monitoring tools to assist the designer during the design and development stages. This kind of tool offers indicators about choice relevancy and writing of rules. An active database management system (active DBMS) is an extension of
a passive (relational or object) DBMS by adding trigger mechanisms. The notion of trigger appeared in the seventies, and has been generalized to the notion of active rule that is based on the Event-Condition-Action (ECA) formalism. The semantics of an ECA rule is as follows: when an event E is produced, if the condition C is satisfied, then the action A is executed. Actions are initiated by the DBMS when appropriate events occur, independently of external requests. These rules allow database designers to specify the active behavior of a database application that provides the enforcement of database integrity.

Current design methods for information systems do not consider explicitly rules at the design level. In systemic methods such as “Structured Analysis Design Technique” (SADT) (Yourdon, 1979), rules are considered as a part of the design process but they are not modeled explicitly. In Object-Oriented methods such as the Object Modeling Technique (OMT) (Rumbaugh, Blaha, Lorensen, Eddy, & Premerlani, 1991), Object-Oriented Analysis (OOA) (Booch, 1994) or methods built upon unified process using unified modeling language (UML), rules are partially represented in dynamic models, particularly in state diagrams. Moreover, at the development level, rules are often coded in the application programs implying a hard maintenance of business rules. These methods are generally supported by CASE to facilitate the specification of applications. However, these tools present some insufficiencies regarding the requirements of business rules that need to be specified at a high level of description as well as at a logical level for a more detailed specification.

In the literature, several approaches were proposed to integrate active concepts into databases. For most systems, the knowledge model is based on ECA rules and the execution model of the nested transaction model, which authorizes different coupling modes (immediate, separate, deferred). Other systems use a weakened version of ECA rules. Furthermore, a number of research projects on active databases have focused on the rules’ management and their evaluation. Several commercial DBMS include event/trigger mechanism proposed initially by Kotz, Dittrich, and Mulle (1988), such as the Postgres rule system (Stonebraker, Jhingran, Goh, & Potamianos, 1990), Starburst’s production and alert rules (Lohman, Lindsay, Prahesh, & Schiefer, 1991), Ariel’s production rule system (Hanson, 1989), the ECA model of HiPAC (Dayal et al., 1988), and the event-action (EA) model of Ode (Gehani, Jagadish, & Shmueli, 1992). In addition, there is a general agreement to consider that the new generation of DBMS systems would include active capabilities (Buchmann, 1993; Campin & William, 1997; Diaz & Jaime, 1997) to support non-conventional applications such as documentation, geographic systems, workflow, and project management.

The design issue of active database applications is known as one of the most open research problems. Indeed, to design active database applications, programmers use either traditional or ad hoc techniques, which increase the complexity of applications by forcing the user to defer several modeling decisions concerning the active behavior to the development stage.

To gain benefits of active database capabilities, new approaches require inclusion of rules during both analysis and design stages. Few researchers have addressed the conceptual specification of behavioral aspects of applications independently from any active DBMS. To our knowledge, only IDEA (Ceri & Manthey, 1993; Ceri & Fraternali, 1997) and SORAC (Peckham, MacKeller, & Doherty, 1995) projects have treated the design of active database. However, IDEA methodology is strongly linked to Chimera, a DBMS developed especially in the framework of IDEA project. In IDEA project, any rules’ specification is proposed. Rules, identified from requirement analysis, are directly expressed to the syntax of Chimera. The SORAC model permits the schema designer to specify enforcement rules that maintain constraints on object and relationships to facilitate the task of the designer. With active
databases, it is interesting to replace a part of the application programs by using active rules that are stored, detected and executed by the DBMS. This technique called knowledge independence allows a database to evolve by managing the rules without altering the program code. But a main problem exists regarding the design of such databases. More precisely, the design methods of information systems do not offer satisfying means to describe active rules that represent active behavior of applications. It is necessary to improve the techniques of active behavior description whether at the designing or derivation level.

Our work presents a framework for designing applications for which active behavior needs to be correctly modeled, particularly applications focused on active databases. The main contributions of this chapter include:
1. Specification of system behavior at a high level of description. This specification improves the modeling of system behavior with the introduction of the ECA rules’ semantics.
2. Derivation process of active behavior specification for relational DBMSs.
3. Definition of an environment for active database design.

This chapter is organized as follows: Section Dynamic Description briefly discusses the techniques currently used to represent the business rules within data models. Section Specification of System Behavior proposes to model the behavior of an active object through a specification based on a description language developed initially for telephone switching systems. Section Case Study presents a case study to support our explanations. Section Derivation Process describes the derivation process of models elaborated in the design stage for a relational schema. Section Rule Modularization presents a strategy for modularizing a set of rules in order to facilitate the rules’ maintenance. This section also presents some features of a prototyping tool. Section Environment for Active Databases describes the original functionality of the architecture of an environment for active databases. The last section concludes this work.

TECHNIQUES FOR MODELING BEHAVIOR

Business rules represent essential elements in the activity paradigm. They may be expressed using different modeling techniques including data-flow and state transition diagrams or Petri-net graphs. Each of these techniques is briefly described. Concerning the modeling approaches, business rules appear through the data and behavior models.

Specification Using Data-Flow Diagrams

Data-flow diagrams (DFD) are utilized to describe data flow between processes or between a process and an external unit or stored data. The action component of the ECA structure corresponds to a process in the DFD framework. Initially, these diagrams (Yourdon & Constantine, 1979) could not represent control and temporal events as data flow. Moreover, DFDs do not allow the representation of the synchronization conditions for process execution. In order to avoid these drawbacks several extensions were proposed to model synchronization and temporal aspects in DFD (Eddins, Crosslin, & Sutherland, 1991).

Specification Through State Transition Diagrams

State-transition diagrams (Davis, 1988; Muck, 1994) allow the representation of the behavioral requirements in terms of states and transitions. Although there are not special symbols to represent ECA rules, all types of rules can theoretically be represented. These diagrams describe inter-dependencies among rules well, but rule effects over objects are not
visible. State-transition diagrams do not offer explicit symbols for all components of a business rule even in some techniques such as OMT (Rumbaugh et al., 1991). Though transitions can be labeled with events, conditions and actions, the diagramming notation doesn’t support labeling of rules.

**Specification With Petri Nets**

Petri nets (PN) are widely used to describe and simulate system behavior (Peterson, 1986). There are many kinds of PN, such as autonomous, non-autonomous, with predicates, colored, generalized, with priorities, temporized, etc. Sinkless, conflict-free, safe and reachability are properties that characterize PN.

A PN is a bipartite graph in which there is a succession of places and transitions in a path constituted with successive edges. To model business rules, places are used to represent events and/or conditions, and transitions represent actions. Gatzui, Geppert, and Dittrich (1993) proposed the use of PN extensions to describe elementary or complex events. Extended PNs use labeled edges to model alternative results depending on the condition clause. Using these extensions, it is possible to model the structure of all business rules.

The drawback to using PNs, however, is the complexity associated with their specification. The PN doesn’t help the user understand the complexity of system behavior.

**Dynamics Within the Extended ER Model**

In the Entity-Relationship (ER) model, rules are directly expressed through the semantics of the model inclusive of relationships, cardinalities, and existential dependencies. The ER model does not allow the designer to formulate separate events, conditions and actions. A rule in this model can only represent an integrity constraint. Transitions and active rules cannot be modeled. Note that Nijssen and Halpin’s (1989) NIAM model, a derived ER model, models integrity constraints with constructors representing uniqueness, optional, and mandatory constraints. Uniqueness constraints correspond to primary keys of a relational database.

Among the ER extensions to embed the database application behavior, we can cite: *Entity-Relationship-Event-Rule (ER²)* is an ER model that includes events and rules (Tanaka, 1992). In this model, events and rules are objects. Relationships between events are represented by precedence relationships, links between rules describe priorities, and relationships between events and rules symbolize triggering relationships. Another association allows the designer to link an event to an entity, this expressing that the entity can be affected by the event. The process model associated with $ER^2$ uses a colored PN to model flow control between processes.

The advantages of this model are the expressiveness of all rules triggered by database operations, and the ability to describe the relations between rules and objects. The drawbacks of this model are ii) the sequence of rule invocations is not considered, and the events that are included in the model concern only database operations.

The model of Dows, Clare, and Coe (1992) is an ER model based on the entity life cycle. This approach focuses on the expression of state changes of entities during their life cycle. The main idea of this model is to describe all events attached to state changes; this concerns all events from object creation to object deletion. This model uses a tree structure representation of which the root corresponds to an entity type and terminal nodes correspond to events affecting this entity type. Intermediate nodes may exist to describe selection and iteration. In this model, conditions are not expressed; that is, each event implies an
unconditional execution of associated operations. One of the model’s strengths is the description of transition rules. However, it has a major limitation of not modeling the effects of a rule over an entity.

**Behavior Specification within the Object-Oriented Context**

The object-oriented design requires the utilization of three models: 1) a static (or data) model to represent classes and their relationships, 2) a dynamic model to represent the behavior of each object type, and 3) an interaction model for message flow between objects. Business rules may be identified at each level in these models according to heuristic and organizational knowledge that may differ from one organization to another. Previous work concerning the extension of object-oriented methods has introduced active behavior during the modeling stage (Bichler, & Schrefl, 1994; Loucopoulos, Theodoulidis, & Pantazis, 1991).

**Business rules within the data model.** For each object class, the object model allows for the definition of attributes as well as associations for static aspects and methods for dynamic aspects. Attributes can be viewed as object states and methods as means to define state changes. Moreover, integrity constraints may be expressed at this level to support data coherence by restricting object states as well as dynamic constraints by restricting state transitions. These constraints are a subset of the business rules.

**Business rules within the dynamic model.** The dynamic model specifies object life cycles through a transition diagram or its derived diagrams (Harel, 1988). Such diagrams implicitly describe business rules. In particular, whenever an object receives a message, it decides, according to its current state, if it is ready to respond. The object executes associated methods to change its state according to its state diagram. It is noted that these diagrams are used to model the behavior of only one class of objects.

**Business rules within the interaction model.** The interaction model describes messages between objects for a given activity or use case. A popular representation technique for message exchange is the interaction diagram, which specifies an interaction sequence and messages sent by a sender to several receivers. Business rules within this type of model concern message ordering, receiver selection, synchronization modes, and temporal constraints (Booch, 1994; Embley, Kurtz, & Woodfield, 1992; Liddle, 1992).

Lists of messages and receiver selection determine the message to transmit and the object that receives it. The synchronization mode defines how objects involved in interactions are synchronized, while temporal constraints specify beginning, delay and end of interactions. Thus, an interaction model expresses the global behavior of a system, showing the exchanged messages between application objects.

**Discussion**

An information system can be described through two components: static and dynamic. Business rules that support theses two components are classified into two main categories: integrity constraints to control the static aspect, and active rules to describe the dynamic aspects. The behavior representation techniques are used to represent the dynamic part of the information system.

All of these techniques dedicated to behavioral specifications present insufficiencies to describe the ECA semantic. The extension of the ER model permits the specification of integrity constraints at the model level, but it is hardly exploitable for behavior description. Although object-oriented methodologies allow the designer to describe the two aspects of the information system, the modeling of dynamic aspects uses several different formalisms
to elaborate system specifications. We propose to specify, in a uniform way, dynamic aspects (integrity constraints and active rules) through a formalism that is characterized by uniformity and derivability properties.

**SPECIFICATION OF SYSTEM BEHAVIOR**

**Meta-Model**

Business rules ensure the proper functioning of an information system. They are descriptive in nature (integrity constraints) or functional (derivation rules, active rules). Independent from the power of marketed DBMS, the rule’s description at a conceptual level offers several advantages, such as 1) the independence between the specification and the implementation, 2) the definition of a user-oriented model (Figure 1), 3) the representation of rules in a uniform way. Classic examples of business rules are trading rules for bonds, inventory control rules, and configuration management rules.

In order to enhance the application modeling, it is necessary to specify business rules at conceptual level. Business rules are activated by events corresponding either to object state changes, temporal events or application decisions. Application behavior is then specified at two levels—The local behavior allowing an object to react when events are raised and global behavior giving the rule chaining to reach a goal. Object behavior corresponds to a task executed when appropriate event (or message) is raised. To express object coherence, integrity constraints can be stated. The role concerns permits to attach integrity constraints to an object class. The object behavior is submitted to business rules that can be either active or derived rules.

Rules may share the same triggering event. Thus, an occurrence of such an event may trigger the execution of several rules. To determine an execution order, a mechanism is necessary to avoid conflicts between rules. To avoid these conflicts, the user defines an order relation between rules. More precisions are given in section Rules Ordering and Rules Priorities.

**High-Level Language Description**

The distinction between active and traditional database designs focuses on the active behavior component that can be expressed by the ECA paradigm to represent business rules.

*Figure 1: Rule-oriented model for generating active databases*
Active behavior is described in terms of rules defined as a set of conventions that must be respected by the system. Business rules ensure correct system behavior. They are either descriptive (integrity constraints) or functional (derivation rules, active rules) rules.

The actual information systems modeling tools do not permit a high-level specification of business rules, except for integrity constraints that are typically taken into account. Though there are currently specification languages for constraint (or rules) modeling, there is little if no interest in their use due to their lack of abstraction capabilities and the complexity of their underlying formalisms.

The work presented in this chapter, in keeping with the general framework of the object-oriented approach, proposes a meaningful expression of business rules that can be readily used by designers. Our proposed framework offers the same support as provided by popular DBMS tools. In addition, it offers several advantages, primarily at the conceptual level of information systems modeling. These advantages include:

1. The independence between the model definition and its implementation. Indeed, the rules’ specification is a high-level abstraction.
2. The definition of a user-oriented model. Business rules are widely included in user requirements.
3. The uniform representation of rules. All types of rules are described with the same formalism.

The behavior of a system is expressed in terms of its global behavior and at a more granular object level. Thus, it is necessary to represent system behavior for each of these levels.

The local level allows us to specify the correct behavior of each object. Local modeling focuses on understanding how objects operate inclusive of events that impact the behavior of the object.

The global level specifies how objects communicate with one another. This requires the identification of global rules (or interaction rules) that represent functional behavior at the system level.

CASE STUDY

We present a case study to illustrate the chapter’s propositions. Let us consider an example of a document-loan management system. In this example, reservation, loan and loan prolongation are managed. Some rules are imposed by the application to fix the number of loans and loan prolongations. Two activities are identified: document loaning and document restitution. We delimit the application schema to entities described in Figure 2. The two main entities are user and document. The three relationships indicate the links between a user and a document. Indeed, a document can be borrowed or reserved. Let us note that a user can prolong a loan under certain conditions.

Relational Schema

Two main entities constitute the relational schema: DOCUMENT and USER. Each document is described by the following attributes: id-doc, the reference of the document; doc-name, the document name; state, the availability of the document in the document center; nb-copies, the number of copies of the document; and nb-loans, the loan number of the document. A reference (id-user) and his name (username) describe each user. Each loan concerns one user (id-user) and one document (id-doc) and begins at b-l-date and would finish at e-l-date. The document is effectively returned at ret-date. Each reservation,
concerning one user (id-user) and one document (id-doc), begins at b-reservation-date and finishes at e-reservation-date. The relational schema is comprised of the following relations and their respective attributes:

- Document(#id-doc,doc-name,state,nb-copies,nb-borrow)
- User(#id-user,name)
- Borrowing(#id-doc,#id-user,b-l-date,e-l-date,ret-date)
- Reservation(#id-doc,#id-user, b-reservation-date,sate).

Note that in this case study, the illustration is simplified by not considering document prolongation. The user requirements provided the basis for the rules concerning the document object (Table 1).

The first activity concerns loan management in a document center. This activity consists of checking the existence and availability of the document to loan. Whenever a document can be borrowed, this activity updates the database and records the loan. The second activity considered in this example focuses on the return of documents. This activity either records the document return by updating the database, or triggers the first activity whenever the document is reserved. Main business rules concerning these two activities are listed in Table 2.

### Tool for Active Behavior Specification

To model the behavior of active objects (Amghar & Meziane, 1996), we propose to extend a graphical description and specification language (SDL) defined by the Consultative Committee for International Telegraph and Telephony (Saracco & Tilanus, 1987). Originally, it was designed for external behavior and internal design of telephone-switching systems.
Table 2: Main business rules

<table>
<thead>
<tr>
<th>Business Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R_loan-req</strong></td>
<td>on loan request &lt;br&gt; if true &lt;br&gt; then verify possibility to &lt;br&gt; loan the document;</td>
</tr>
<tr>
<td><strong>R_ret-req</strong></td>
<td>on document return &lt;br&gt; if true &lt;br&gt; then verify document reservation;</td>
</tr>
<tr>
<td><strong>R_end-loan</strong></td>
<td>on end of loan verification &lt;br&gt; if the loan is possible &lt;br&gt; then mark document as borrowed else record a reservation;</td>
</tr>
<tr>
<td><strong>R_reservation</strong></td>
<td>on end of reservation verification &lt;br&gt; if document has been reserved &lt;br&gt; then mark document as borrowed else mark document as available;</td>
</tr>
</tbody>
</table>

Figure 3: Automaton describing behavior pattern

(Davis, 1988). SDL is a super-set of the state chart diagrams in which only state and event components are defined. The extension consists of new elements representing the condition and action of a rule.

The research goal is to adapt SDL to the ECA rule semantics. The model resulting from this extension is more useful than state diagrams because it permits the description of all components of an ECA rule, and supports the execution semantics. The significance of this graphical language is that it describes a state transition in terms of events, conditions and actions in an explicit way, thus facilitating the active behavior specification. By using this approach, a uniform and complete representation of business rules is obtained, which constitutes an important part of information systems. The generic description of the object behavior, through the use of SDL, provides automated support for two key concepts: state and business rules, as depicted in Figure 3.

The graphical model depicted in Figure 3 shows the object states as rounded rectangles. A state is described by an order number (num) identifying the state in the automaton, a status (s), and a label (l). Status is predefined and may be starting, waiting, or final. Starting state corresponds to an initial state of the object, waiting state indicates that the object is waiting for an event, and final state indicates the end of the object life cycle. The transition between the initial state and waiting state corresponds to the initial transaction, while the transition between two waiting states does not commit the initial transaction. The transition between the waiting state and the final state corresponds to the transaction commit or abort. A state label is a comment that more precisely describes the state description and waiting event. For example, the states associated with a document can be: starting when a loan request occurs, waiting for verification of existence of document, waiting for availability of document, and final when the document has been returned.
Complete Specification of Object Behavior

An object leaves its current state whenever a rule is triggered by one or more event occurrences. This triggering may lead to a change in state depending on the condition. A business rule that specifies the transition from one state to another is an extended ECA rule type. We use the ECA² rule: \( \text{on event if condition the action-1 else action-2} \), in which the graphical modeling (shown in Figure 4) necessitates the representation of each of the following elements.

**Event.** An event is considered as an aggregation of three parts and is formally written as a triplet \(<\text{type, parameters, designation}\>\) where: \( \text{type} \) is internal (database event, noted \( \text{In}\_E \)), external (event defined by user, noted \( \text{Ex}\_E \)), or temporal (reference to time in an explicit way, noted \( \text{Te}\_E \)). \( \text{Parameters} \) correspond to value or references and \( \text{designation} \) is a comment concerning the event to improve readability. The automaton distinguishes between a consumed event (input event) and a produced event (output event). The input event represents the rule trigger while the output event is raised by the action execution. These two kinds of events are represented differently.

**Synchronization.** This mechanism is utilized to specify the composite event occurrence. An ellipse in Figure 4 depicts it. It allows the description of a composite event with Boolean operators (disjunction, conjunction and negation), sequence or temporal operators. The synchronization input is a set of elementary events and the synchronization output represents the composite event.

**Condition.** A condition is a predicate or a query over a database (such as a comparison between object attributes). The condition consists of controlling the action execution. A diamond depicts it graphically depicts it. The condition evaluation may necessitate attribute values. This component offers two alternatives according to the evaluation result and renders easy the specification of extended rules. For example, ECA² is an extended rule indicating that two actions may be executed alternatively. The first one is executed in the case where the condition is true and the other whenever the condition is false. An extension of this may be denoted by ECA*' which means that various possibilities may be considered depending on the condition value.

**Action.** The action part of a rule may be seen as an operation or a task to be executed. An action can produce one or more events. Actions are depicted by a rectangle that contains a brief description of the action or event.

**Resources.** Resources are necessary for the execution of each rule component. These resources represent either attributes or objects used by both condition evaluation and action. They represent the context of the triggered rules.

*Figure 4: Graphical model for behavior representation*
Business rules are considered to be the basis for state transitions. A business rule groups one or more input events with one or more output events. For each input event, the event’s synchronization, optional condition, and action are included. The input event is mandatory because a rule cannot be triggered without such an event. At least one of the other two components must appear in the rule. The two kinds of rules are: 1) on event if condition then action, and 2) on event do action that is equivalent to on event if true then action.

**Expressiveness of the Graphical Model**

The active behavior has to be modeled in a uniform way, which means that activities are also represented through the graphical LDS. Accordingly, a diagram representing the behavior modeling with ECA rules and its relationships is elaborated for each activity. The number of business rules increases as the activity complexity increases, thus making comprehension of the diagram more difficult.

To improve the expressiveness of such a representation, a business rule is viewed (or visualized) in terms of three levels of abstraction: the **global level** in which each rule is simply identified by its name and depicted by a rectangle (Figure 4): the **event level** in which a rule is represented as an event that triggers the transition from one state to another graphically, the transition is an edge labeled by the event name; and the **complete level** in which all rules’ components are represented in detail, as shown in Figure 5. Within a behavior diagram, rules are represented in one of the levels of abstraction depending on the degree of detail chosen by the designer. These representation levels aid the visualization of the rule design. If we use a menu-based tool, the specification of the behavior is simple.

**Example of Behavior Specification**

The modeling of interactions between objects and activities of applications is very important to design active databases. Indeed, a system is considered active when objects and activities interact among themselves. The interaction diagram is essential to represent the exchange of messages. It shows the events produced and consumed by each object and each activity. It constitutes a fundamental step before the specification of each object behavior. In order to illustrate this, Figure 5 shows an example of such a diagram as applied to our case study.

Based on that interaction diagram, the behavior is specified for each object of the application. To illustrate a behavior specification, let us consider the object *Document*, of which active behavior is described in Figure 6. Note that the object *Document* leaves its initial state when the event *I_verif-loan (e2)* is raised by the activity *loan a document*. This event triggers two rules. The first rule *R_verif-nb-loans* verifies the document existence in the database, through the condition *c1*. The second rule *R_verif-availability* verifies the possibility for the user to loan an additional document. That verification is done through the condition *c2*. After that, the activity *loan document* decides the possibility of a loan, and produces an event *I_end-verif-loan (e3)*. The activity tests the parameter value of the event *I_end-verif-loan*. The object *Document* reacts to the event *I_loan (e4)* whenever borrowing is possible. The rule *R_borrow* is thus triggered and consists of recording the document loan. Furthermore, whenever the event *I_verif-reservation (e9)* is raised, the object *Document* reacts by activating the rule *R_verif-reservation* that verifies if the returned document was reserved. This rule produces the event *I_reservation-verified (e10)* allowing the activity to reserve the document by triggering the rule *R_reservation*. Table 3 summarizes this behavior as it relates to the behavior of Document as shown in Figure 6.
Figure 5: Interaction between objects of the application

<table>
<thead>
<tr>
<th>User</th>
<th>Loan activity</th>
<th>Document</th>
<th>User</th>
<th>Return activity</th>
<th>Document</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e1</td>
<td></td>
<td></td>
<td>e8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e2</td>
<td>e3</td>
<td></td>
<td>e9</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e4</td>
<td></td>
<td></td>
<td>e10</td>
<td></td>
</tr>
</tbody>
</table>

- e1: Ex_loan-req
- e2: I_verif-loan
- e3: I_end-verif-loan (possible/impossible)
- e4: I_loan
- e5: Ex_loan-notification
- e6: I_reservation
- e7: number of loans exceeded or document not available
- e8: Ex_doc-return
- e9: I_verif-reservation
- e10: I_reservation-verified (yes/no)

Table 3: Rules used by the object document

<table>
<thead>
<tr>
<th>Rule name</th>
<th>Triggering events</th>
<th>Test condition</th>
<th>Produced events</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_verif-nb-loans</td>
<td>e2: I_verif-loan</td>
<td>(nb-copies-nb-borrow)</td>
<td>c3: I_end-verif-loan</td>
</tr>
<tr>
<td>R_verif-availability</td>
<td>e2: I_verif-loan</td>
<td>state of document</td>
<td>c3: I_end-verif-loan</td>
</tr>
<tr>
<td>R_borrow</td>
<td>e4: I_loan</td>
<td>No</td>
<td>no</td>
</tr>
<tr>
<td>R_verif-reservation</td>
<td>e9: I_verif-reservation</td>
<td>state of reservation</td>
<td>e10: I_reservation-verified</td>
</tr>
<tr>
<td>R_reservation</td>
<td>e6: I_reservation</td>
<td>reservation is possible?</td>
<td>c7: Ex_notify-user</td>
</tr>
</tbody>
</table>

Figure 6: Behavior of document object
Rules Ordering and Rules Priorities

Rules may share the same triggering event. Thus, an occurrence of such an event may trigger the execution of several rules. To determine an execution order, a mechanism is necessary to avoid conflicts between rules. To resolve these conflicts, priority management may be used.

The user defines an order relation between rules. The main approaches used to define priorities are: total order and partial order. Whenever a rule is defined, a default priority is assigned to the rule. This priority may be modified. Whenever a rule is defined, “any” priority is assigned. So initially, rules have the same priority. Later, the user may specify priorities between couples of rules to determine their execution order. Rules for which “any” priority is specified are executed in their creation order. In our example, the rules R_verif-availability and R_verif-nb-loans are triggered by the same event I_verif-loan. However, no priority is necessary to fix their execution order, because it concerns only a verification of the database state. For this reason, these two rules have the same priority.

DERIVATION PROCESS

This section describes the process used during the derivation step. As shown in Figure 7, the different concepts of our model are mapped into a relational environment. Several relationships are explained.

[T1]: Each application process (or activity) is represented by a package, which can be considered an independent object.
[T2]: Each object of the data schema is transformed into a relational table.
[T3]: An active or derivation rule is translated into a stored procedure.
[T4]: When rules identified at the design level concern integrity constraints, they are mapped either to constraints embedded in the database schema or triggers. Events linked to these rules are database operations (insert, update, and delete).
[T5]: The events triggering integrity constraints or derivation rules are considered as database operations, and the events identified in each activity are replaced by procedure calls.

Figure 7: Mapping of rule-oriented model to relational representation
**Derivation of Constraints and Rules**

The rules associated with objects of the application are considered next. Rules applied to the application schema concerning database coherence are categorized as follows:

1. **Value constraints**, which concern the values of attributes. This type of constraints is supported by the word: *check*. For example, the value of attribute *state* of entity document is constrained as follows: Rule \( R_{\text{availability}}: \text{check (Document.state} \in \{\text{"available," "not available"}\} \).

2. **Referential integrity**, which supports association between objects. This type of constraint is supported by the word: *references*. For example, a document cannot be borrowed when it doesn’t exist in the document center: \( R_{\text{doc-borrow}}: \text{Borrow.id-doc references Document.id-doc} \).

3. **Cardinality constraints**, which give the number of entities that may participate in a relationship. Triggers support these constraints.

4. **Inter-entity constraints** that express a relation between two (or more) attributes belonging to different entities. Triggers also support these constraints (or more generally, these rules). They concern the application and they do not depend on a database schema.

5. **Transformation of derivation rules**. A derivation rule is used to propagate the effect of attribute modification over other attributes. This type of rule allows the determination of the value of an attribute depending on the database updating. For example, each time a new order line is performed, the amount of invoice is increased consequently. Derivation rules are mapped to triggers. These rules react to database operation.

6. **Derivation of active rules**. This type of rules is used to describe the behavior of objects. The action component corresponds to part of application code. These rules are generally triggers by either temporal or application events. Active rules are mapped to database procedures.

**Methodology**

The methodology used to model our application is based on object-oriented method and is described as follows:

1. Description of object and relationship through an ER diagram (Section *Case Study*).
2. Building of the interaction model. For our case study, Figure 5 of Subsection *Example of Behavior Specification* is an illustration of such a diagram.
3. Elaboration of state diagrams for each object and each activity. That gives a global view of the application dynamics. For example, state diagrams of document object and the two activities (loaning and restitution) are depicted in Figure 8.
4. Active behavior specification throughout our model as defined in Section *Specification of System Behavior*. This step introduces business rules including integrity constraints in an explicit way.
5. Derivation of specifications for a specific DBMS.

**Active Behavior Description**

As shown in Figure 6, a document reacts to events produced by the activity *loan document*. The document behavior, as described in Section *Example of Behavior Specification*, is a sequence of steps: checking the number of documents for a given user, and then the availability of the requested document. Table 4 illustrates the different rules used in these activities.
The active behavior of two activities is depicted in Figure 9. The goal of the first activity is the loan management in the document center. This activity, triggered by a user request, is composed of two operations, each one supported by an ECA² rule. The first rule checks the possibility of borrowing the document. The second rule processes the document loan. The second activity concerns the document restitution. Its goal is to manage returned documents. This activity consists of two rules. The first rule identifies whether the returned document was reserved. The second rule marks the returned document as available or records the new loan. Figure 9 depicts the behavior specification of the loan and return document activities.

**Activity Description**

Each activity (loan and restitution) is a mapped package where each activity rule is translated to a stored procedure. Thus, each stored procedure represents a business rule. This mapping allows the designer to consider the rules because of their modularization into the package’s procedures. We summarize the derivation of different rules from our case study. Table 6 shows the translation of each rule into a relational model. Note that rules implemented with relational database system mechanisms are triggered by events dealing with database operations (insert, update and delete) and rules represented by procedures are triggered by procedure calls. Table 5 illustrates some rules with their triggering events.

**RULE MODULARIZATION**

Active rules have the property to react among themselves to build a complex interaction net. In order to facilitate exploitation and maintenance of an important set of rules, several authors (Baralis, Ceri, & Paraboschi, 1996; Aiken, Widom, & Hellerstein, 1992; Aiken,
Hellerstein, & Widom, 1995) propose to subdivide the rule set into disjoint subsets called strata. Each of them must individually support termination and confluence of rules’ stratum.

**Termination.** Whenever rules are executing, they may trigger themselves. The property of termination is important to warrant that all rules belonging to the same triggering graph terminate. Particularly, if there are no cycles in the graph, termination is guaranteed.

**Confluence.** Whenever two rules must be executed due to the occurrence of the same event, they are said to be confluent whenever their execution order has no influence on the final result on the database. Otherwise, priorities would be given to the rules.

**Rule Stratification**

The different types of stratification defined by Baralis et al. (1996) are:

**Behavioral stratification.** A stratum contains rules corresponding to an activity or a task of an application. Each stratum convergence is described by a metric measuring the progression of rules during activity execution. Strata are ordered according to priorities. The
metric is a decreasing numeric function \( m_i \) corresponding to a progression of the state of the database to a final state. More precisely, the final value of the metric can only converge to a minimum. To verify conservation between strata, any stratum can increase the metric of strata of high-level priority.

**Assertional stratification.** It is used to group rules whose goal is to achieve a compensatory action after a constraint violation. A post-condition is associated with each stratum, and whenever stratum rules are activated, they must restore the post-condition. Conservation between strata verifies that any stratum can modify the post-condition value of a stratum of higher priority.

**Event-based stratification.** Event-based stratification is applicable when it is possible to associate rules with specific events. Rules within a stratum respond to input events and produce output events; therefore, each stratum establishes a dependency relationship between sets of events. Event-based stratification requires such a relationship to be non-cyclic. Baralis et al. (1996) have shown that behavioral stratification subsumes assertional and event-based stratification and the intersection of the latter two is not empty. Consequently it is possible to define a metric over any type of stratification.

**A New Proposal for Partitioning**

We propose that the difference between these three stratifications concerns convergence criterion. Indeed, behavioral stratification must observe a metric progression, assertional stratification must verify a post-condition (Boolean expression), and event-based stratification implies a hierarchy of strata such that a stratum of a given priority cannot trigger rules of strata of higher priority. The main drawback of these stratifications remains the choice of convergence criterion for a given application. It is difficult to choose between stratification types. Although partitioning criterion for event-based stratification is interesting, the constraint of stratum that does not trigger a stratum of a higher priority is not realistic.

Concerning assertional stratification, it differs from behavioral stratification only by the type of criterion (Boolean in first case and numeric in second case). Moreover the post-condition can be a rule condition. For behavioral stratification, the main problem concerns metric identification that is the difficulty to find a particular function describing database progression from one state to final state. This function is different according to applications. To conclude, these types of stratification do not lead to result unity and render any automaton process difficult.

**Partition Based on Triggering Graphs**

After these remarks, we propose to partition a rule set according to the triggering graph. This graph is issued from rule chaining triggered by event occurrence. Analysis of automaton (object diagram presented in Section Specification of System Behavior), and particularly of their links, allows the building of a graph in which an oriented link between a node \( A \) and a node \( B \) means that rules corresponding to \( A \) trigger rules corresponding to \( B \). The rule set is subdivided into a set of the partitions. Let us note that partitions obtained are not necessarily disjoint. The partitioning process can be summarized as follows:

1. Choose a node (a rule \( r_k \)) from the graph (a rule that is not triggered by another one).
2. Build a triggering graph of this rule as being a set of rules that are activated from \( r_k \) directly or in a transitive way.
3. Delete \( r_k \) from the initial rule set.
4. Repeat step 1 until the rule set is empty.
Based on Figure 9, three examples of rule stratifications issued from loan and return management of documents can be considered. Each stratum corresponds to a triggering graph. The first stratum corresponds to a set of rules that are executed to respond to the user loan request. Let us note that the rule \textit{R\_verif-availability} may be skipped. There are two alternatives to this stratum because there is no priority between \textit{R\_verif-availability} and \textit{R\_verif-nb-borrow} corresponding to stratum \textit{s11} and stratum \textit{s12}. The rules \textit{R\_verif-availability} and \textit{R\_verif-nb-borrows} are thus confluent. Stratum \textit{s2} corresponds to the treatment of the user loan request in the case where the loan is impossible. Stratum \textit{s3} corresponds to the document return.

**Metric for a Convergence**

A metric allows for the measurement of rule execution progression within a partition. However, it is important to establish a hypothesis warranting convergence and confluence of rules. The convergence translates a distance expressing the progression of database from one state to final state. These hypotheses are:

1. Whenever an event occurs, a transaction is created to execute all rules triggered by this event. Rules can belong to different partitions.
2. To delete cycles, a rule is triggered only once within the same transaction. In other words, a rule is activated for the current transaction. Since a rule cannot be triggered any other time, a cycle is impossible. The rule is reactivated at the commit point of the transaction.
3. A rule can trigger one or more other rules through its action part. Rule action is an operation sequence including database operations.
4. Whenever several rules are in conflict, priorities are applied. The programmer explicitly attributes them or they can be affected, for example, according to the creation order of the rules.

**ENVIRONMENT FOR ACTIVE DATABASE DESIGN**

**Architecture for New Functionalities of CASE**

The different stages of the design process, with regard to stages used in other object-oriented methods, remain unmodified. However, the models’ elaboration within each stage must include the specification of active rules. We have shown in the previous section how active rules can be represented in the specification of an object behavior. Static models would also include integrity-constraint descriptions. The proposed environment is designed as a set of functionalities to support all steps of the active database design. It may be considered an extension of existing CASE tools for both designing and generating a set of rules that control the behavioral aspect of a database. Modules required treating dynamic aspects concern mainly:

1. The validation of both the semantic and the syntactic of the specification developed in Section Specification of System Behavior. The first verification avoids the transition from one state to another without taking into consideration the business rule, while the second verifies that the produced event is done during (or at the end of) the action and after the event, which has triggered the action. This module
Modeling of Business Rules For Active Database Application Specification

2. The partitioning of active rules considered as the principle key design to warrant their modularization and control. Rules are partitioned into subsets so that the designer can abstract the rule behavior by reasoning locally on each individual subset. In order to generate coherent rule sets, rules are organized according to a criterion that consists of grouping rules by call graphs according to the partitioning algorithm (Section Rule Modularization).

3. The generation of rules from each rule partition elaborated by the partitioning module using capabilities of the target DBMS. It is obvious that if the DBMS has active features, the derivation is easier in comparison with a passive DBMS in which rules must be coded as program pieces. The generation can be done towards any kind of DBMS (relational or object-oriented). In the particular case of relational DBMS, rules are essentially used to enforce database integrity and are coded through triggers. Some details were given in Section Derivation Process.

Active Database Applications

Several categories of applications can be distinguished (Paton & Diaz, 1999):

Database system extension. Active rules are generally used as a primitive mechanism for supporting the implementation of other components of a database system. ECA rules have been widely used to support integrity constraints, materialized views, derived data, transaction models, and automatic screen updating in the context of database change.

Internal database application. In this case, applications imply the use of active functionality to describe some of behavior to be manifested by a software system without reference to external devices or other system. Stock management and sales monitoring are some examples of this category of applications.

External database applications. Applications focus on the use of active database properties in order to record and respond to external situations. For instance, in a medical domain, physicians can be warned of changes in patients’ conditions. Applications respond to generic requirements and are generated ad hoc. Rules of these applications are called business rules to indicate their business-oriented origin. In some cases, external applications model the laws regulating the behavior of physical systems.

CONCLUSION

This chapter is a contribution to modeling business rules at the design level as any part of an information system. Considering a behavior specification with state graph, business rule is seen as a means of transit from one state to another describing the conditions under which the system to model is submitted whenever an event occurs. Thus, the primary interest was to explicitly introduce business rules during the modeling of the different components of a system. The proposal was to extend a graphical description and specification language in order to represent all components of an ECA rule. In addition, the general architecture of an environment to design active databases and the specifications of theoretical aspects to elaborate a monitoring and debugging tool are presented. The classic cycle of analysis, design, prototyping and implementation of database applications is enhanced with a more complete specification of behavior since business rules are considered in the beginning of the designing cycle. The specification language of behavior, presented in this chapter, is
chosen such that all details of business rules, represented with ECA formalism, are described with a same level of importance.

This work addresses more generally the extension of CASE to add active functionalities to existing ones. A promising way is to add mechanisms to generate templates for coding rules from behavior specification tools. For example, from graphical representation of UML state diagrams in ROSE case, it would be interesting to generate ECA pattern corresponding to transitions between states. Moreover, the rule modularization is important and necessary to abstract all rules identified during the analysis stage (Geppert, Berndtsson, Lieuwen, & Roncancio, 1998). This work contributes to the extension of existing models and tools to take into account active features of a database. Future work will study the correctness of such extensions. For this proposed work, the extended Petri Net seems to constitute an ideal solution because 1) the transformation from state diagrams including ECA rules to Petri Nets is relatively obvious, and 2) the formal basics of Petri Nets can be very useful to show, for example, the trace of a transaction or to verify the termination of a set of rules.

ENDNOTE

* With the assumption that the action part of a rule always terminates.

REFERENCES


In a distributed database system, an increase in workload typically necessitates the installation of additional database servers followed by the implementation of expensive data reorganization strategies. We present the Partial REALLOCATE and Full REALLOCATE heuristics for efficient data reallocation. Complexity is controlled and cost minimized by allowing only incremental introduction of servers into the distributed database system. Using first simple examples and then, a simulator, our framework for incremental growth and data reallocation in distributed database systems is shown to produce near optimal solutions when compared with exhaustive methods.

INTRODUCTION

Recent years have witnessed an increasing trend of the implementation of Distributed Database Management Systems (DDBMS) for more effective access to information. An important quality of these systems, consisting of \( n \) servers loosely connected via a communication network, is the ability to adjust to changes in workloads. To service increases in demand, for example, additional servers may be added to the existing distributed system and new data allocations computed. Conventionally, this requires a system shutdown and an exhaustive data reallocation. Such static methods are not practical for most organizations, for these methods result in high costs and in periods of data unavailability.
We present the incremental growth framework to address incremental expansion of distributed database systems. Data is reallocated using one of two data reallocation heuristics - Partial REALLOCATE or Full REALLOCATE. Both heuristics are greedy, hill-climbing algorithms that compute new data allocations based on the specified optimization parameter of the objective cost function. Due to their linear complexity, both heuristics can be used to solve both small and large, complex problems, based on organizational needs. The robustness of the heuristics is demonstrated first by simple, illustrative examples and then by parametric studies performed using the SimDDBMS simulator.

The REALLOCATE algorithms in conjunction with SimDDBMS can be used to answer many practical questions in distributed database systems. For example, in order to improve system response time, a database administrator (DBA) may use SimDDBMS for parametric evaluation. For example, the DBA may analyze the effect of upgrading CPU processing capability, increasing network transfer speed, or adding additional servers into the distributed database system. Furthermore, SimDDBMS may easily be modified to evaluate heterogeneous servers, with different CPU processing capabilities. A DBA may also use SimDDBMS to determine the impact and cost-benefit analysis of adding some number, \( s \geq 1 \), of additional servers at one time.

**RELATED WORK**

The allocation of data amongst the servers of a distributed database system has been widely studied as the “data allocation” or “file allocation” problem. The data allocation problem generally attempts to strategically place relations at servers in the distributed database system so as to yield maximum composite system performance (as determined by the global objective function). Remote access to data is generally slower and more costly than local access to data, because remote access results in network delays and communication costs in addition to disk access delays and costs. A distribution of files that results in many remote accesses to data files will undoubtedly degrade system performance. Burdening a server with large local processing demands may cause bottlenecks in the system, also leading to degradation in system performance. Therefore, data allocation algorithm(s) must consider both remote access costs and local processing costs when computing the new data allocation. Finally, since the query strategies implemented in the system also contribute significantly to the actual optimal data allocation, these must also be considered.

Following the pioneering work in Porcar (1982), many researchers have studied the data allocation problem (Daudpota, 1998; So, Ahmad, & Karlapalem, 1998; Tamhankar & Ram, 1998; Ladjel, Karlapalem, & Li, 1998). The single data allocation problem has been shown to be intractable (Eswaran, 1974), which means that as the problem size increases, problem search space increases exponentially (Garey & Johnson, 1979). Due to the complex nature of the problem, some researchers (Cornell & Yu, 1990; Rivera-Vega, Varadarajan, & Navathe, 1990; Lee & Liu Sheng, 1992; Ghosh & Murthy, 1991; Ghosh, Murthy, & Moffett, 1992) have resorted to integer programming methods in search for good solutions. Since optimal search methods can only be used for small problems, heuristic methods are often used for solving large data allocation problems (Apers, 1988; Blankinship, 1991; Ceri, Navathe, & Wiederhold, 1983; Du & Maryanski, 1988).

Researchers have studied both the static data allocation problem, in which data allocations do not change over time, and the dynamic data allocation problem (Theel &
Pagnia, 1996; Wolfson, Jajodia, & Huang, 1997; Brunstrom, Leutenegger, & Simha, 1995),
which may be adaptive or non-adaptive. Adaptive models (Levin, 1982; Son, 1988; Levin &
Morgan, 1978) are implemented when the system senses a substantial deviation in access
activities; these models determine a one-time reallocation (for a relatively short period of time)
in response to surges in demand. For example, the volume of reservations for a particular
airline route may increase during a specific season. Therefore, an airline reservation system
may temporarily store additional copies of the files associated with the route at a local server.
However, this is a short-term situation that is resolved by introducing replicated file copies.
Non-adaptive models (Levin, 1982; Porcar, 1982, Segall, 1976) are employed at the initial
system design stage or upon system reorganization; these models do not adjust to variations
in system activities.

Most previous research on data allocation assumes a fixed number of servers in the
distributed database system (Carey & Lu, 1986; Chu, 1969, Laning & Leonard, 1983; Lee &
Liu Sheng, 1992; Rivera-Vega, Varadarajan, & Navathe, 1990). Experiments and simulations
are designed to test DDBMS factors such as the degree of data replication, workloads per
server, and different levels and classes of queries and transactions (Carey & Lu, 1986; Ciciani,
Dias, & Yu, 1990). Simulation runs vary the number of servers to arbitrary values. However,
these values are fixed per run and vary only between runs. Incremental system growth and
subsequent data reallocation has not previously been addressed.

RESEARCH ASSUMPTIONS

We assume a fully connected, reliable communication network (Liu Sheng, 1992) with
all servers having equal local storage and processing capacity. Additionally, we assume
transmission capacity is equal between all pairs of servers (Blankinship, 1991; Goyal, 1994).
Since storage costs are small compared to local processing and transmission costs, only the
latter two costs are considered in cost modeling.

We assume a single copy of each relation is allocated to the distributed database system
(Blankinship, 1991; Cornell & Yu, 1989). (Data partitioning and data replication are not
considered in this research, but are included in the discussion of future extensions of this
work.) We assume that for every relation in the system, there is only one copy that is
designated as the primary copy (Garcia-Molina & Abbott, 1987; Son, 1988). Although back-
up copies may be stored in the distributed database system, only the primary copy is used
in transaction processing.

The relational data model is used to describe the data and query processing on the data.
Only simple queries are considered. Queries are assumed to be independent and are solved
independently. Queries are processed in parallel in the distributed database system. To
simplify the estimation of query result sizes, the concept of selectivity (Blankinship, 1991;
Chin, 1999; Date, 1991; Goyal, 1994) is utilized. Attribute values are assumed to be
uniformly distributed and each attribute in a relation is assumed to be independent of all other
attributes in the database. The simple query environment has been chosen because it has a
manageable complexity while remaining realistic and interesting.

For local transaction processing, we assume each server in the distributed database
system maintains an incoming queue of requests. Queries are maintained in the queue until
they are processed. A First In, First Out (FIFO) order is used for query processing.
Additionally, we assume the distributed database system maintains a centralized data dictionary. The following information is maintained in the data dictionary (Blankinship, 1991; Chin, 1999; Goyal, 1994; Hevner & Yao, 1979):

- **S**: Network Servers, \( i = 1, 2, \ldots, s, s+1 \) (\( S_{s+1} \) = the new server joining the system)
- **R**: Relations, \( j = 1, 2, \ldots, r \)

For each relation \( R_j \), \( j = 1, 2, \ldots, r \):
- \( n_j \): number of tuples,
- \( a_j \): number of attributes,
- \( \beta_j \): size (in bytes)

For each attribute \( d_{jk} \), \( k = 1, 2, \ldots, a_j \) of relation \( R_j \):
- \( p_{jk} \): attribute density, the number of different values in the current state of the attribute divided by the number of possible attribute values. During joint operations the density is used as a selectivity coefficient.
- \( w_{jk} \): size (in bytes) of the data item in attribute \( d_{jk} \).

The density, \( p_{jk} \), of attribute \( d_{jk} \) is defined as the number of different values in the attribute divided by the number of all possible values in the domain of the attribute. So, \( 0 \leq p_{jk} \leq 1 \) (Hevner & Yao, 1979).

**RESEARCH APPROACH**

If we define \( S \) as the number of servers in the distributed database system and \( R \) as the number of relations that are to be allocated amongst the \( S \) servers, then the search space of the data allocation problem is \( S^R \). This means, for example, that if we are allocating a single copy of each of ten relations in a distributed database system consisting of five servers, there are \( 5^{10} \), or 9,765,625 possible allocations. Allocating a single copy of ten relations in a distributed database system consisting of six servers yields 60,466,176 possible allocations. As the number of servers and relations increases, the data allocation problem quickly becomes NP hard (Eswaran, 1974; Rivera-Vega et al., 1990). Therefore, in this research, we develop heuristic, “hill-climbing” algorithms to determine new data allocations.

The optimal data allocation is dependent on the query strategies used by the query optimization algorithm. Algorithm Serial (Hevner & Yao, 1979) considers serial strategies to minimize total transmission time in the simple query environment. For each query \( q \) accessing \( i \) relations, there are \( i! \) possible combinations for processing \( q \). Consider, for example, a query that accesses relations \( A \), \( B \), and \( C \). Then, the \( i! = 6 \) processing combinations for the query are:

\[
\begin{align*}
A &\rightarrow B \rightarrow C \\
A &\rightarrow C \rightarrow B \\
B &\rightarrow A \rightarrow C \\
B &\rightarrow C \rightarrow A \\
C &\rightarrow A \rightarrow B \\
C &\rightarrow B \rightarrow A
\end{align*}
\]

Therefore, if we have four queries, two of which access two relations, one of which accesses three relations, and one of which accesses four relations, then the number of possible serial strategy combinations is \( (2!)(2!)(3!)(4!) = (2)(2)(6)(24) = 576 \). The serial strategy consists of transmitting each relation, starting with \( R_i \), to the next relation in a serial order. The strategy is represented by \( R_1 \rightarrow R_2 \rightarrow \ldots \rightarrow R \), where \( ? \) is the number of relations in the
Algorithm Development, Simulation Analysis and Parametric Studies 161

query (Hevner & Yao, 1979). The serial order is computed so that \( ?_1 = ?_2 = \ldots = ?_r \), where \( ?_j \) is the size of relation \( R_j, j = 1, \ldots, r \) (Hevner & Yao, 1978).

Optimizing query strategies is not within the scope of this research. However, since the optimal data allocation is dependent on the implemented query strategy, when computing new data allocations, we implement the Hevner and Yao (1979) Algorithm Serial for query processing. Any query optimization algorithm from the research literature, however, can be used in place of Algorithm Serial.

THE INCREMENTAL GROWTH FRAMEWORK

The incremental growth framework (see Figure 1) may be invoked when system performance, as computed using the objective cost function, is below the acceptable threshold (specified by the DBA). To return to an acceptable state, new servers are introduced incrementally, one at a time, into the distributed database system. With the introduction of each new server, a new data reallocation for the system is computed. This process is iteratively executed until acceptable performance is achieved or the number of servers equals the number of relations in the distributed database system (the latter constraint can easily be relaxed in a distributed database system housing partitioned data).

The incremental growth framework, which can easily be adapted for one-server or multiple-server systems, can be used by small, mid-size, and large organizations, each having distributed database systems of varying size. In one-server systems, the initial data allocation locates all relations at the server. Future allocations are computed based on this information. In multiple-server systems, the current data allocation is required as input into the framework.

Additional input information required for the incremental growth framework includes: the database server or servers, including the local processing capacity; the network topology, including transmission capacity; the database relations, including relation sizes and selectivities; the query set, the optimization parameter, and the acceptable threshold for the optimization parameter.

By independently reallocating each relation, one relation at a time, to only the new server joining the distributed database system, Algorithm REALLOCATE only considers the effect on the global objective function of strategically chosen data reallocations. The effect of moving a relation to any server except the new server is not considered.

The logic of the REALLOCATE heuristic is summarized in the pseudocode of the following procedure incremental Reallocation:

\[
S_{s+1}: \text{ new server in the distributed database system} \\
R = \{R_1, \ldots, R_r\}: \text{ relations in the distributed database system} \\
\delta( R_j \rightarrow S_{s+1} ): \text{ value of objective cost function when } R_j \in R \text{ is allocated to } S_{s+1}
\]

procedure incrementalReallocation;
{ 
    bestAllocation = notFound;
    while ( numberOfIterations = r ) && ( bestAllocation = notFound ) do
        for each \( R_j \in R \)
            \( \delta_j = \delta( R_j \rightarrow S_{s+1} ); \)
            \( \Phi_s = \min \delta_j; \)
        
}
Figure 1: Incremental Growth Framework

Inputs:
- Database Server(s)
- Network Topology
- Database Relations
- Queries
- Threshold
- (Data Allocation)

Outputs:
- Database Servers
- New Data Allocation

- Introduce New Server
- REALLOCATE Algorithm

Current Cost > Threshold

Yes
- Local Optimum

No
Algorithm Development, Simulation Analysis and Parametric Studies  

Algorithm REALLOCATE seeks to find good data allocation solutions using the minimum number of relation-new_server configurations. Because Algorithm REALLOCATE does not perform a complete data reallocation, it does not have to analyze moving each of the relations to each of the servers. Because Algorithm REALLOCATE is an iterative, greedy, hill-climbing heuristic, it will not traverse the same path twice. With each iteration, it will find a lower cost solution or it will terminate.

Since the maximum number of relations under consideration in Algorithm REALLOCATE at any given time is finite and the number of servers under consideration always equals one, Algorithm REALLOCATE is guaranteed to converge. The total search space for Algorithm REALLOCATE is bounded by $\sum_{r=1}^{R} r = \sum_{r=1}^{10} r = 55$ relation-new_server configurations; and, the maximum number of iterations of Algorithm REALLOCATE is bounded by $R = 10$. When adding an additional server (for a total of seven servers), Algorithm REALLOCATE is still bounded by a maximum of only $\sum_{r=1}^{R} r = \sum_{r=1}^{10} r = 55$ relation-new_server configurations and by $R=10$ iterations!

A better data reallocation may be obtained by completely reallocating all of the data amongst all of the servers each time a new server is added into the distributed database system. However, the search space for an Exhaustive Search reallocation, given by $S^R$ (where $S$ is the number of servers and $R$ is the number of relations) increases exponentially. Therefore, an exhaustive reallocation would be costly and would quickly become intractable (Eswaran, 1974; Rivera-Vega et al., 1990).

APPLYING THE INCREMENTAL GROWTH FRAMEWORK IN THE SIMPLE QUERY ENVIRONMENT

When measuring system response time, the objective cost function consists of three cost components: transmission costs, local processing costs, and queuing costs. Costs are
measured in terms of the number of CPU cycles or time ticks (Goyal, 1994) needed to complete a task. The system response time is equal to the number of CPU cycles needed to process \( Q = \{Q_1, ..., Q_q\} \) queries.

Transmission cost equations are identical between any two servers (Blankinship, 1991; Goyal, 1994), and costs are based on the amount of data transmitted. Local processing cost equations are also identical at any two servers, and costs are based on the amount of data processed. Queuing costs are based on the number of CPU cycles a query spends in queue at each server. Storage costs are considered negligible and are not considered in the cost analysis.

Using the additional notation (Table 1), we state our cost equations as follows (Goyle, 1994):

\[
QR_n = (CS_n)(\beta_j) \quad n = 1, ..., q; j = 1, ..., r
\]

\[
LPT_{ni} = \frac{\beta_j + QR_n}{\rho} \quad \text{at } S_i \quad n = 1, ..., q; j = 1, ..., r; i = 1, ..., s+1
\]

\[
NTT_{ni} = \frac{QR_n}{\mu} \quad \text{from } S_i \text{ to } S_i \quad n = 1, ..., q; i, i = 1, ..., s+1; i \neq i
\]

\[
QWT_{ni} = T_{p_n} - T_{Q_n} \quad n = 1, ..., q; i = 1, ..., s+1
\]

where,

\( T_{Q_n} \) = the CPU cycle \( S_i \) places \( Q_n \) into its queue

\( T_{P_n} \) = the CPU cycle \( S_i \) begins processing \( Q_n \)

\[
\begin{array}{|c|c|}
\hline
\text{CS}_n & \text{cumulative selectivity for } Q_n \quad n = 1, ..., q \\
\hline
\text{QR}_n & \text{query result for } Q_n \quad n = 1, ..., q \\
\hline
\text{QWT}_{ni} & \text{wait time in queue for } Q_n \text{ at } S_i \quad n = 1, ..., q; i = 1, ..., s+1 \\
\hline
\text{LPT}_{ni} & \text{local processing time for processing } Q_n \text{ at } S_i \quad n = 1, ..., q; i = 1, ..., s+1 \\
\hline
\text{NTT}_{ni} & \text{network transfer time for transferring } Q_n \text{ from } S_i \text{ to } S_i \quad n = 1, ..., q; i, i = 1, ..., s+1; i \neq i \\
\hline
\Theta_{ni} & \text{transmission of } Q_n \text{ to } S_i \quad n = 1, ..., q; i = 1, ..., s+1 \\
\hline
\rho & \text{CPU rate per CPU cycle} \\
\mu & \text{network transfer rate per CPU cycle} \\
\end{array}
\]
The objective cost function when computing data allocations to minimize time is to minimize:

\[ RT = \sum_{i=1}^{T} C(t) + \varsigma \]

where

\[ C(t) = \begin{cases} 
1 & \text{if} (Q_n \in Q \text{ in queue at } S_i) \\
\lor (Q_n \in Q \text{ in process at } S_i) \\
\lor (Q_n \in Q \text{ in transmission } S_i \rightarrow S_i) \\
0 & \text{otherwise} 
\end{cases} \]

and \( \varsigma \) = system idle time while processing \( Q = \{Q_1, ..., Q_q\} \).

The following example demonstrates cost computations:

**Example**

Given the relations \( R = \{ R_A, R_B, R_C \} \), with selectivities, size \( \beta \), and server allocations on \( S = \{ S_1, S_2 \} \) as follows (Table 2) and the following queries (Table 3) and query strategies (computed so that \( \beta_1 = \beta_2 = ... = \beta \)). Assuming \( \rho = 10 \) bytes per unit time and \( \mu = 20 \) bytes per unit time (Table 4).

Based on the above example, the system response time is 370 CPU cycles. The response time for query \( Q_1 \) is 221-10=211 CPU cycles and the response time of \( Q_2 \) is 370-15=355 CPU cycles.

### Table 2: Relations and allocations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Selectivity</th>
<th>( \beta )</th>
<th>Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_A</td>
<td>0.2</td>
<td>200</td>
<td>S_1</td>
</tr>
<tr>
<td>R_B</td>
<td>0.4</td>
<td>400</td>
<td>S_1</td>
</tr>
<tr>
<td>R_C</td>
<td>0.6</td>
<td>600</td>
<td>S_2</td>
</tr>
</tbody>
</table>

### Table 3: Queries and query strategies

<table>
<thead>
<tr>
<th>Query ID</th>
<th>Query</th>
<th>Query Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q_1</td>
<td>Join A, B, &amp; C</td>
<td>R_A \rightarrow R_B \rightarrow R_C</td>
</tr>
<tr>
<td>Q_2</td>
<td>Join B &amp; C</td>
<td>R_B \rightarrow R_C</td>
</tr>
</tbody>
</table>
Table 4: Example–Cost computations

<table>
<thead>
<tr>
<th>CPU cycle</th>
<th>Processing and Computations (unit = CPU cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>( \Theta_{11}; S_1 ) places ( Q_1 ) in its queue; ( QR_1=0; CS_1=1.0 )</td>
</tr>
<tr>
<td>15</td>
<td>( \Theta_{21}; S_1 ) places ( Q_2 ) in its queue; ( QR_2=0; CS_2=1.0 )</td>
</tr>
</tbody>
</table>
| 30        | \( S_1 \) begins processing \( Q_1 \); \( QWT_{11}=30-10=20; \)
|           | \( \frac{200 + 0}{10} = 20 \)
|           | \( LPT_{11} = \frac{200 + 0 + 200}{10} = 80 \)
|           | \( QR_1 = (1.0)(200) = 200; CS_1 = (1.0)(0.2) = 0.2 \) |
| 50        | Since \( R_B \) is also at \( S_1 \), \( NTT_{111} = 0; S_1 \) begins processing \( Q_1 \) with \( R_B \); \( QWT_{11} = 20; \)
|           | \( \Theta_{21}; NTT_{112} = \frac{80}{20} = 4 \)
|           | \( S_1 \) begins processing \( Q_2 \); \( QWT_{21} = 95; LPT_{21} = \frac{400 + 0}{10} = 40 \)
|           | \( QR_1 = (0.2)(400) = 80; CS_1 = (0.2)(0.4) = 0.08 \) |
| 110       | \( S_1 \) begins processing \( Q_2 \); \( QWT_{21} = 95; LPT_{21} = \frac{400 + 0}{10} = 40 \)
|           | \( QR_2 = (1.0)(400) = 400; CS_2 = (1.0)(0.4) = 0.4 \) |
| 114       | \( S_2 \) places \( Q_1 \) in its queue |
| 150       | \( \Theta_{21}; NTT_{212} = \frac{400}{20} = 20 \)
|           | \( S_2 \) begins processing \( Q_1 \); \( QWT_{22} = 36; LPT_{22} = \frac{600 + 80}{10} = 68 \)
|           | \( QR_1 = (0.8)(600) = 48; CS_1 = (0.08)(0.6) = 0.048 \) |
| 170       | \( S_2 \) places \( Q_2 \) in its queue |
| 190       | \( S_2 \) begins processing \( Q_2 \); \( QWT_{22} = 20; LPT_{22} = \frac{600 + 400}{10} = 100 \)
|           | \( CS_2 = (0.4)(0.6) = 0.24 \) |
| 218       | \( \Theta_{(client)}; NTT_{12(client)} = \frac{48}{20} = 2.4 \) |
| 221       | Client receives result of \( Q_1 \) |
| 290       | \( \Theta_{(client)}; NTT_{22(client)} = \frac{160}{20} = 80 \) |
| 370       | Client receives result of \( Q_2 \) |
DATA REALLOCATION HEURISTICS

We present two heuristics for data reallocation: Partial REALLOCATE and Full REALLOCATE. Both algorithms are greedy, iterative, “hill-climbing” heuristics that will not traverse the same path twice. With each iteration, they will find a lower cost solution, or they will terminate. Both algorithms require as input: the current data allocation, the relations, and the queries in the distributed database system.

We define the notation:

\[ S = \{S_1, \ldots, S_s, S_{s+1}\} : \text{set of servers (} S_{s+1} = \text{the new server)} \]

\[ R = \{R_1, \ldots, R_r\} : \text{set of relations allocated to } S_i, i = 1, \ldots, s \]

\[ R' = \{R'_1, \ldots, R'_r\} : \text{set of relations allocated to } S_{s+1} \]

\[ R \cap R' = \emptyset \]

\[ R_j \Rightarrow S_i : \text{permanent allocation of } R_j \text{ to } S_i, j = 1, \ldots, r; i = 1, \ldots, s \]

\[ R_j \rightarrow S_i : \text{temporary allocation of } R_j \text{ to } S_i, j = 1, \ldots, r; i = 1, \ldots, s \]

\[ O_o = \delta( R_j \in R, R_j \Rightarrow S_i), \ i = 1, \ldots, s \]

\[ O_{(s+1)j} = \delta( R_j \rightarrow S_{s+1}) \ j = 1, \ldots, r \]

where \( \delta( R_j \rightarrow S_i) \) and \( \delta( R_j \Rightarrow S_i) \) is the objective cost function evaluated for \( R_j \rightarrow S_i \) and \( R_j \Rightarrow S_i \), respectively.

Each relation \( R_j \) must be allocated to a server and can be allocated to only one server at any given time. Therefore,

\[ \sum_{i=1}^{s+1} X_{ij} = 1 \quad \text{where} \quad X_{ij} = \begin{cases} 1, & R_j \rightarrow S_i \lor R_j \Rightarrow S_i \\ 0, & \text{otherwise} \end{cases} \]

PARTIAL REALLOCATE

The Partial REALLOCATE algorithm computes the new data allocation based on the specified optimization parameter. Partial REALLOCATE is a greedy heuristic that minimizes the number of \( \delta( R_j \rightarrow S_{s+1}) \) evaluations while searching for a better solution. Therefore, Partial REALLOCATE does not traverse the same path twice. The number of Partial REALLOCATE tests per additional server is bounded by \( R \), the number of relations in the distributed database system. The steps of the Partial REALLOCATE algorithm are:

Step 1: Compute \( O_o \).

Step 2: For each \( R_j \in R \), Compute \( O_{(s+1)j} = \delta( R_j \rightarrow S_{s+1}) \), where for \( R'' = R - R_j \), \( R_k \)'s \( \in R'' \), \( R_k \not\Rightarrow S_{s+1} \), \( 1 = k = (r-1) \).
Step 3: Compare $O_\gamma = \min_j O_{(s+1)j}$ to $O_o$. If $O_\gamma = O_o$, $O_o$ is the local optimum. If $O_\gamma < O_o$, update $O_o$ to $O_\gamma$. $R' = R' + R_j, R = R - R_j, R_j \Rightarrow S_{s+1}$.

Consider, for example, the relations and server allocations in Table 1. Assume $S_{s+1} = S_3$ is the new server joining the distributed system. Then, in Step 1, Partial REALLOCATE computes the value of the objective function given the current allocation. Assume $O_o = 115$. In Step 2, Partial REALLOCATE computes the value of the objective function when independently moving each relation to $S_{s+1}$:

- Move only $R_A$ to $S_3$: Compute $O_{(s+1)A} = \delta(R_A \rightarrow S_3); R'' = \{R_B, R_C\}$
- Move only $R_B$ to $S_3$: Compute $O_{(s+1)B} = \delta(R_A \rightarrow S_3); R'' = \{R_A, R_C\}$
- Move only $R_C$ to $S_3$: Compute $O_{(s+1)C} = \delta(R_A \rightarrow S_3); R'' = \{R_A, R_B\}$

Assume $O_{(s+1)A} = 100, O_{(s+1)B} = 75$, and $O_{(s+1)C} = 125$. Then, in Step 3, Partial REALLOCATE selects the move resulting in the minimum cost, $O_\gamma = O_{(s+1)B} = 75$. Since $O_\gamma < O_o$, a lower cost solution has been found; $R_B$ is relocated from $S_1$ to $S_3$.

Partial REALLOCATE minimizes the number of $\delta(R_j \rightarrow S_{s+1})$ evaluations while searching for a better solution. The number of Partial REALLOCATE tests per additional server is bounded by $R$, the number of relations in the distributed database system. Partial REALLOCATE is not guaranteed to find the optimum solution (as determined using Exhaustive Search). However, if evaluating $\delta(R_j \rightarrow S_{s+1})$ is expensive and/or if Partial REALLOCATE’s percentage deviation from the optimal solution is acceptable, Partial REALLOCATE is more cost-effective than either Full REALLOCATE or Exhaustive Search.

**FULL REALLOCATE**

The Full REALLOCATE algorithm differs from the Partial REALLOCATE algorithm in that Full REALLOCATE iterates a greater number of times before choosing $O^y = \min_j O_{(s+1)j}$ from the $y$th iteration of the algorithm. As in Partial REALLOCATE, Full REALLOCATE computes $\delta(R_j \rightarrow S_{s+1})$ for each $R_j \in R$. Holding the $R_j$ yielding $O_{(s+1)j}$ at $S_{s+1}, R' = R' + R_j, R = R - R_j$. Full REALLOCATE reiterates with a new $R_j \in R$ until either $O_x = O_{(s+1)x}$ or $R = \emptyset$. The number of Full REALLOCATE tests per additional server is bounded by $\sum_{r=1}^{R'}$.

Step 1 & Step 2: Same as in Partial REALLOCATE.

Step 3: Holding the $R_j$ yielding $O_{(s+1)j}$ at $S_{s+1}, R_j \Rightarrow S_{s+1}, R' = R' + R_j, R = R - R_j$. Full REALLOCATE reiterates with a new $R_j \in R$ until either $O_x = O_{(s+1)x}$ or $R = \emptyset$.

Step 4: Compare $O^y = \min_j O_{(s+1)j}$ from the $y$th iteration, $y = x$ yielding
MIN(O_x?, O_{(x-1)}?), to O_o. If O_y?=O_o, O_o is the local optimum. If O_y?<O_o, update O_o to O_y?, R’ = R’ + R_j, R = R - R_j, R_j ⇒ S_{s+1}.

As in Partial REALLOCATE, Full REALLOCATE begins by computing δ(R_j → S_{s+1}) for each R_j ∈ R. Rather than outputting the MIN δ(R_j → S_{s+1}), Full REALLOCATE holds the R_j yielding O_1? at S_{s+1}, so R_j ⇒ S_{s+1}, R’ = R’ + R_j, R = R - R_j. Full REALLOCATE then reiterates with a new R_j ∈ R until either O_x?=O_{(x-1)}? or R = Ø. Full REALLOCATE iterates a greater number of times than Partial REALLOCATE before choosing from the yth iteration of the algorithm.

**EXAMPLE–COMPARISON OF PARTIAL, FULL REALLOCATE, EXHAUSTIVE SEARCH**

We provide an illustrative example to demonstrate the benefits of implementing the REALLOCATE algorithms over Exhaustive Search. We solve the same problem using each of the three data reallocation algorithms—first using Exhaustive Search, then using Partial REALLOCATE, and finally using Full REALLOCATE. Response times have been computed by the SimDDBMS simulator. The serial query strategy (Hevner & Yao, 1979) has been implemented and query results are computed as described in Blankinship (1991), Goyal (1994), Hevner and Yao (1979). Assume the system parameters in Table 4, the relations in Table 5, and the queries in Table 6 and the following relations, R = \{R_A, R_B, R_C, R_D\} (Table 6), and the following queries (Table 7).

**BASE CASE**

The base case assumes there is only one server, S = \{S_1\}, in the system and R_j’s ∈ R, R_j ⇒ S_1. Therefore, the initial allocation is R_A ⇒ S_1, R_B ⇒ S_1, R_C ⇒ S_1, and R_D ⇒ S_1. Using a poisson arrival rate for queries, SimDDBMS has computed a total response time of 307 CPU cycles. Since 307 is greater than the specified threshold of 200, a new server is added to the distributed database system.

*Table 5: System parameters*

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Rate</td>
<td>50</td>
</tr>
<tr>
<td>Network Transfer Rate</td>
<td>100</td>
</tr>
<tr>
<td>Threshold</td>
<td>200</td>
</tr>
<tr>
<td>Optimization Parameter</td>
<td>Response Time</td>
</tr>
</tbody>
</table>
Table 6: Relations

<table>
<thead>
<tr>
<th>Relation</th>
<th>Selectivity</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_A</td>
<td>1.0</td>
<td>1000</td>
</tr>
<tr>
<td>R_B</td>
<td>0.9</td>
<td>900</td>
</tr>
<tr>
<td>R_C</td>
<td>0.8</td>
<td>800</td>
</tr>
<tr>
<td>R_D</td>
<td>0.7</td>
<td>700</td>
</tr>
</tbody>
</table>

Table 7: Queries

<table>
<thead>
<tr>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Join A, B &amp; C</td>
</tr>
<tr>
<td>Join B &amp; D</td>
</tr>
<tr>
<td>Join A &amp; C</td>
</tr>
<tr>
<td>Join A, B, C &amp; D</td>
</tr>
</tbody>
</table>

EXHAUSTIVE SEARCH

S = \{S_1, S_2\}: (incremental growth from s=1 to s+1=2 servers)

There are 16 possible allocations of four relations to two servers (see Table 8).

The Exhaustive Search method finds the lowest response time of 216, with the allocations (R_A \rightarrow S_1), (R_B \rightarrow S_2), (R_C \rightarrow S_2), (R_D \rightarrow S_1) and (R_A \rightarrow S_2), (R_B \rightarrow S_1), (R_C \rightarrow S_1), (R_D \rightarrow S_2). The first allocation found is arbitrarily chosen as the new data allocation. So, R_B \Rightarrow S_2 and R_C \Rightarrow S_2. Since the minimum total response time found with two servers is greater than the specified threshold (216 > 200), an additional server is added to the distributed database system.

S = \{S_1, S_2, S_3\}: (incremental growth from s=2 to s+1=3 servers)

There are 81 possible allocations of four relations to three servers. The Exhaustive Search method finds the minimum response time of 182, with the allocations (R_A \rightarrow S_1), (R_B \rightarrow S_2), (R_C \rightarrow S_3), (R_D \rightarrow S_1) and (R_A \rightarrow S_2), (R_B \rightarrow S_3), (R_C \rightarrow S_2), (R_D \rightarrow S_1). Since the response time resulting from adding the third server meets the specified threshold (182 = 200),
Table 8: Exhaustive search solution with two servers, four relations

<table>
<thead>
<tr>
<th></th>
<th>RA</th>
<th>RB</th>
<th>RC</th>
<th>RD</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>307</td>
</tr>
<tr>
<td>2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>279</td>
</tr>
<tr>
<td>3)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>273</td>
</tr>
<tr>
<td>4)</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>242</td>
</tr>
<tr>
<td>5)</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>224</td>
</tr>
<tr>
<td>6)</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>267</td>
</tr>
<tr>
<td>7)</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>216</td>
</tr>
<tr>
<td>8)</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>244</td>
</tr>
<tr>
<td>9)</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>244</td>
</tr>
<tr>
<td>10)</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>216</td>
</tr>
<tr>
<td>11)</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>267</td>
</tr>
<tr>
<td>12)</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>224</td>
</tr>
<tr>
<td>13)</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>242</td>
</tr>
<tr>
<td>14)</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>273</td>
</tr>
<tr>
<td>15)</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>279</td>
</tr>
<tr>
<td>16)</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>307</td>
</tr>
</tbody>
</table>

no additional servers are added. The final data allocation found by Exhaustive Search is RA ⇒ S1, RB ⇒ S2, RC ⇒ S3, RD ⇒ S1 or RA ⇒ S1, RB ⇒ S3, RC ⇒ S2, RD ⇒ S1 with the total response time of 182 ticks.

**PARTIAL REALLOCATE**

S = \{S_1, S_2\}: (incremental growth from s=1 to s+1=2 servers)

There are R = 4 reallocation tests required. The response time is computed for each independent allocation (see Table 8). The Partial REALLOCATE algorithm finds the lowest response time of 224, with the allocation (RA → S1), (RB → S2), and (RC → S1), (RD → S1), so RB ⇒ S2. Since 224 > 200, an additional server is added to the distributed database system.

S = \{S_1, S_2, S_3\}: (incremental growth from s=2 to s+1=3 servers)

Again, R = 4 reallocation tests are required. The response time is computed for each independent allocation (see Table 10). The Partial REALLOCATE algorithm finds the
Table 9: Partial REALLOCATE solution with two servers, four relations

<table>
<thead>
<tr>
<th>R_j → S_2)</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_A → S_2)</td>
<td>2111</td>
<td>244</td>
</tr>
<tr>
<td>(R_B → S_2)</td>
<td>1211</td>
<td>224</td>
</tr>
<tr>
<td>(R_C → S_2)</td>
<td>1121</td>
<td>273</td>
</tr>
<tr>
<td>(R_D → S_2)</td>
<td>1112</td>
<td>279</td>
</tr>
</tbody>
</table>

Table 10: Partial REALLOCATE solution with three servers, four relations

<table>
<thead>
<tr>
<th>R_j → S_3)</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_A → S_3)</td>
<td>3211</td>
<td>188</td>
</tr>
<tr>
<td>(R_B → S_3)</td>
<td>1311</td>
<td>224</td>
</tr>
<tr>
<td>(R_C → S_3)</td>
<td>1231</td>
<td>182</td>
</tr>
<tr>
<td>(R_D → S_3)</td>
<td>1213</td>
<td>206</td>
</tr>
</tbody>
</table>

lowest response time of 182, with the allocation (R_A → S_1), (R_B → S_2), and (R_C → S_3), (R_D → S_1), so RC ⇒ S_2. Since 182 = 200, Partial REALLOCATE terminates. The final data allocation found by Partial REALLOCATE is R_A ⇒ S_1, R_B ⇒ S_2, R_C ⇒ S_3, R_D ⇒ S_1 with the total response time of 182 CPU cycles.

FULL REALLOCATE

S = \{S_1, S_2\}: (incremental growth from s=1 to s+1=2 servers)

There are a maximum of \( \sum_{r=1}^{R} r = 10 \) reallocation tests. In the first iteration, the response time is computed for R = 4 independent allocations (Table 11).

The Full REALLOCATE algorithm finds the lowest response time of 224, with the allocation (R_A → S_1), (R_B → S_2), and (R_C → S_1), (R_D → S_1), so R_C ⇒ S_2. Full REALLOCATE reiterates with R_A, R_C, and R_D.

The Full REALLOCATE algorithm finds the lowest response time of 216 with the allocation (R_A → S_1), (R_B → S_2), (R_C → S_2), (R_D → S_1), so R_C ⇒ S_2. Full REALLOCATE reiterates with R_A and R_D.

Since the minimum response time found in this iteration is greater than the response time in the previous iteration (244 > 216), Full REALLOCATE does not reiterate with two
Algorithm Development, Simulation Analysis and Parametric Studies

Table 11: Full REALLOCATE solution with two servers, four relations, first iteration

<table>
<thead>
<tr>
<th>Rj → S2)</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RA → S2)</td>
<td>2111</td>
<td>244</td>
</tr>
<tr>
<td>(RB → S2)</td>
<td>1211</td>
<td>224</td>
</tr>
<tr>
<td>(RC → S2)</td>
<td>1121</td>
<td>273</td>
</tr>
<tr>
<td>(RD → S2)</td>
<td>1112</td>
<td>279</td>
</tr>
</tbody>
</table>

Table 12: Full REALLOCATE solution with two servers, four relations, second iteration

<table>
<thead>
<tr>
<th>Rj → S2)</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RA → S2)</td>
<td>2211</td>
<td>242</td>
</tr>
<tr>
<td>(RC → S2)</td>
<td>1221</td>
<td>216</td>
</tr>
<tr>
<td>(RD → S2)</td>
<td>1212</td>
<td>267</td>
</tr>
</tbody>
</table>

Table 13: Full REALLOCATE solution with two servers, four relations, third iteration

<table>
<thead>
<tr>
<th>Rj → S2)</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RA → S2)</td>
<td>2221</td>
<td>279</td>
</tr>
<tr>
<td>(RD → S2)</td>
<td>1222</td>
<td>244</td>
</tr>
</tbody>
</table>

servers. Therefore, the allocation is (RA → S1), (RB → S2), (RC → S2), (RD → S1) with a response time of 216 time ticks. Since 216 > 200, an additional server is added to the distributed database system.

S = {S1, S2, S3}: (incremental growth from s=2 to s+1=3 servers)

Even with S = {S1, S2, S3}, there are still only ten maximum possible allocations with the Full REALLOCATE algorithm. We start with the best allocation (1221) from the result of Full REALLOCATE at two servers.

Again, we independently reallocate each Rj to Ss+1 and evaluate the resulting response time (Table 14).

The Full REALLOCATE algorithm finds the minimum response time of 182 time ticks with the allocations (RA → S1), (RB → S2), (RC → S2), (RD → S1) and (RA → S1), (RB → S2), (RC → S2), (RD → S1). Arbitrarily choosing the first allocation, RB → S3, we now test reallocating an additional Rj to S3 (Table 15).
Table 14: Full REALLOCATE solution with three servers, four relations, first iteration

<table>
<thead>
<tr>
<th>Relation</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_j → S_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(R_A → S_3)</td>
<td>3221</td>
<td>216</td>
</tr>
<tr>
<td>(R_B → S_3)</td>
<td>1321</td>
<td>182</td>
</tr>
<tr>
<td>(R_C → S_3)</td>
<td>1231</td>
<td>182</td>
</tr>
<tr>
<td>(R_D → S_3)</td>
<td>1223</td>
<td>216</td>
</tr>
</tbody>
</table>

Table 15: Full REALLOCATE solution with three servers, four relations, second iteration

<table>
<thead>
<tr>
<th>Relation</th>
<th>Allocation</th>
<th>RT</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_j → S_3)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(R_A → S_3)</td>
<td>3321</td>
<td>242</td>
</tr>
<tr>
<td>(R_C → S_3)</td>
<td>1331</td>
<td>216</td>
</tr>
<tr>
<td>(R_D → S_3)</td>
<td>1323</td>
<td>246</td>
</tr>
</tbody>
</table>

Since 216 > 182 and 182 < 200, Full REALLOCATE terminates. The final data allocation found by Full REALLOCATE is R_A ⇒ S_1, R_B ⇒ S_3, R_C ⇒ S_2, R_D ⇒ S_1 or R_A ⇒ S_1, R_B ⇒ S_2, R_C ⇒ S_3, R_D ⇒ S_1 with the total response time of 182 CPU cycles.

SUMMARY

In this example, we have demonstrated that for at least some problems, the Full and Partial REALLOCATE algorithms find the same optimal data allocation solution that is found using the Exhaustive Search algorithm. Additionally, we have shown that the Full and Partial REALLOCATE algorithms greatly reduce the problem search space, and hence, the cost of determining the new data allocation. In this particular example, the search space when incrementing from S = \{S_1\} to S = \{S_1, S_3\} was reduced by 75% using Partial REALLOCATE and 37.5% using Full REALLOCATE. Incrementing from S = \{S_1, S_3\} to S = \{S_1, S_2, S_3\}, the search space was reduced by 95% using Partial REALLOCATE and 87.6% using Full REALLOCATE. While Exhaustive Search and Partial REALLOCATE test 100% of their search space, Full REALLOCATE tested only 90% of its search space when incrementing from S = \{S_1\} to S = \{S_1, S_2\} and only 70% when incrementing from S = \{S_1, S_2\} to S = \{S_1, S_2, S_3\}. 
SIMULATION RESULTS

To simulate incremental growth and reallocation, including the implementation of Partial and Full REALLOCATE, we have developed SimDDBMS. We have run over 5,800 simulation experiments: 1,100 smaller, tractable problems have been run using the Exhaustive Search optimal, and 4,700 larger, more realistic problems have been run using only Full and Partial REALLOCATE. The experiments have been run to minimize system response time in the simple query environment.

A “base case” simulation is run for each problem. The “base case” assumes there is only one server in the distributed database system, with all of the relations allocated to this one server. (This parameter can be relaxed for distributed database systems consisting of more than one server.)

The cost delay for the specified optimization parameter of the base case is compared with the specified threshold (see Figure 2). If the threshold is not met, a new server is added into the distributed database system and the new data allocation is computed using the specified reallocation algorithm. This iterative process continues until the threshold is met or until the number of servers in the system equals the number relations (the latter parameter can be relaxed on a system housing partitioned data).

The Exhaustive Search algorithm is used for benchmark comparisons. However, due to the exponential growth in the search space of the Exhaustive Search algorithm, Exhaustive Search computations are restricted to a maximum of four servers and five relations (a combined search space of $1+32+243+1,024 = 1,300$ possible allocations). The effect on the system response time of each of the following parameters (as independent variables) is studied: CPU processing rate, network transfer rate, number of queries, and number of relations. In all graphs, each data point represents the average cost for 100 randomly generated problems. Fixed parameter values are displayed in a box at the top of each graph.

Effect of CPU Rate

As shown in Figure 3, the REALLOCATE algorithms performed very close to optimal across a broad range of CPU rates. We observed that at the lowest CPU rate, Full REALLOCATE was on average only 4.92% from the Exhaustive Search optimal with a worst case of 28.4% from optimal. Partial REALLOCATE deviated 6.66% on average from the Exhaustive Search optimal with a worst case of 32.53%.

As the CPU rate increased, both REALLOCATE algorithms achieved results even closer to optimal. While achieving close to optimal cost results, Full REALLOCATE reduced execution time on average by at least 94.5% when compared to Exhaustive Search, and Partial REALLOCATE reduced execution time on average by at least 96.5% compared to Exhaustive Search.

Figure 4 compares the results of Full and Partial REALLOCATE with a larger number of relations. Partial REALLOCATE actually performs better than Full REALLOCATE at the lower CPU rates and matches Full REALLOCATE at the higher CPU rates. However, closer inspection reveals that Partial REALLOCATE requires significantly more servers to achieve these results, often close to double the number of servers that Full REALLOCATE specifies. In Figure 5, we extract the results Partial REALLOCATE obtains at the same number of servers that Full REALLOCATE produces. At the lower CPU rates, Partial REALLOCATE does much worse in this comparison; however, as the CPU rate increases, Partial REALLOCATE obtains closer results. Since Partial REALLOCATE is server hungry, increasing the CPU rate per server makes this algorithm more efficient than the Full REALLOCATE algorithm.
**Figure 2: Simulation flow**

- **Inputs:**
  - Database Server
  - Network Topology
  - Database Relations
  - Queries, Threshold, Optimization Parameter, Reallocation Algorithm(s)

  "Base Case"

  Compute New Allocation
  (Exhaustive Search, Full REALLOCATE, Partial REALLOCATE)

  Yes

  Exceed Threshold?

  Yes

  # Servers = # Relations?

  No

  Final Database Servers Solution: Data Allocation

  No

  Yes

  **Effect of Network Transfer Rate**

  Figure 6 demonstrates the effect of varying network transfer rate. As with the effect of varying CPU rates, both Partial and Full REALLOCATE performed well against Exhaustive Search. At worst, Full REALLOCATE performed within 3% of optimal and Partial REALLOCATE within 2.7%. As network transfer rate increased beyond 100, the effect of network transfer rate became minimal.

  Figure 7 shows the performance of the Full and Partial REALLOCATE algorithms with larger problems. Partial REALLOCATE performed within 3% of Full REALLO-
Figure 3: Average response time with varying CPU rate: Exhaustive Search, Full REALLOCATE, Partial REALLOCATE

Figure 4: Average response time with varying CPU rate: Final results—Full REALLOCATE, Partial REALLOCATE
Figure 5: Average response time with varying CPU rate: Equivalent number of servers—Full REALLOCATE, Partial REALLOCATE

Figure 6: Average response time with varying network transfer rate: Exhaustive Search, Full REALLOCATE, Partial REALLOCATE
Figure 7: Average response time with varying network transfer rate: Final results—Full REALLOCATE, Partial REALLOCATE

CATE. Again, Partial REALLOCATE required significantly more servers (sometimes three times as many) than Full REALLOCATE in order to achieve the same results. Figure 8 shows Partial REALLOCATE’s performance at the same number of servers as Full REALLOCATE. Partial REALLOCATE consistently performed around 35% worse than Full REALLOCATE.

Both the Full and Partial REALLOCATE algorithms behaved nearly identical to optimal over a wide range of network transfer rates. Even with a larger number of relations, both Full and Partial REALLOCATE returned consistent results.

Effect of Number of queries

The close to optimal performance of the REALLOCATE algorithms when increasing the number of queries per problem is shown in Figure 9. With a large number of queries, Full REALLOCATE achieves solutions on average of only 3.3% worse than the Exhaustive Search optimal, while Partial REALLOCATE achieves solutions averaging 4% worse than optimal. Executing the REALLOCATE algorithms with a larger number of servers and relations, Figure 10 shows identical final cost results. Partial REALLOCATE, however, required a much larger number of servers than Full REALLOCATE.

Figure 11 compares the results of the REALLOCATE algorithms at an equivalent number of servers. Here, Partial REALLOCATE performs worse that Full REALLOCATE. With a small number of queries, Partial REALLOCATE averaged 15.9% worse than Full REALLOCATE; with a large number of queries, Partial REALLOCATE average 52.3% worse
Figure 8: Average response time with varying network transfer rate: Equivalent number of servers—Full REALLOCATE, Partial REALLOCATE

<table>
<thead>
<tr>
<th>Network Transfer Rate</th>
<th>Full REALLOCATE</th>
<th>Partial REALLOCATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1052</td>
<td>1052</td>
</tr>
<tr>
<td>10</td>
<td>932</td>
<td>346</td>
</tr>
<tr>
<td>50</td>
<td>305</td>
<td>305</td>
</tr>
<tr>
<td>100</td>
<td>301</td>
<td>301</td>
</tr>
</tbody>
</table>

(Avg # Servers = 2.14) (Avg # Servers = 3.50) (Avg # Servers = 3.78) (Avg # Servers = 3.82)

Figure 9: Average response time with varying number of queries: Exhaustive Search, Full REALLOCATE, Partial REALLOCATE

<table>
<thead>
<tr>
<th>Number of Queries</th>
<th>Exhaustive Search</th>
<th>Full REALLOCATE</th>
<th>Partial REALLOCATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>324</td>
<td>324</td>
<td>324</td>
</tr>
<tr>
<td>25</td>
<td>335</td>
<td>335</td>
<td>335</td>
</tr>
<tr>
<td>50</td>
<td>632</td>
<td>632</td>
<td>632</td>
</tr>
</tbody>
</table>
Figure 10: Average response time with varying number of queries: Final results—Full REALLOCATE, Partial REALLOCATE

![Average Response Time with Varying Number of Queries: Final Result - Varying Number of Servers](image)

Figure 11: Average response time with varying number of queries: Equivalent number of servers—Full REALLOCATE, Partial REALLOCATE

![Average Response Time with Varying Number of Queries: Equivalent Number of Servers](image)
than Full REALLOCATE.

**Effect of Number of Relations**

Figure 12 shows that varying the number of relations does not effect the comparative results of Partial REALLOCATE and Full REALLOCATE. Again, the final result is misleading since Partial REALLOCATE requires a greater number of servers than Full REALLOCATE in order to achieve the same results. Examining the response time with an equivalent number of servers in Figure 13 shows that Partial REALLOCATE’s response time grows linearly as the number of relations increase. Thus, the efficiency of Partial REALLOCATE drops per server as compared to Full REALLOCATE.

Figures 14 and 15 compare the execution times of the three data reallocation algorithms. The execution times have been collected on a NeXT workstation having a Motorola 68040 CPU running at 25mhz and 20 MG of RAM. The network transfer rate is varied in Figure 14 and the number of queries is varied in Figure 15 (varying the other system parameters showed similar results). Figure 14 shows that Full REALLOCATE in the best case, only requires 3.58% of the time required by Exhaustive Search, and in the worst case, only requires 7.73% of Exhaustive Search. Partial REALLOCATE, in the best case, only requires 1.74% of the time required by Exhaustive Search, and in the worst case, only requires 6.16% of Exhaustive Search’s time. Figure 15 shows similar results with varying number of queries. In the best case, Full REALLOCATE only requires 4.03% of Exhaustive Search’s time and only 5.75% in the worst case. Partial REALLOCATE varies from a low of

---

**Figure 12: Average response time with varying number of relations: Final Results–Full REALLOCATE, Partial REALLOCATE**

---
Figure 13: Average response time with varying number of relations: Equivalent number of servers—Full REALLOCATE, Partial REALLOCATE

Figure 14: Comparison of algorithm execution times with varying network transfer rate—Exhaustive Search, Full REALLOCATE, Partial REALLOCATE
1.78% to a high of 2.81% of Exhaustive Search’s time.

Tables 16 and 17, which correspond to the parameters in Figures 14 and 15, compare the average number of tests required for each of the three REALLOCATE algorithms. Each line in the table represents the average of 100 randomly generated problems.

**SUMMARY**

In summary, the Partial and Full REALLOCATE algorithms have been shown to considerably reduce problem search space, and hence, the cost of testing relation-server combinations. If the cost of each test is one unit, implementing Partial REALLOCATE over exhaustive search results in a cost savings of \( S^R - R \) units; implementing Full REALLOCATE over exhaustive search results in a cost savings of \( S^R - \sum_{r=1}^{R} I^r \) units.

Using SimDDBMS, parametric studies across a range of parameters, including CPU Rate, Network Transfer Rate, Number of Queries, and Number of Relations have been performed. The parametric studies have demonstrated the consistency of the REALLOCATE algorithms across a broad range of parametric values.
Table 16: Comparison of average number of algorithm tests with varying network transfer rate

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Network Transfer Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Partial REALLOCATE</td>
<td>7.02</td>
</tr>
<tr>
<td>Full REALLOCATE</td>
<td>11.37</td>
</tr>
<tr>
<td>Exhaustive Search</td>
<td>365.34</td>
</tr>
</tbody>
</table>

Table 17: Comparison of average number of algorithm tests with varying number of queries

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Partial REALLOCATE</td>
<td>6.29</td>
</tr>
<tr>
<td>Full REALLOCATE</td>
<td>13.85</td>
</tr>
<tr>
<td>Exhaustive Search</td>
<td>862.63</td>
</tr>
</tbody>
</table>
Although both Partial and Full REALLOCATE provide good approximations to Exhaustive Search optimaums, each algorithm has a different strength. As shown in Table 18, if the cost of testing relation-server combinations is high, Partial REALLOCATE is the algorithm of choice. This is because Partial REALLOCATE has a much smaller search space than Full REALLOCATE ($R \leq \sum_{r=1}^{R} r'$, where $R$ is the number of relations in the distributed database system). If the cost of adding additional servers is high, Full REALLOCATE is the algorithm of choice. As demonstrated in the simulation experiments, Partial REALLOCATE is a server-hungry algorithm. It generally requires two to three times as many servers as Full REALLOCATE in order to find a comparable solution.

**ADDITIONAL APPLICATIONS**

The REALLOCATE algorithms in conjunction with SimDDBMS can be used to answer many practical questions in distributed database systems. For example, in order to improve system response time, a DDBMS administrator may use SimDDBMS to analyze the effect of upgrading CPU-processing capability instead of adding additional servers into the distributed database system. Additionally, SimDDBMS may be modified to allow for heterogeneous servers, with different CPU processing capabilities, in the distributed system. A DDBMS administrator may also use SimDDBMS to determine the cost-benefit analysis of adding some number, $W \geq 1$, of additional servers at one time.

**CONCLUSIONS AND FUTURE RESEARCH**

We have presented an incremental growth framework and two data reallocation heuristics — Partial REALLOCATE and Full REALLOCATE — for incremental growth and reallocation in distributed database systems. Through simple examples and then through parametric studies performed using the SimDDBMS simulator, we have demonstrated the robustness of both data reallocation heuristics. Due to their linear complexity, the Partial and Full REALLOCATE algorithms can be used for large, complex problems while achieving good results as compared to the Exhaustive Search optimal.

<table>
<thead>
<tr>
<th>Cost</th>
<th>Level</th>
<th>Algorithm of Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation-server testing</td>
<td>High</td>
<td>Partial</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Partial, Full</td>
</tr>
<tr>
<td>Additional Server</td>
<td>High</td>
<td>Full</td>
</tr>
<tr>
<td></td>
<td>Low</td>
<td>Partial, Full</td>
</tr>
</tbody>
</table>
The concepts, algorithms, and software application developed in this research can easily be adapted and extended to other problems in distributed database research. In this section, we discuss some interesting extensions of this work.

**Extended Incremental Growth**

This work assumes growth in a distributed database system is at a rate of one additional server at a time. Relaxing this assumption and allowing system growth at a rate of \( \theta \) additional servers at a time, where \( \theta = 1 \) will prove interesting. The Full and Partial REALLOCATE algorithms can be modified to accommodate growth in varying increments.

**Data Replication**

Our research assumes only one copy of each relation has been allocated to the distributed database system. Although back-up copies may exist, only one copy—the primary copy (Garcia-Molina & Abbott, 1987; Son 1988)—is used in transaction processing. An interesting extension of this work would be to allow multiple copies of relations to be stored in the distributed database system. Duplicate copies may be used to reduce processing time as well as to guard against loss of data in case of server failures. This enhancement would add considerable complexity to the problem.

**Fault Tolerance**

This research assumes network and server reliability. An interesting extension of this work is to implement the Incremental Growth methodology in conjunction with fault tolerance algorithms. This would provide an additional level of integration of distributed database technologies.

**Incremental Reduction**

This research concentrates only on system growth. An interesting research topic is to use the concepts of the iterative Incremental Growth methodology for incremental system reduction, i.e., decrease the number of servers in the distributed database system from \( \theta \) servers to \( \theta - 1 \) servers. This research could provide an integration between data replication, data reallocation and reorganization, and fault tolerance.

**Data Fragmentation**

This research assumes relations are allocated in their entirety to their assigned server. Expanding the REALLOCATE algorithms to encompass data fragmentation over some number of servers would be especially useful research. This would add some additional complexity to the REALLOCATE algorithms.

**REFERENCES**


Chapter X

Using Weakly Structured Documents at the User-Interface Level to Fill in a Classical Database

Frederique Laforest and André Flory
National Institute of Applied Sciences, France

Electronic documents have become a universal way of communication due to Web expansion. But using structured information stored in databases is still essential for data coherence management, querying facilities, etc. We thus face a classical problem—known as “impedance mismatch” in the database world; two antagonist approaches have to collab-
orate. Using documents at the end-user interface level provides simplicity and flexibility. But it is possible to take documents as data sources only if helped by a human being; automatic document analysis systems have a significant error rate. Databases are an alternative as semantics and format of information are strict; queries via SQL provide 100% correct responses. The aim of this work is to provide a system that associates document capture freedom with database storage structure.

The system we propose does not intend to be universal. It can be used in specific cases where people usually work with technical documents dedicated to a particular domain. Our examples concern medicine and more explicitly medical records. Computerization has very often been rejected by physicians because it necessitates too much standardization and form-based user interfaces are not easily adapted to their daily practice. In this domain, we think that this study provides a viable alternative approach. This system offers freedom to doctors; they would fill in documents with the information they want to store, in a convenient order and in a freer way. We have developed a system that allows a database to fill in quasi-automatically from documents paragraphs.

The database used is an already existing database that can be queried in a classical way for statistical studies or epidemiological purposes. In this system, the document fund and the database containing extractions from documents coexist. Queries are sent to the database, answers include data from the database and references to source documents.
INTRODUCTION

Information capture is an important barrier for end-users software acceptance. In domains where the end-user is not compelled to use a computer or is demanding because of activity constraints, classical computerized systems have difficulty being accepted and widely used. Difficulties are more accurate when documents are the paradigm used to manipulate information. One can find many domains of this type: lawyers, doctors, etc., who use technical documents to store the information they need. These are domains where computerization is particularly little used. The example of the medical domain is obvious. Doctors are not satisfied by today’s systems and prefer using paper-based medical records. Many trials have been and are still conducted in this field, but success has not completely come. The main barrier concerns information capture speed and facilities compared to computerized systems advantages. Capture forms have been chosen by computer scientists because they have the great advantage providing important querying capacities, as they are most often easily related to a database. Capture forms do not satisfy physicians as they cannot adapt to each case encountered. Forms impose data to be captured, the order in which to capture, and a strict format for each data.

With the current prevalence of the Web, and consequently, of electronic documents, the next idea that comes is to use electronic documents as a basis for the system. This idea has the first advantage of removing the mismatch between paper-based documents and capture forms. Information is captured in electronic documents and queries on documents can be made using a dedicated document querying language. To go forward in this idea, we have to note that one can find three different types of documents:

- **Free-text documents** only contain information and formatting instructions (these instructions are also known as physical structure). These documents are very easy to write, but very difficult to query. Free-text analysis is still a research domain; the results are not yet satisfying enough to be used in sensitive domains. Compared to paper-based documents, systems based on free-text documents still do not provide enough precision to be widely used.

- **Strongly structured documents** contain information and semantics guides (also known as logical structure). These guides are most of the time represented by tags circling information pieces. SGML (ISO, 1986) documents are good examples of such documents. These documents set a structure that the user has to follow. This structure is defined in a Document Type Definition (DTD) that provides tags, composition rules of tags, compulsory tags, attributes for tags, etc. This structure is not as rigorous as forms structure; no format is imposed for data. Moreover, only effectively captured information appears in each document. In forms, all fields are present even if not filled for the current case. Queries on strongly structured documents can be made using dedicated query languages like SgmlQL (LeMaitre, Murisasco, & Rolbert, 1998) or UnQL (Buneman, Davidson, & Hillebrand, 1996). These languages are currently under the form of prototypes and answers lack precision. In systems that link strongly structured documents to a database, each information to be stored in the database is tightly tagged so that there is a one-to-one relationship between data in the database and tagged information in documents. The database stores a representation of the tree structure of the document, without any treatment on information pieces; filling a database is thus rather easy but does not provide the same facilities as a real information system database filled through a form; queries are not as precise as queries on atomic data. This approach still does not satisfy end-users,
as writing such documents is time consuming and too much constraining. As each stored
information piece has to be tightly tagged, the DTD looks much like a disguised form.

- Weakly structured documents may be seen as an intermediate level between free-text
documents and strongly structured documents. Weakly structured documents use
DTDs containing optional tags (compulsory tags are defined only to contain the
identification of the document subject, e.g., patient and doctor IDs). Most tags
delimitate paragraphs rather than data. A paragraph contains free text which may
include many data. For example, a prescription paragraph contains one sentence in
which one can find a medication name, a dose, a frequency and a duration. That is the
type of documents our system manages. To query data stored in such documents, we
link the DTD to a relational database. In each paragraph one can find information that
should belong to the database or not. These documents have a double advantage: (1)
to be easily captured and read by the end-user, and (2) to be semantically more precise
than free text, thus easier to analyze automatically.

In the following of this chapter, we use the medical record for illustrations. Here is a rapid
tour of our system. The end-user captures information in a weakly structured document. He tags
paragraphs according to a DTD conforming to his application domain. For example, tags can be
<prescription>, <past history>, <diagnosis>, etc. Once a document is validated by the end-user,
analyzer extracts data from the tagged paragraphs, builds SQL queries and sends its results
to a database. At the opposite of usual document databases, the database contains a classical
information system database (tables like “patient” and “prescription,” no “paragraph” or
“keyword” tables). For each tagged paragraph, the analyzer can extract many data to be stored
in many tables. Relationships between paragraphs and the database are stored under the form of
patterns to be recognized and links to the database. These patterns are defined a priori by the
application designer at the same time as the DTD.

The medical record domain seems a good candidate for this system for many reasons.
The first is that medicine is still an art that cannot be strongly structured, so that no form-
-based user interfaces are adequate. Documents are used in the daily work. For chronic
diseases, records rapidly become thick; searching information looks like mining. Doctors are
interested in automated systems that could help in information retrieval for the care of a patient
or for statistical purposes. They also appreciate security improvements (e.g., prescriptions
validation). The second reason is that the language used by physicians has already been
classified. There exists many classifications, ontologies and thesauri referencing the medical
language. This is an advantage for the analyzer that has referentials to identify data.
Classifications of diseases have been written for years, medications thesauri can be found
in any country, acts are listed in social security files. The third reason is that many paragraphs
have standard ways of writing, and patterns can be defined for a quite large use. For example,
the prescription paragraph should include a medication name (from a formulary), a duration
(always followed by a duration unit and very often preceded by a word like “during”), a
frequency (always followed by a frequency unit) and so on. Each sub-part of the prescription
sentence can be represented by a few patterns to search.

This chapter presents in a deeper way the system we defined. The next section gives
the position of our work compared to the literature. Next, a system overview presents the
differences between a classical document database and our system and then presents
answers to the main questions one can think of. After that, we give the architecture of our
system using the UML formalism. The next section provides a description of the first
prototype we have written and an analysis example based on a medical record. We then
conclude with the current progress state of this project.
BACKGROUND

Much has been done on documents in the literature. Querying free-text documents is a difficult task that has been studied for years. Recall and precision are difficult to ensure and balance (Salton, 1986). Research activities in this domain are still numerous. A lot of studies are dedicated to a specific application domain, like medicine (Blanquet & Zweigenbaum, 1999), law (Jackson, Al-Kofahi, Kreilick, & Grom, 1998), or automobile ads (Embley, Campbell, & Jiang, 1998a). Passage retrieval (Kaszkiel & Zobel, 1997), excerpts and resume construction (Goldstein, Kantrowitz, & Carbonell, 1999), ontology-based indexing (Stairmand, 1997) are some of the directions currently studied to improve free-text mastering.

Studies on structured documents rapidly grew with the arrival of SGML (ISO, 1986). SGML is well adapted to documentation structuring and is widely used in technical documentation. DTDs propose many mechanisms to define structuring rules for documents. Systems to query SGML documents are both using words, as in free-text, but also the document structure seen as a graph. Some languages use pattern-based search algorithms (Laforest, 1999), (Lakshmanan, 1996), (Baeza, 1996) based on tree representations of documents. Some propose a SQL-like user interface (Neven, 2000), (Kilpelainen, 1993).

Many structured documents querying languages use this principle (Lemaitre, 1998), (Goldman, 1997), (Buneman, 1996), (Deutsch, 1999a), (Mendelzon, 1997), (Konopnicki, 1998), (McHugh, 1997), (Abiteboul, 1997a), (Abiteboul, 1997b) for querying documents written in any structuring language.

Another way to query structured documents is to store information pieces into a database. Some have adapted HTML by adding specific database tags (Dobson, 1995). Others have adapted the database to documents (Christophides, 1994), (Stonebraker, 1983), (Cluet, 1999), (Cohen, 1998), (Bohm, 1997), (Atzeni, 1997), (Deutsch, 1999b), (Abiteboul, 2000). The database contains tables like “paragraph”, “section”… that do not provide any semantics to the information stored. A variant of these approaches proposed in (Frenot, 1999) uses XML semantic tags adapted to an application domain so that queries are contextually oriented by tags semantics. Storing documents in such a database does not allow to take advantage of SQL. In these approaches, columns of the database do not contain formatted data but text pieces. Queries cannot thus be made on a standard way, exact matching and joins have to be re-defined and are difficult to implement. These systems also index text pieces. Using indexes follows the same idea as using indexing databases on file-stored documents.

Structured maps (Delcambre, 1997) propose an additional modeling construct that provides structured and managed access to data. They are based on Topic Navigation Maps (Biezunski, 1995). The main idea is to provide a database that is manually filled to index information pieces in documents. The database is not generic but domain related (e.g. “painter” and “painting” tables). It allows classical querying on the database. Each data in the database is related to the source information piece so that the database answers provide the linked text pieces. The system also provides topic navigation maps, i.e. documents that refer to all documents concerning a topic.

In (Riahi, 1998) and (Embley, 1998a), one can find an approach which is closer to ours, even if dedicated to HTML documents. The main idea is to build a real application domain database associated to an important “classical” indexing feature, so that queries are sent to the database rather than processed on documents. In (Riahi, 1998) the database is filled using the very few semantic tags offered by HTML (keyword, author) and mainly using formatting information: it uses the geographical location of tags to “guess” data semantics. Rules to extract information from documents have to be written for each type of document, i.e. for
each site. Moreover, each site has to ensure presentation coherence rules for all documents. The problem faced by this project is mainly due to HTML and its lack of semantics. The model of the database is highly dependent on document contents and it seems very difficult to define one general model that would satisfy an application domain treated by many different independent Web sites. Embley, Campbell, Smith, and Liddle (1998a) use an ontology to describe each data with all its essential properties. The database to be filled is automatically created according to the ontology structure. A main concept in the database represents the main subject of the data (e.g., one person in an obituary). Documents are separated into different records, each record concerning one main subject instance. Data properties mainly include PERL5-based rules patterns to extract data from HTML text, as well as relationships between the main subject of a record and all its data. This project is interesting for the capture of data into databases that contain few tables, in other words in domain that are “narrow in ontological breadth” (Embley, Campbell, & Jiang, 1998b).

As a conclusion, we can say that documents are more and more used in electronic systems. But systems that include documents management do not see them as a new way to capture data. Documents contain information for the end-user. Systems offer information retrieval techniques to search for documents, but do not provide complete data management. The type of documents implies different indexing techniques, but queries made by end-users are always of the type “give me documents that …” Our proposal goes further—we use documents as data sources, so that we can fill in a database. Data in the database can be searched for themselves, independently of source documents. Queries made by end-users can be of the form “select data from databaseTables where…”

We have found very few proposals dealing with this issue. Some use manual database filling. Others are based on HTML documents and face much difficulties as HTML is not semantically tagged. The last ones use XML strongly structured documents, that are dedicated to system-to-system data interchange but cannot be written by end-users. We propose to use XML weakly structured documents and to take advantage of an application domain where the language is already classified in thesauri, databanks or classifications.

THE DRUID SYSTEM

Issues and First View of the Proposed Solution

The Documents- and Rules-based User Interface for Databases (DRUID) system intends to use weakly structured documents as a user interface paradigm. Internal rules are dedicated to extract information from these documents and thus to fill in a database. Documents are seen as a new way to capture data, and not as an additional system parallel to the classical information system.

Classical Documents Databases Versus Our System

Classical systems that associate a database and a documentary fund are not dedicated to manage data but to manage the documentary fund. Their aim is to provide rapid access to documents and tools to select documents. They do not intend to provide an application domain model nor to use documents as a way to fill in an application domain model. They work as follows (Figure 1):

• documents are written in a non-structured, strongly structured or weakly structured manner, in accordance with the rules of the enterprise. The document file is stored in the documentary fund (often the file system of a server).
• the author or a librarian enters indexing information concerning the new document in
the database using capture forms. These forms ask for contextual information on the
document; location, author references, date, version number and key-words are the
most usual data captured. The database contains this indexing information. It does not
contain the domain model. We call it an indexing database.
• searching for a document is done by querying the indexing database. This query does
not concern the document content but captured contextual information. The query
returns the list of documents corresponding to it.
• some systems add to this manual referencing an automatic system. This indexing can
be based on the frequency appearance of words in the document. In HTML, the META
section allows the document writer to define key-words on his document. Indexing
systems on the Web use such techniques.

These systems do not consider documents as data sources for the computerized system,
but as information sources for the end-user. The database is an indexing database. It permits
the end-user to access the documents that contain information, but it does not allow a direct
access to application domain information. The indexing database can be seen as a dictionary
containing metadata on a documents fund. It contains tables and attributes like “document
name,” “document id,” “author,” “authoring date,” “section title,” etc., that are comparable
to the dictionary of databases that contain tables and attributes like “table name,” “table key,”
“author,” “authoring date,” etc. These elements do not depend on an application domain.
The database is orthogonal to application domains and can be multi-domain.

The system we propose is totally different. It uses:
• a classical information system database (the model represents data of the application
domain),
• weakly structured documents instead of classical forms to capture information,
• an analyzer to fill in the database through an automated analysis of the captured
documents, and
• queries that can concern the database itself, as it is classically done, or can ask for
source documents.

Our system has to drop the fastidious manual step of database update through a
capture form, by automating this step using an automatic analyzer (Figure 2). The

Figure 1: Classical systems associating a documentary fund and an indexing database
database contains both contextual information and semantically extracted information. The extraction of information does not concern all information in the document, but only information that has a correspondence in the database. We propose to capture weakly structured documents, so that documents are quite easy to capture and the extraction mechanism is guided by semantic tags.

The following paragraphs give an overview of the principles of the DRUID system. They are detailed in the next section throughout the meta-model and the software architecture used.

**How Documents Are Built Within DRUID**

We have chosen the XML (W3C, 1998b) language to build weakly structured documents. With this language, the user builds documents by including free pieces of text quoted by tags. A DTD provides the definition of tags allowed and of their composition relationships. Tags depend on the application domain and on the related database.

One can find two types of XML documents (Figure 3):

- Strongly structured documents tightly tag each information piece that have to be inserted in the database. Such documents can be classified as strongly structured documents, since all information for the database has to be tagged and correctly formatted. They can also be seen as flexible forms, as they allow putting free text around structured data.
- Weakly structured documents that tag paragraphs, so that a tag does not quote a data, but a piece of free text that needs to be analyzed to get data for the database. DRUID follows this case.

In this work, the application domain is the electronic medical record. The database was previously defined. We built the DTD manually, in accordance with the dedicated domain users’ habits and with the database structure. We noticed that documents are divided into paragraphs that can be classified according to their semantics, and we noticed tables or subsets of attributes corresponding to each paragraph. We thus decided to define one tag for each paragraph type. We provide an example in the prototype section.

**How DRUID Goes From Information Pieces to Data**

We need a correspondence between the tagged paragraphs and the attributes that may be filled in the database from these paragraphs. Correspondences are stored in a meta-model

*Figure 2: Our proposal based on automated database update from weakly structured documents*
Using Weakly Structured Documents at the User-Interface Level

Figure 3: A strongly structured document followed by a weakly structured document (information for the database in bold)

A first step is based on pattern matching rules. Some pieces of text correspond to patterns, from which a value may be extracted. For example, the duration of a prescription is searched in the following patterns: `<during x days>` or `<during x months>`. In fact, patterns are composed of possibly coupling pieces of text from thesauri. In our example the pattern looks more like `<“during” x duration-unit>` where duration-unit takes its values in a thesaurus containing all possible values.

A formatting phase then translates the detected text into the appropriate attribute format.

How DRUID Fills in the Database

When a user finishes writing a document and validates it, the analyzer is launched. It parses the document and uses the mapping model to search for the values to put into the database. Once attributes have been detected and formatted, SQL queries are built and sent to the database.

Some additional information is necessary to identify the database table key values for the new attribute value. Most of the time, these values are given at the beginning of the document (a first Id paragraph is often written), or in the file system (name, location, creation date). Recognition of these key values is determinant for the efficiency of our system. At the moment, we define compulsory tags in each DTD to include key information that is not managed by the file system.

DRUID Architecture

Meta-Model

The schema on Figure 4 provides a UML class diagram of the meta-model used to extract information from the document and to insert them into the database.

We have divided this meta-model into three packages: the database world, the document world, and the mapping world to associate the two previous worlds.

The document world contains documents produced, DTD used to produce the documents, and tags defined in the DTDs. Documents and DTDs are located with their URLs. Each tag belongs to one DTD, has a label, and may be compulsory in the documents produced. There may be composition relationships between tags.

The database world refers to a simplified relational database dictionary. It represents tables and their attributes, noticing keys and foreign key relationships. A counter key is an
attribute whose value is incremented for each new occurrence without any semantic meaning. A specific URL attribute is added to each database table that may be filled from documents. It refers to the document from which data have been extracted.

The mapping world is the center of the system. It requires more attention; it contains all information necessary to determine which information to search in which tagged paragraph and to fill which database attribute.

A pattern is composed of an ordered list of pattern elements, each of which is extracted from a thesaurus (detailed later). A pattern also contains an SQL query to be built using the data found. For example, to insert the name of a medication from a “prescription” tagged paragraph, the pattern refers to a drug thesaurus, and the SQL query is something like `insert into PRESCRIPTION values ($patientcode, $date, $1)`. $patientcode and $date are key data extracted from the document and kept in mind to store other data in tables that use these key data as key attributes. $1 refers to the drugs thesaurus value found in the paragraph. Some attributes may be filled many times from one paragraph; it is allowed when the `one/many values?` attribute is set to MANY.

We have defined three types of thesauri:

1. File thesauri concern thesauri stored as files. These files contain one or two columns. One column contains patterns to be searched, the second (when existing) provides the SQL query.

Figure 4: Meta model to connect a database to a document capture system
code of the pattern. For example, in a medication file, the pattern column contains medications names while the code column contains medications identifiers. To search information in such thesauri, the system has to search for a pattern in the paragraph that corresponds to a label in the file. Most of the time, the database does not contain the label, but the identifier. When needed, the code? attribute of the fills association is set to true, and the code of the detected label is put into the database rather than the label.

2. Table thesauri concern thesauri stored as tables in the database. Such thesauri are similar to file thesauri. A label is given as an attribute of the table; code is the key of the table.

3. Value thesauri refer to values. These are virtual thesauri that correspond to numbers, dates or strings. They cannot be extracted from any thesaurus but have to be put into the database. For example, the number of pills to be taken, the duration of the prescription, the name of the patient, etc.

Here is an example of a composite pattern from the medical domain. The duration of a prescription contains three parts: the pre-duration expression (during, to be taken, etc.), the duration value (number) and the duration unit (hours, days, weeks, months, etc.). Each part comes from a different thesaurus. All the possible values for the pre-duration expression are stored in a file. The duration value comes from a value thesaurus of integer type. The possible expressions for the duration unit are stored in a second file. The pre-duration expression is used to point at the text corresponding to the duration value, but it will not be stored in the database. On the other hand, the duration value and the duration unit are stored in the database. The duration value is stored in the database as it is. The duration unit is located using the label column of the duration units file; it is stored in the database using the code column of the same file.

Software Architecture

The schema on Figure 5 presents the 3-tier architecture of our system, using the UML components diagram.

1. The Storage component includes:
   - the information database to be filled and queried;
   - the documents fund containing all documents created by end-users as well as DTD files; and
   - the database management system, a classical relational DBMS without any modification. It manages the information database, the dictionary of this database. It can manage the mapping world base if stored as a relational database.

2. The User interface component manages all interaction with the end-user; creating documents, querying documents, reading documents are the three main interaction scenarios. It has connections with the Mapping component for the integration of new documents, with the DBMS for querying data and existing documents (get URL), with the documents fund for reading documents.

3. The Mapping component is the core of our system and we have divided it into three sub-components:
   - the mapping world base refers to the mapping world part of the meta-model mentioned above. It contains references to all thesauri available, and classes containing the patterns to search and the SQL queries to generate. This mapping world base can be stored as a relational database (and thus can be managed by the DBMS), or as structured files.
• the Document analyzer has to analyze documents and find new values for data. It may have to ask for help from the end-user if some compulsory information is not automatically detected in the document. It sends two results: first, a strongly structured document that is the written document plus tags around each data found, and second, a list of SQL queries for the Mapping-DBMS component. It also communicates with the Mapping-DBMS component to get information about the database dictionary and the mapping world if stored in the DBMS.
• the Mapping-DBMS interface manages the communication between the document analyzer and the DBMS. It asks for information concerning the Database dictionary and the Mapping world (as part of the meta-model). It transfers detected information to the database via SQL queries. It manages connection sessions with the DBMS. It also ensures that queries sent to the database are sent in an order compatible with foreign key references and other constraints on the database.

Document Analyzer Algorithm

The document analyzer algorithm provides the steps followed by the document analyzer component each time a new document is validated. Its algorithm is divided into two main parts: paragraphs location and queries generation. We present in Figure 6 and Figure 7 a macro-algorithm for each of these two parts.

Prototype

Implementation

We have developed a prototype of this system. This prototype has been developed with the Java language, XML and Enterprise Java Beans (EJB) (Sun, 2001) as major tools. We have used the IBM Alphaworks XML parser (Alphaworks, 2000) to parse the XML documents and identify paragraphs. This parser follows the DOM specification (W3C, 1998a) and the SAX 1.0 specification (Megginson, 1998). Enterprise Java Beans have been used to build the 3-tier architecture. We have used the Weblogic (BEA, 2001) EJB platform.
• The document base contains XML documents and DTDs on the file system of the computer.

Figure 5: Software architecture

![Software architecture diagram]
get the meta-model information - if inconsistencies detected, stop
parse the document into a DOM representation
for each paragraph of the document
  if this paragraph contains sub-paragraphs, recur on sub-paragraphs
  else
    get the tag and verify that it exists in the meta-model
    if it does not exist ignore it
    else provide the paragraph to the queries generation algorithm
    get the resulting queries and strongly-structured document
if a compulsory tag has not been encountered, generate an error
if not, provide the strongly-structured document to the end-user for validation.
on return, transmit all queries to the Mapping-DBMS interface component

Figure 7: Queries generation macro-algorithm
get the pattern information corresponding to the tag under processing
prepare a new tuple to be filled by this process according to the table it refers to
get key attributes values:
  if it is a counter, generate a new value for this counter
  if it is a value coming from another paragraph,
    get the tag name and copy the value it tags
    verify that the key value does not already exist in the table
get an attribute value:
  get the related thesaurus
  find in the paragraph all expressions corresponding to the thesaurus
  if many expressions found and not many values possible
    generate an error
  if no expression found but compulsory, generate an error
  for each expression found, code it if necessary and tag in the document with the column name in the database
  if an insert query has to be generated, generate the query - if many, generate as many as needed
  if an update query has to be generated, verify that the occurrence to be modified exists in the database; if it exists generate the update query; otherwise generate an insert query - if many, generate as many as needed
return the query / queries
The test database we have used is presented in the following section. It is an excerpt of a real medical record database. We have used the Cloudscape (Informix, 2001) relational database management system. Access to the DBMS is made through Entity Enterprise Java Beans, which transparently manages concurrent access and synchronization between data in the database and the software.

The Mapping world base is stored in XML documents. We have defined a XML DTD to define the meta-model of the Mapping world base. Thesauri are all stored as file thesauri (except value thesauri) containing one column (label) and sometimes a second column (code).

The User interface component has been implemented. It is presented in more detail in the next section of this chapter. It contains both classes that are clients of EJBs and some EJBs dedicated to end-user connection and other such stuff.

The Mapping component is the core of our prototype. It analyzes XML documents and generates SQL queries in accordance with the macro-algorithms provided above. Today, it can only do exact matching. We are studying existing systems that allow non-exact matching. It has been implemented as EJBs. It also presents generated queries in a dedicated window, for debugging purposes.

We have chosen to implement the mapping world as XML documents, mainly to test XML. We have defined a DTD for thesaurus description. Thanks to this DTD, we can create documents describing the thesauri used in each particular application.

We have also defined a DTD for mapping documents. Each DTD for the end-user is associated to a mapping document that explains the mapping for each tag in the end-user DTD. Each tag in the end-user DTD that has a link to the database is associated with a pattern in the mapping document.

We do not provide these DTDs in this chapter because they are quite long and necessitate a lot of explanations.

Example On Medical Patient Record

This paragraph presents an example of the work of our prototype on a medical patient record. We do not provide all XML documents produced (thesauri XML description and mapping documents) because they are long to explain.

The medical record database used is a simplified version of a real relational medical records database. It contains the following tables (Figure 8).

A Patient is identified by a number. We also store his name and family name. The URL attribute refers to the documents used to provide the occurrence.

An Encounter occurrence gathers all information noticed during an encounter between the patient and a practitioner. It references the patient encountered (patient Id), provides the date of the encounter and the URL of the document produced concerning this encounter.

Figure 8: Database model (Underlined attributes take part of the key of the table. Italic attributes are foreign keys.)
Other encounter information are stored in two other tables: the Problem table and the Prescription table.

The Problem table contains all the diseases detected during encounters. The diseaseCode is a reference to the diseaseCode attribute of the disease table.

The Prescription table refers to the encounterId of the corresponding encounter. It also refers to the medication prescribed, and provides posology information (duration, frequency, dose).

Based on this database model and on handwritten documents, we have defined the DTD in Figure 9 for encounter documents. Each time a physician meets a patient, he writes a document of this type.

Figure 10 presents an encounter document written by a physician. It follows the encounter DTD given above. This document is not captured as is: the end-user captures it in the end-user interface that hides the XML structure of information, as presented below. The analysis of this document by our prototype provides six SQL queries. They are given in Figure 11. Diseases are coded using the ICPC classification (WONCA, 1997) and medications using an excerpt of a French drug databank (Flory, Paultre, & Veilleraud, 1983).

**Prototype of the End-User Interface**

We have developed a prototype of the end-user interface. This interface allows him:

- to identify himself: the end-user selects his references in the window in Figure 12, validates and provides his password within the following window provided on the same figure.
- to write a new document about a patient and send it to the analyzer: first the end-user chooses the patient and the document type he wants to write (Figure 13). The document type allows the system to get the corresponding DTD. Secondly, the end-user writes the document itself and tags paragraphs (see window in Figure 14). The left part of the window contains the structure of the document, i.e., paragraphs tags. The right part presents information as captured by the end-user (capture of free-text paragraphs).

**Figure 9: DTD for encounter documents**

```xml
<?xml version='1.0' encoding='ISO-8859-1' standalone='no' ?>
<!DOCTYPE encounter SYSTEM "encounter.dtd">
<encounter>
  <patient_id>2</patient_id> <date>11-06-1999</date>
  <Problem>The problem concerns the left eye. eye pain. red eye. Afraid of eye illness.</Problem>
  <prescription>Visiodose every 2 days during 7 days</prescription>
  <prescription>1 pill A Vitamin (1 week)</prescription>
  <comment>verify the eye does not change aspect</comment>
</encounter>
```

**Figure 10: An encounter document**

```xml
<?xml version='1.0' encoding='ISO-8859-1' standalone='no' ?>
<!DOCTYPE encounter SYSTEM "encounter.dtd">
<encounter>
  <patient_id>2</patient_id> <date>11-06-1999</date>
  <Problem>The problem concerns the left eye. eye pain. red eye. Afraid of eye illness.</Problem>
  <prescription>Visiodose every 2 days during 7 days</prescription>
  <prescription>1 pill A Vitamin (1 week)</prescription>
  <comment>verify the eye does not change aspect</comment>
</encounter>
```
When a paragraph is written, the end-user selects the paragraph tag using a pop-up menu (tags proposed depend on the document type). Patient id and date are the identifiers of the document; they are not captured at this time but come from the previous window. When the end-user has finished writing his document, he clicks on the button on the right of the buttons bar to send it to the analyzer.

• to validate what the analyzer has recognized in the document and allow it to send data to the DBMS: Figure 15 shows the window used by the end-user when the analyzer has finished the document analysis. This document contains the same information and paragraph tags as the original document. Other tags have been added around each piece of information detected by the analyzer. The end-user can make modifications on the analysis by using a popup menu. This menu allows the end-user to delete, add or change a tag in respect of the corresponding DTD. When the end-user clicks on the right button of the buttons bar, the analyzer sends insert queries to the database.
to search for existing documents: we have rapidly developed a window to search for information in the database. This window is not appropriate for a naive end-user, but allows us to verify the content of the database in the prototyping stage. Its ergonomic should by significantly improved. On the left part, one can find tables and their attributes in the database. The user builds his query by using “drag & drop” onto the right part of the window. A click on the “execute” button builds the corresponding SQL query, sends it to the database, and shows results as in Figure 17. The result of the query is seen in a “standard” table. Each line of the table is a hyperlink that shows the source document of this line in the other window.

FUTURE TRENDS

The proposal we presented here does not intend to be a universal capture system. It cannot be adapted to any application domain. It can only deal with domains that have
classifications, thesauri and databanks that refer the domain technical terms. Matching rules are based on specific terms and cannot apply to any free speech domain. Another drawback of the current system concerns the necessary time to build extraction rules. In the future, we intend to study automatic definition of DTDs according to an existing database model, so that the fastidious step of rules description could partially be assisted.
This prototype has to be improved. Currently rules can only do exact matching. We have to overcome this huge drawback in order to make the system usable in end-user daily work. For example, we have to make the analyzer misspelling-resistant.

Moreover, we are studying a more complete analysis mechanism based on transformation rules (Badr, Sayah, Laforest, & Flory, 2001). Transformation rules contain four steps to select, extract, structure information and to build queries to the database. We also will have to take into account command sentences that are at a meta-level; e.g., how to manage sentences like “replace prescription of Aspirin made yesterday by…”

We are also still working on the user interface. As presented in the previous figure, we have simplified XML tagging at the end-user view. We also want to include tools like end-of-word prediction, so that the end-user starting to write “aspi” is proposed to have directly “aspirin.” This end-of-word prediction should be context-driven; according to the paragraph type, some thesauri words have a high probability to appear.

**CONCLUSION**

This paper provides a description of the DRUID system. This system can be used to structure information in a database using weakly structured documents. We show how it is possible to provide a system associating documents capture freedom with database storage structure. A prototype allowed us to see that this system can be used with interest by end-users. In particular, we use it for medical records and doctors showed a great interest for this system.

The system we propose here provides much capture freedom to end-users, without limiting data management capabilities. Information captured in documents are transformed...
into data sent to a traditional domain-oriented database. Exact queries and statistical studies can be performed in the same way as if data were directly captured in form-based windows. As opposed to systems found in the literature, capture techniques do not impact data management power. Our proposal can be seen as a package including a user interface and an analyzer that should not replace the capture interface of the existing application. We see it as an alternative capture module, without any impact on host system data management policy.

We have developed a prototype including all specifications we presented in this chapter. Most of all, we have to improve the analyzer capabilities and the querying module of the user interface. This robust prototype will then allow experiments in daily work dimension.

REFERENCES


Chapter XI

Cooperative Query Processing via Knowledge Abstraction and Query Relaxation

Soon-Young Huh, Kae-Hyun Moon and Jin-Kyun Ahn
Korea Advanced Institute of Science and Technology, Korea

As database users adopt a query language to obtain information from a database, a more intelligent query answering system is increasingly needed that cooperates with the users to provide informative responses by understanding the intent behind a query. The effectiveness of decision support would improve significantly if the query answering system returned approximate answers rather than a null information response when there is no matching data available. Even when exact answers are found, neighboring information is still useful to users if the query is intended to explore some hypothetical information or abstract general fact. This chapter proposes an abstraction hierarchy as a framework to practically derive such approximate answers from ordinary everyday databases. It provides a knowledge abstraction database to facilitate the approximate query answering. The knowledge abstraction database specifically adopts an abstraction approach to extract semantic data relationships from the underlying database, and uses a multi-level hierarchy for coupling multiple levels of abstraction knowledge and data values. In cooperation with the underlying database, the knowledge abstraction database allows the relaxation of query conditions so that the original query scope can be broadened and thus information approximate to exact answers can be obtained. Conceptually abstract queries can also be posed to provide a less rigid query interface. A prototype system has been implemented at KAIST and is being tested with a personnel database system to demonstrate the usefulness and practicality of the knowledge abstraction database in ordinary database systems.

Copyright © 2002, Idea Group Publishing.
INTRODUCTION

Query language is used as a handy tool to obtain information from a database. However, current query processing has not been satisfactory in supporting effective decision analysis. This is largely due to exactness in nature of the relational databases and the query languages. When there is no matching data available, database users usually get a null information response. In terms of rigidness of query structure, even expert users have been frequently frustrated by the precise query specification syntax that requires them to be well-versed with the database schema for formulating correct queries. For many queries, it may be better for the database system to produce approximate answers when no exact answer is available. Usability of the database queries enhances significantly if the users are allowed to write imprecise queries, and the system understands the intent behind the queries.

Cooperative query answering aims at developing such intelligent systems that can accept less-precisely specified queries, analyze the intentions of such queries, suggest related questions or relax query conditions, and provide approximate neighborhood answers (Chu & Chen, 1994; Chu, Chen, & Lee, 1991; Cuppens & Demolombe, 1989; Minock & Chu, 1996; Motro, 1990; Prade & Testemale, 1984; Vrbsky & Liu, 1993).

One of the best applications of the cooperative query processing approach is probably the responsive online sales support in Web sites. Without intelligent help in conducting searches for a specific item on a Web site, customers are likely to be dissatisfied with their query’s answer set. Unless users are very familiar with the contents of the database, they will obtain either no answer to the query or an excess of answers that might not be sorted in any usable way. This system entirely fails to provide sales support (Poo, Toh, Khoo, & Hong, 1999).

This chapter proposes a practical approach that can be easily developed for getting approximate answers from ordinary corporate databases. As a framework, it adopts an extended conceptual clustering framework, called knowledge abstraction hierarchy (KAH), capturing the value and domain abstraction and providing non-technical explanations for the approximate query answering mechanisms. Based on the KAH, a knowledge abstraction database is designed to support approximate query answering mechanisms. A prototype system has been implemented at KAIST to demonstrate the usefulness of KAH in ordinary database application systems.

This chapter is organized as follows. The second section provides backgrounds and literature review. The next section discusses the knowledge abstraction hierarchy (KAH) in terms of two abstraction perspectives: value abstraction and domain abstraction. Next, we construct a knowledge abstraction database that incorporates the KAH on a relational data model. Then, we describes the approximate query answering operations on the basis of the knowledge abstraction database. Section 6 provides the conclusion of the chapter.

BACKGROUND

A variety of approaches have been attempted along with semantic distance, fuzzy set, rule, and conceptual classification. The semantic distance approach (Ichikawa, 1986; Jain & Dubes, 1988; Motro, 1988) represents the degree of similarity between a pair of data objects by a numerical distance. This approach has advantages of ease and efficiency in developing query relaxation algorithms, since quantitative distances among data objects are easy to compute. Multiple distance metrics are also available for each attribute domain, and users are allowed to choose the direction of relaxation by mixing a selected set of distance metrics to compute a simplistic similarity value. However, this approach is limited due to the difficulty
in transforming quantitative and qualitative data into a uniform quantitative measure. In addition, there is a lack of objective criteria for assessing semantic similarities among various data objects.

The fuzzy database approach (Buckles & Petry, 1982; Bordogna & Pasi, 1995; Bosc & Pivert 1995; Ram, 1995; Turban, 1988; Zemankova & Kandel, 1985) assumes that data objects in each domain can be assigned similarity values between 0 and 1. In addition, the knowledge base can store various kinds of imprecise information such as mixed hair color (e.g., dark-brown: 0.6 brown and 0.4 black) and a certain person’s residence (e.g., Boston or New York). Relations are extended to accept fuzzy values or to allow values that are sets of domain objects, and fuzzy comparators such as much-greater-than are provided to allow users to compose approximate queries. The answer to a query on a fuzzy database is also a fuzzy set, and thus a value in the answer has a membership degree between 0 and 1. On the basis of the rich semantic capabilities of the fuzzy set theory, the fuzzy database approach remedies the limitations of the relational database approaches by supporting various kinds of fuzziness derived from data themselves and linguistic queries. However, substantial semantics such as linguistic qualifiers for vague query (e.g., dark-brown), and impreciseness or similarity information on data should be previously incorporated in order to provide sufficient facilities for expressing various flexible data requests. Also, fuzzy databases systems demand the development of their own query languages that fit in their fuzzy database model and application domain to accommodate fuzzy-based query conditions (e.g., much-greater-than), as well as crisp conditions.

The rule-based approach (Cholvy & Demolombe, 1986; Cuppens & Demolombe, 1989; Hemerly, Casanova, & Furtado, 1994) adopts first-order logic as its formal framework and delineates semantic information about data objects and data integrity constraints using first-order formulas over a set of (base, built-in, derived) predicates. In this approach, the entire database is understood as a set of base predicates, and a database query also consists of a predicate rule whereby searching information is specified with free variables. The query is answered through conflict resolution and inference mechanisms, and query relaxation is carried out by coordinating the integrity constraints. The weakness of this approach includes a lack of systematic organization for guiding the query relaxation process and the less intuitive query answering process.

The conceptual clustering approach (Cai, Cercone, & Han, 1993; Chen, Chu, & Lee, 1990; Chu & Chen, 1994; Chu et al., 1991; Shum & Muntz, 1988; Vrbisky & Liu, 1993), to which this paper belongs, emphasizes the abstract representation of object instances and replaces the instance attribute values by the abstract concepts to which they belong. For example in the PET attribute, dogs and cats are mammals, snakes and turtles are reptiles, and so on. The abstract representations form a clustering hierarchy where similar attribute values are clustered by the abstract concepts and the individual abstract concepts are related to one another by a certain abstraction function or the abstraction similarities. It aims at accepting a user-defined casual query that may be incompatible to the database schema, and transforming the query interactively into a schema-compatible form using the clustering hierarchy. The approach helps the user to compose the query of interest with user’s familiar expressions so that both user-friendly interface and vague queries are supported. Vague queries are accommodated and transformed by mapping appropriate query conditions or by relaxing the query conditions to be more generalized with a broader search scope. The conceptual clustering approach is specifically advantageous when the characteristics of data objects are qualitative and categorical (Chu & Chen, 1994; Chu et al., 1991). However, existing conceptual clustering approaches have limitations in the diversity of vague queries and the
flexibility of clustering hierarchy maintenance. This is largely due to ambiguous distinction between value abstraction and domain abstraction information. Most studies focus on the construction of the clustering hierarchy itself from data object instances in the underlying database, but need further elaboration on the semantics of the hierarchy. If the value and domain abstraction information could be managed with explicit distinctions and support mechanisms, the clustering hierarchy can be constructed in a more systematic way by aligning a set of abstract values into a certain domain while maintaining value and domain abstraction information individually.

The extended conceptual clustering framework, proposed in this chapter, KAH, captures not only the value abstraction hierarchy but also the domain abstraction hierarchy. Specifically, the value abstraction hierarchy consists of abstract values derived from specific data values in the underlying database and constitutes a hierarchical structure based on generalization and specialization relationships. The domain abstraction hierarchy is built on the data value domains and classifies them into super-domains and sub-domains.

CONSTRUCTS OF KNOWLEDGE ABSTRACTION HIERARCHY

Knowledge abstraction hierarchy (KAH) is a knowledge representation framework that facilitates multilevel representation of data and meta-data for an underlying corporate database using data abstraction. Specifically, the KAH is constructed on generalization and specialization relationships between abstract values and specialized values.

To illustrate the KAH, Figure 1 shows two KAH instances derived from the same underlying database. The top one is Career Development Education hierarchy classifying data on courses into multiple abstraction levels including course name, course area, and course group; the bottom is College Major hierarchy on employee majors. Values constituting the hierarchies may be parts of the underlying database or artificial values added to describe the semantic relationship among the existing data values.

The KAH is composed of two types of abstraction hierarchies: value abstraction hierarchy and domain abstraction hierarchy. First, in the value abstraction hierarchy, a specific value is generalized into an abstract value and the abstract value can be generalized further into a more abstract value. Conversely, a specific value is considered as a specialized value of the abstract value. Thus, a value abstraction hierarchy is constructed on the basis of generalization/specialization relationships between abstract values and specific values in various abstraction levels. The value abstraction relationship can be interpreted as IS-A relationship. For instance, Finance is a (major name of) Management while Management is a (major area of) Business. As such, higher levels provide a more generalized data representation than lower levels.

While the cardinal relationship between an abstract value and its specific values is assumed to be one-to-many, a specific value can also have multiple abstract values that are located in different abstraction levels along a path from the specific value to its most abstract value at the highest abstraction level. In such capacity, an abstract value is called n-level abstract value of the specific value according to the abstraction level difference n.

Second, the domain abstraction hierarchy consists of domains that encompass all individual values in the value abstraction hierarchy and there exist INSTANCE-OF relationships between the domains and values. Much as generalization/specialization relationships exist between the data values in two different abstraction levels of the value
abstraction hierarchy, a super-domain/sub-domain relationship exists between two different domains, which is obtained by domain abstraction. For instance, MAJOR_AREA is the super-domain of MAJOR_NAME. All the abstract values of instance values in a sub-domain correspond to the instance values of the super-domain. Thus, the super-domain MAJOR_AREA is more generalized than sub-domain MAJOR_NAME, since MAJOR_AREA contains more generalized values than MAJOR_NAME. The cardinal relationship between two adjacent domains is assumed to be one-to-one, and a super-domain is called n-level super-domain of the sub-domain according to the abstraction level difference n.

Multiple KAH instances, as shown in Figure 1, can exist for a single underlying database when multiple perspectives are required for the database. Though the same value can be located in multiple hierarchies and thus, in multiple domains, a domain should be unique and be located only in one hierarchy. The Economics value in Figure 1, for instance, is found in both College Major and Career Development Education hierarchies, and belongs to different domains depending on the hierarchy. It belongs to the MAJOR_AREA domain in the College Major hierarchy, while the COURSE_AREA domain in the Career Development Education hierarchy. In this sense, values with an identical name can exist in multiple hierarchies, and thus its abstract values and specific values are not uniquely determined by the value itself. Domain information is additionally needed to identify the abstract and specific values for a given specific value. We call this property the domain-dependency of abstract values and specific values.

**A KNOWLEDGE ABSTRACTION DATABASE**

**Constructor Relations of Knowledge Abstraction Database**

The knowledge abstraction database is a cooperative knowledge database that incorporates data semantics involved in the KAH and facilitates approximate query answering processes. A sample knowledge abstraction database accommodating the KAH instances in Figure 1 is provided in Figure 2, which comprises three relations: DOMAIN_ABSTRACTION, VALUE_ABSTRACTION, ATTRIBUTE_MAPPING.
The DOMAIN_ABSTRACTION maintains the semantics of the domain abstraction hierarchy. Since a domain is unique in the KAH instances and has one-to-one mapping correspondence with its super-domain, “Domain” attribute becomes the key attribute. The VALUE_ABSTRACTION maintains the semantics of the value abstraction hierarchy. Since the same value name can exist in multiple hierarchies and is differentiated by its domain, both the “Value” and “Domain” attributes become the composite key of the relation, which is related with domain-dependency. In terms of abstraction relationships, each tuple of the relations represents only the 1-level abstraction relationship. On the basis of such a 1-level abstraction relationship, an abstract value in any arbitrary level can be transitively retrieved. The ATTRIBUTE_MAPPING maintains the domain information of the underlying database and helps to analyze of the query intent in query processing. Since one attribute name in the underlying database can be used in multiple relations, the “Relation” and “Attribute” become the composite key in the ATTRIBUTE_MAPPING relation. Specifically, the ATTRIBUTE_MAPPING relation integrates the DOMAIN_ABSTRACTION and VALUE_ABSTRACTION relations with relations and attributes of the underlying databases, and constructs a knowledge abstraction database on top of the existing databases. Such a context-based knowledge abstraction database can be used to support approximate query answering as well as dynamically accommodate changes in the underlying databases.

In the following, the generalization and specialization processes for a value or a domain in a knowledge abstraction database are discussed as the base processes in approximate query answering.

**Generalization and Specialization Processes for Domain and Values**

An abstract value in a KAH corresponds to multiple specialized values at lower levels in the hierarchy. The generalization process provides a higher-level abstract value by moving up the KAH from a given value of a query condition. Consider the value of Finance in the College Major KAH in Figure 1. A 1-level generalization returns its abstract value, Management, which covers Accounting and Marketing, in addition to Finance. A 2-level generalization returns a more abstract value, Business, which corresponds to wider neighborhood values ranging from Management to Economics. The n-level generalization process uses both the VALUE_ABSTRACTION and DOMAIN_ABSTRACTION relations. In principle, an n-level abstract value can be obtained by repeating the 1-level generalization process by n times.

The specialization process searches arbitrary n-level specialized values for a given abstract value by moving down the KAH. Consider the Business value in the College Major KAH of Figure 1. The 1-level specialization process returns Management and Economics as its 1-level specialized values. The 2-level specialization process returns six values ranging from Finance to Econometrics, which are the specialized values of both Management and Economics. Like the generalization process, the specialization process also uses both the VALUE_ABSTRACTION and DOMAIN_ABSTRACTION relations to obtain n-level specialized values. In general, the 1-level specialization for a given abstract value starts with the identification of the domain of an abstract value. From the domain of the abstract value, the DOMAIN_ABSTRACTION relation identifies the domain of its specialized values. Once the domain of the specialized values is identified, we can obtain the specialized values for the abstract value from the VALUE_ABSTRACTION relation. As an example, consider the Economics value in the College Major KAH of Figure 1 and the knowledge abstraction
Figure 2: The constructor relations of knowledge abstraction database

### DOMAIN_ABSTRACTION

<table>
<thead>
<tr>
<th>Domain</th>
<th>Super Domain</th>
<th>Hierarchy</th>
<th>Abstraction Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAJOR_NAME</td>
<td>MAJOR_AREA</td>
<td>College Major</td>
<td>1</td>
</tr>
<tr>
<td>MAJOR_AREA</td>
<td>MAJOR_GROUP</td>
<td>College Major</td>
<td>2</td>
</tr>
<tr>
<td>MAJOR_GROUP</td>
<td></td>
<td>College Major</td>
<td>3</td>
</tr>
<tr>
<td>COURSE_NAME</td>
<td>COURSE_AREA</td>
<td>Career Development Education</td>
<td>1</td>
</tr>
<tr>
<td>COURSE_AREA</td>
<td>COURSE_GROUP</td>
<td>Career Development Education</td>
<td>2</td>
</tr>
<tr>
<td>COURSE_GROUP</td>
<td></td>
<td>Career Development Education</td>
<td>3</td>
</tr>
</tbody>
</table>

### VALUE_ABSTRACTION

<table>
<thead>
<tr>
<th>Value</th>
<th>Domain</th>
<th>Abstract_Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Finance</td>
<td>MAJOR_NAME</td>
<td>Management</td>
</tr>
<tr>
<td>Accounting</td>
<td>MAJOR_NAME</td>
<td>Management</td>
</tr>
<tr>
<td>Marketing</td>
<td>MAJOR_NAME</td>
<td>Management</td>
</tr>
<tr>
<td>Macro Economics</td>
<td>MAJOR_NAME</td>
<td>Economics</td>
</tr>
<tr>
<td>Micro Economics</td>
<td>MAJOR_NAME</td>
<td>Economics</td>
</tr>
<tr>
<td>Econometrics</td>
<td>MAJOR_NAME</td>
<td>Economics</td>
</tr>
<tr>
<td>Management</td>
<td>MAJOR_AREA</td>
<td>Business</td>
</tr>
<tr>
<td>Economics</td>
<td>MAJOR_AREA</td>
<td>Business</td>
</tr>
<tr>
<td>Business</td>
<td>MAJOR_GROUP</td>
<td></td>
</tr>
<tr>
<td>Cost Accounting</td>
<td>COURSE_NAME</td>
<td>Accounting</td>
</tr>
<tr>
<td>Financial Accounting</td>
<td>COURSE_NAME</td>
<td>Accounting</td>
</tr>
<tr>
<td>Managerial Accounting</td>
<td>COURSE_NAME</td>
<td>Accounting</td>
</tr>
<tr>
<td>International Trade</td>
<td>COURSE_NAME</td>
<td>Economics</td>
</tr>
<tr>
<td>Business Forecasting</td>
<td>COURSE_NAME</td>
<td>Practice Course</td>
</tr>
<tr>
<td>Accounting</td>
<td>COURSE_AREA</td>
<td>Practice Course</td>
</tr>
<tr>
<td>Economics</td>
<td>COURSE_AREA</td>
<td>Practice Course</td>
</tr>
<tr>
<td>Practice Course</td>
<td>COURSE_GROUP</td>
<td></td>
</tr>
</tbody>
</table>

### ATTRIBUTE_MAPPING

<table>
<thead>
<tr>
<th>Relation</th>
<th>Attribute</th>
<th>Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>EMPLOYEE MAJOR</td>
<td>MAJOR</td>
<td>MAJOR_NAME</td>
</tr>
<tr>
<td>CAREER_PATH</td>
<td>Task</td>
<td>COURSE_NAME</td>
</tr>
<tr>
<td>CAREER_PATH</td>
<td>Prerequisite Task</td>
<td>COURSE_NAME</td>
</tr>
<tr>
<td>TASK MAJOR</td>
<td>Task</td>
<td>COURSE_NAME</td>
</tr>
<tr>
<td>TASK MAJOR</td>
<td>Required Major Area</td>
<td>MAJOR_AREA</td>
</tr>
<tr>
<td>TASK_HISTORY</td>
<td>Task Performed</td>
<td>COURSE_NAME</td>
</tr>
</tbody>
</table>

To obtain the 1-level specialized values of the Economics value, we first need to know the domain of the specialized values of Economics, due to the domain dependency of the abstract and specialized values. Since Economics has MAJOR_AREA as its domain, we get MAJOR_NAME as the sub-domain of MAJOR_AREA from the DOMAIN_ABSTRACTION relation. Next, in the VALUE_ABSTRACTION relation, by selecting the VALUE_ABSTRACTION tuples having MAJOR_NAME and Economics for the Domain and Abstract_Value attributes, we obtain specialized values of Economics including Macro economics, Micro economics, and Econometrics. By carrying out such 1-level specialization process n times, we can obtain the n-level specialized values for the given abstract value.

Table 1 defines the 1-level generalization and specialization operators in terms of the Structured Query Language (SQL) statements.

In what follows, the details of approximate query answering operations are explained where the generalization and specialization operators in Table 1 are used to facilitate diverse query relaxation paths.
**Table 1: 1-level generalization and specialization operators**

<table>
<thead>
<tr>
<th>Processes</th>
<th>Operators</th>
<th>SQL Statements Implementing the Operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-level domain generalization</td>
<td>Get Super Domain(D_i) = D_{i+1} where D_i is a sub-domain of D_{i+1}</td>
<td>SELECT Abstract Domain FROM DOMAIN_ABSTRACTION WHERE Domain = D_i</td>
</tr>
<tr>
<td>1-level domain specialization</td>
<td>Get Sub Domain(D_i) = D_{i-1} where D_i is a sub-domain of D_{i-1}</td>
<td>SELECT Domain FROM DOMAIN_ABSTRACTION WHERE Super_Domain = D_i</td>
</tr>
<tr>
<td>1-level value generalization</td>
<td>Get Abstract Value(v_i, D_i) = v_{i+1} where v_i is a specific value of v_{i+1}</td>
<td>SELECT Abstract Value FROM VALUE_ABSTRACTION WHERE Value = v_i and Domain = D_i</td>
</tr>
<tr>
<td>1-level value specialization</td>
<td>Get Specific Values(v_i, D_i) = {v_{i-1}, …} where v_{i-1} is a specific value of v_i</td>
<td>SELECT V.Value FROM VALUE_ABSTRACTION V, DOMAIN_ABSTRACTION D WHERE D.Super_Domain = D_i And D.Domain = V.Domain And V.Abstract_Value = v_i</td>
</tr>
<tr>
<td>Obtaining domain information from an attribute name</td>
<td>Get Attribute Domain(R,A) = D</td>
<td>SELECT Domain FROM ATTRIBUTE WHERE Relation = R AND Attribute = A</td>
</tr>
</tbody>
</table>

**OPERATIONS FOR APPROXIMATE QUERY ANSWERING**

In approximate query answering, a query that has no exact match in the database can still return neighborhood or generalized information. This is made possible by relaxing the search conditions—broadening the answer scope—to include additional information. We present four typical query answering mechanisms on the basis of generalization and specialization processes in the knowledge abstraction hierarchy: approximate selection, conceptual selection, approximate join, and conceptual join. A prototype query answering system has been developed at KAIST, and is tested with a personnel database system, to demonstrate the usefulness and practicality of the knowledge abstraction database in ordinary database application systems. The prototype system has the three-tier architecture as shown in Figure 3. The user client tier is implemented as a simplistic windows application supporting only user interface functions while the application server tier serves as an interactive extended SQL middleware, performing interpretation of the vague queries and transforming them into the conventional queries. The database server accommodates both the knowledge abstraction database and a corporate database such as a personnel database and executes conventional queries submitted from the application server. By virtue of such a flexible architecture, overall approximate query answering mechanisms can be continuously enhanced by replacing modules such as lexical analyzer, parser, and converter with better upgraded modules without revising client part. Example screens captured in the prototype system are presented later.

For the demonstration and explanation of the query answering processes, we use a simplified personnel database that is defined as follows:

```sql
EMPLOYEE {(id, emp_name, dept, title)}
TASK_HISTORY {(id, beginning_date, ending_date, task_performed)}
```
Approximate Selection

In a query, an attribute can be approximated to get neighborhood answers. Specifically, an attribute in the selection condition can be abstracted into a range of neighboring values approximate to a specified value. In the personnel database example, suppose that a personnel manager wants to find out people majoring in Finance but the right candidate employees are unavailable or insufficient, thus other employees with related majors need to be obtained. In the KAH, searching approximate values for a specialized value is equivalent to finding the abstract value of the specialized value, since specialized values of the same abstract value constitute approximate values of one another. Thus, given an approximate selection condition $C$, the core relaxation steps are:

1. Search the exact value that satisfies the query condition $C$. If unsuccessful, then
2. Move upward along the KAH and generalize (or relax) the specified value in the query condition to obtain an abstract one (generalization process), i.e., $C \rightarrow C'$. Then,
3. Move downward along the KAH and specialize (or shrink) the abstracted value in the abstract query to transform it into a range of specialized values in an ordinary query (specialization process), i.e., $C' \rightarrow C''$.

With the steps in view, the approximate selection query for searching employees whose major is either Finance or an approximate neighborhood value is written as follows:

$$\text{COLLEGE\_MAJOR}\{(\text{id, entrance\_date, major, graduation\_date})\}$$
$$\text{CAREER\_PATH}\{(\text{task, prerequisite\_task})\}$$
$$\text{TASK\_MAJOR}\{(\text{task, required\_major\_area})\}$$

The EMPLOYEE relation provides the current job position information of an employee, while the TASK\_HISTORY relation provides the history of tasks that the employee has performed. The COLLEGE\_MAJOR relation contains the college education records, and CAREER\_PATH shows the career development path defining prerequisite relationships among job tasks. Finally, the TASK\_MAJOR relation prescribes the relationships between individual tasks and the college major area requirements for the tasks.
Original Query
Select e.emp_name, e.dept
From employee e, college_major c
Where c.major =? "Finance" and e.id = c.id

In the query, the approximate selection is specified by the relaxation operator, =?. Note that if the relaxation operator is to be meaningful, both the attribute and specified value should be in the same domain, and thus the query is valid since both domains are identically MAJOR_NAME. Now, the generalized query at the one level is made as follows by finding 1-level abstract value of Finance.

Query Generalization
Select e.emp_name, e.dept
From employee e, college_major c
Where c.major is-a Get_Abstract_Value("Finance") and e.id = c.id

An additional query language construct, is-a, indicates the generalization relationship between a specialized value and an abstract value. In the generalized query, since the domain of Finance is identified as MAJOR_NAME through query analysis, Get_Abstract_Value("Finance") in the generalized query returns the abstract value, Management, and thus the query condition is relaxed as follows:

Select e.emp_name, e.dept
From employee e, college_major c
Where c.major is-a "Management" and e.id = c.id

The is-a operator transforms the abstract value, Management, in the following manner, to evaluate if the c.major has membership in the set of specialized values of Management:

Query Specialization
Select e.emp_name, e.dept
From employee e, college_major c
Where c.major in Get_Specific_Values("Management," MAJOR_AREA) and e.id = c.id

The 1-level specialization of the abstract value, Management, returns a set of specialized values \{Finance, Accounting, Marketing\} as neighborhood values around Finance. Thus, the specialized query is finally written as follows and can be answered as an ordinary SQL query:

Select e.emp_name, e.dept
From employee e, college_major c
Where c.major in ("Finance," "Accounting," "Marketing") and e.id = c.id

The prototype test example is provided in the Figure 5-a). In the system, the user is asked to designate the values out of the set of specialized values, \{Finance, Accounting, Marketing\} to relax the condition, and the candidate employees who majored in either Finance or Accounting are presented as the result of the approximate selection query.
Conceptual Selection

If a personnel manager is not acquainted with the values such as Accounting, Finance, and Marketing due to unfamiliarity with college majors in the Business area but still wants to find out people majoring in the Business area in general, he cannot expect to formulate an accurate query. In the approximate query answering mechanism, however, users can pose conceptual queries like “Find the name and department of the employee who has majored in any field that belongs to the general Business area.” When an abstract value is encountered in the selection condition, it is interpreted as a conceptual query and the abstract value is specialized into an ordinary query. Thus, given a conceptual selection condition $C$, the core relaxation steps are:

1. Since there is no approximate operator, assume the query as an ordinary one. Thus, search the exact values satisfying the query condition $C$. If unsuccessful, then
2. Check if the conceptual value is an abstract value in the KAH under consideration.
3. Move downward along the KAH and specialize (or shrink) the specified abstract value in the abstract query to transform it into a range of specialized values in an ordinary query (specialization process), i.e., $C \rightarrow C'$.

For example, the conceptual query searching for employees with a Business background is written as follows:

**Original Query**

```sql
Select e.emp_name, e.dept
From employee e, college_major c
Where c.major = "Business" and e.id = c.id
```

In ordinary query processing, this query is rejected since there is no Business major in the database. However, the approximate query answering mechanism interprets the mismatching condition as a conceptual condition, since the value Business is found as an abstract value in the hierarchy of Figure 1. As such, the conceptual condition refers to the selection condition case where the two compared parties, including the attribute and specified value, are located in different domains of the same hierarchy. In the above query, the domain of the c.major attribute is found to be MAJOR_NAME while the domain of Business is found to be MAJOR_GROUP. If the conceptual condition is to be valid, the domain of the Business value must be a super-domain of MAJOR_NAME (i.e., domain of the c.major attribute). Once the query condition is proven to be a conceptual one, it is automatically specialized in the following way:

**First Step of Query Specialization**

```sql
Select e.emp_name, e.dept
From employee e, college_major c
Where c.major in Get_Specialized_Values("Business," MAJOR_GROUP) and e.id = c.id
```

The 1-level specialization of the abstract value, Business, returns a set of specialized values \{Management, Economics\} as neighborhood values around Finance.

```sql
Select e.emp_name, e.dept
From employee e, college_major c
Where c.major in ("Management," "Economics") and e.id = c.id
```
Since the domain of Management and Economics comprise the super-domain of the c.major attribute, further specialization is performed along the hierarchy until the domains of the two compared parties are in the same domain. The following query is the result of a second query specialization in which all values in the selection condition are in the same domain, thereby making the query an ordinary query.

**Second Step of Query Specialization**

```sql
Select e. emp_name, e.dept 
From employee e, college_major c 
```

**Approximate Join**

As an extension of the approximate selection, an attribute in the join condition can be abstracted into an approximate range of nearby values. In an approximate join condition, a target attribute is joined with a range of neighboring values of another join attribute. If viewed in terms of the KAH, the approximate join query is equivalent to joining the two attributes on the basis of their abstract values, which would bring in broader join results than the join based on the ordinary specialized values. Thus, given an approximate join condition C, the core relaxation steps are:

1. Based on the approximate operator (e.g., =?), move upward along the KAH and generalize (or relax) both the join attributes in the query condition to have corresponding abstract values (generalization process), i.e., C ® C'. Then,
2. Perform ordinary join operation on the basis of the abstracted values of both join attributes.

For example, the approximate join query searching for people having experienced a task that is close to the prerequisite tasks for Asset Management task is written as follows.

**Original Query**

```sql
Select e. emp_name, e.dept, e.title 
From employee e, task_history t, career_path c 
Where e.id = t.id and c.task = "Asset_Management" and t.task =? c.prerequisite_task
```

In the query, the relaxation operator, =?, is used between the t.task and c.prerequisite_task. The use of the relaxation operator is quite similar to the case in the approximate selection query in the sense that both compared attributes are to be in the same domain. In the above example, both domains are TASK_NAME. Subsequently, query relaxation is made by generalizing the two attributes with their abstract values as follows:

**Query Generalization**

```sql
Select e. emp_name, e.dept, e.title 
From employee e, task_history t, career_path c 
Where e.id = t.id and c.task = "Asset_Management" and 
Get_Abstract_Value(t.task) = Get_Abstract_Value(c.prerequisite_task)
```

Joining the two relations on the basis of common abstract value can be performed in several ways with the knowledge abstraction database. One intuitive approach is introducing the ABSTRACTION relation that selects only the Value and Abstract_Value attributes from
the VALUE_ABSTRACTION relation. In the above example, records having the TASK_NAME domain are extracted into the ABSTRACTION relation as follows.

Since the ABSTRACTION relation provides abstract values for individual specialized values, it can be used as an intermediary between the two joined attributes and makes an ordinary query shown below.

**Query Specialization**

Select e. emp_name, e.dept, e.title
From employee e, task_history t,career_path c, ABSTRACTION a1, ABSTRACTION a2
Where e.id = t.id and c.task = “Asset Management” and
c.prerequisite_task = a1.value and t.task = a2.value and
a1.abstract_value = a2.abstract_value

**Conceptual Join**

As an extension of the conceptual selection, a conceptual join condition can be composed where the two attributes in the join condition can have different domains and thus be in different abstraction levels. In the personnel database example, the TASK_MAJOR relation prescribes the required major area for each task. Note that the domain of the required major area is the MAJOR_AREA and is more general than that of the major attribute (i.e., MAJOR_NAME) in the COLLEGE_MAJOR relation. This can help find people whose college major belongs to the major area required for performing a certain task, e.g., Health Insurance tasks. When the domains of the two join attributes are different as such, it is interpreted as a conceptual join, and the two domains are made the same through generalization into an ordinary join query. Thus, given a conceptual join condition C, the core relaxation steps are:

1. Since there is no approximate operator, assume the query as an ordinary one. Thus, perform an ordinary join operation and search the exact value satisfying query condition C.
2. If the join operation fails and both join domains are in a super/sub-domain relationship, then generalize the sub-domain attribute so that both domains are same.
3. Perform an ordinary join operation on the basis of the abstracted values of both join attributes.

The conceptual join query, which retrieves people whose college major belongs to the major area required for performing Health Insurance tasks, can be written as follows:

---

**Figure 4: The ABSTRACTION relation**

<table>
<thead>
<tr>
<th>ABSTRACTION</th>
<th>Abstract Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll</td>
<td>Personnel Management</td>
</tr>
<tr>
<td>Health Insurance</td>
<td>Personnel Management</td>
</tr>
<tr>
<td>Account Receivable Management</td>
<td>General Accounting</td>
</tr>
<tr>
<td>Account Payable Management</td>
<td>General Accounting</td>
</tr>
<tr>
<td>Cost Accounting</td>
<td>General Accounting</td>
</tr>
<tr>
<td>Asset Management</td>
<td>General Accounting</td>
</tr>
<tr>
<td>......</td>
<td></td>
</tr>
</tbody>
</table>
Original Query
Select e. emp_name, e.dept
From employee e, task t, college_major c
Where t.task_name = “Health Insurance” and t.required_major_area = c.major and e.id = c.id

Both join attribute domains are different but since one domain, MAJOR_AREA, is the super-domain of the other, MAJOR_NAME, the query is valid as a conceptual join query. Subsequently, generalization must be performed on the lower domain attribute, c.major, as follows:

Query Generalization
Select e. emp_name, e.dept
From employee e, task t, college_major c
Where t.task_name = “Health Insurance” and t.required_major_area = Get_Abstract_Value(c.major) and e.id = c.id

As in the previous approximate join mechanism, the ABSTRACTION relation is used to mediate the two join attributes, as follows:

Query Specialization
Select e. emp_name, e.dept, e.title
From employee e, task t, college_major c, ABSTRACTION a1
Where e.id = c.id and c.task_name = “Health Insurance” and c.major = a1.value and t.required_major_area = a1.abstract_value

The conceptual join example is provided in the Figure 5b). In the example, the system recognizes the conceptual join in the query and informs the user that the major areas required for the Health Insurance task are Management and Economics. If the user chooses Management, the candidate employees who have college majors (e.g., Finance, Accounting, Marketing) in the area of Management are presented as the result of the conceptual join query. As a whole, each mechanism serves different aspects of approximate query answering. Depending on the user’s need and knowledge about the KAH, the four types of query answering conditions can be selectively combined in a single query and enable more sophisticated queries.

CONCLUSION
The purpose of the knowledge abstraction database is to provide an intelligent interface to a relational database system by facilitating approximate query handling and producing approximate answers when exact data is unavailable. The knowledge abstraction database incorporates the knowledge abstraction hierarchy that is derived as a multilevel knowledge representation framework from the underlying corporate database. The knowledge abstraction hierarchy can be easily constructed by extracting value abstraction knowledge from the database. Since value abstraction knowledge is sensitive to the context of application, the designer of the hierarchy wants to study the database query purposes and usage patterns of the company, structure a binary relationship between abstract values and specialized
Figure 5: Approximate query answering examples

a) Approximate Selection Operation

b) Conceptual Join Operation
values using generalization relationships, and construct the entire hierarchy incrementally so that the values at the top level are most abstract while the values at the bottom are most specific. Once the value abstraction hierarchy is constructed, the domain abstraction hierarchy can be constructed in a straightforward manner by identifying the domains of all the individual abstraction levels of the value abstraction hierarchy.

The approach can be compared with other approaches, including the conceptual clustering approach, the object-oriented database approach, the rule-based approach, and fuzzy database approach. Firstly, in terms of knowledge representation, the notion of both value and domain abstraction hierarchies in the KAH is not captured by any conventional approach. The abstraction concepts in the conceptual clustering approaches (Cai et al., 1993; Chen et al., 1990; Chu & Chen, 1994; Chu et al., 1991; Shum & Muntz, 1988; Vrbsky & Liu, 1993) are closer to the value abstraction hierarchy in the KAH. They allow for multiple instance layers between a super-type and a sub-type, so that the instances of a sub-type can have different instances of the super-type. However, since the domain abstraction concept is not explicitly supported or is mixed with the value abstraction concept, various limitations are found in the approximate query answering as well as maintainability of the abstraction hierarchy. In terms of approximate query processing, existing approaches do not support join relaxation (e.g., approximate join, conceptual join), which could facilitate multiple relations to be approximately joined at once, and thus require several intermediate approximate selection queries to perform the same approximate join query. Existing approaches also have difficulties in accommodating the changes incurred in the abstraction database. In contrast, generalization and specialization relationships are applied to both the instance layers and the domain layers in the KAH. Thus, the instance of a sub-type has a specialized abstract instance of its super-type, and the domain of an abstract instance is identified as the super-domain of the instance of the sub-type. Because of the dual abstraction hierarchies, the join relaxation is supported in the KAH and such join-related approximate queries can be performed in a single query. Additionally, structural and content changes in the knowledge abstraction database can be flexibly accommodated. Details of the change maintenance in the KAH are the subject of a separate document (Huh & Moon, 1998).

Meanwhile, object-oriented database approaches support two layers: the class layer and instance layer. In the class layer, classification is used for grouping objects with common characteristics into one class. Classes are interrelated via super-class/sub-class on the basis of generalization and specialization relationship among them. These aspects of classification are more concerned with the types of involved objects as a whole rather than object instance values. Such class-oriented abstraction is almost identical to the domain abstraction hierarchy in the KAH. However, in the instance layer, there is no such concept as value abstraction whereby instances in a single class are categorized into multiple abstract instance groups. Thus, in an object-oriented database, a specialized set of instances of a sub-class cannot be directly associated with an individual abstract instance of a super-class.

Second, in terms of approximate query processing, the rule-based approaches (Cholvy & Demolombe, 1986; Cuppens & Demolombe, 1989; Hemerly et al., 1994) use inference engines on the basis of a set of logical query relaxation rules; however, the relaxation process is not intuitive enough for users to carry out interactive control. The KAH-based knowledge abstraction database can guide more interactive and flexible query transformation processes. Through the generalization and specialization processes, the value abstraction hierarchy is used as base query relaxation knowledge, and the domain abstraction hierarchy supplements more dynamic query transformation by using information about the domain of a value, the super-domain of a domain, and the abstraction level of a domain. In comparison
with a fuzzy database approach, our approach is more advantageous in that KAH does not require any additional linguistic qualifiers (e.g., dark-brown, much-greater-than) for vague query, nor impreciseness or similarity information on data for expressing query requests, and allows the use of ordinary SQL in writing approximate queries rather than demanding the development of a new query language.

At the present time, the integration of the most commonly used approaches – semantic distance, fuzzy database, and conceptual clustering – is being studied to enrich the semantics of the KAH. As a result, the integrated approach will embrace the advantages of various existing approaches and increase the intelligent characteristics of the knowledge abstraction database.

REFERENCE


Section III

Integration with the Internet Technology
Chapter XII

CMU-WEB: A Conceptual Model With Metrics For Testing and Designing Usability in Web Applications

Akhilesh Bajaj and Ramayya Krishnan
Carnegie Mellon University, USA

With the ubiquitous availability of browsers and Internet access, the last few years have seen a tremendous growth in the number of applications being developed on the World Wide Web (WWW). Models for analyzing and designing these applications are only just beginning to emerge. In this work, we propose a three-dimensional classification space for WWW applications, consisting of a degree of structure of pages dimension, a degree of support for interrelated events dimension, and a location of processing dimension. Next, we propose usability design metrics for WWW applications along the structure of pages dimension. To measure these, we propose CMU-WEB—a conceptual model that can be used to design WWW applications, such that its schema provide values for the design metrics. This work represents the first effort, to the best of our knowledge, to provide a conceptual model that measures quantifiable metrics that can be used for the design of more usable Web applications, and that can also be used to compare the usability of existing Web applications, without empirical testing.

INTRODUCTION

Over the last five years, there has been a tremendous growth of applications being developed to run over the World Wide Web (WWW) (Berners-Lee, Caillau, Luotonen, Nielsen, & Secret, 1994). Several technologies are in vogue for writing these applications
(Bhargava & Krishnan, 1998). Depending on the kind of technology used, different classes of applications can be created using the WWW as the medium of transport.

Given the large number of systems analysis and design methods available, there is some confusion as to which methods are suitable for WWW applications. This work makes two contributions. First, we present a three dimensional classification space for WWW applications. The dimensions used are the location of processing, the degree of support for interrelated events, and the structure of pages. The classification scheme we use provides insight into which existing modeling methodologies are useful for designing a WWW application along each dimension. We find that adequate models exist for the location of processing and the interrelated events dimension. However, while several modeling methods (e.g., Bichler & Nusser, 1996; Isakowitz, Stohr, & Balasubramanian, 1995) have been recently proposed for the documentation and maintenance of WWW applications, there is a need for a conceptual model that can facilitate the design of WWW applications along the degree of structure of pages dimension, such that the applications are more usable. We propose design criteria that relate to usability and hold along the structure of pages dimension in the form of high-level requirements. These requirements represent questions that should be answered by a conceptual model that seeks to facilitate the design of WWW applications so that they are more usable.

The second contribution of this work is a model that solves the above need to be able to quantitatively evaluate the high-level requirements. We propose Conceptual Model for Usable Web Applications (CMU-WEB), a conceptual model of WWW applications that facilitates the design of more useable WWW applications by providing a schema that provides values for metrics that measure the high-level requirements along the structure of pages dimension.

The rest of this chapter is organized as follows. In the following section, we present a 3-dimensional classification space for WWW applications. Next, we propose a list of high-level usability metrics, along one dimension of the space. Then, we define CMU-WEB and show how usability metric values can be derived from CMU-WEB schema. After that, we discuss how CMU–WEB can be used to design new Web applications, and how it fits in with commonly used heuristic techniques for Web application design. Finally, we give directions for future research and the conclusion in the final section.

**A CLASSIFICATION SPACE FOR WWW APPLICATIONS**

In this work, we define a WWW application as one that runs using the hypertext transfer protocol (HTTP) as the transfer protocol. In our view, this is what differentiates WWW applications from other networked applications. We define an application as consisting of a series of zero or more events. We define an event as a subset of an application that consists of at least one user input, followed by some processing.

Networked applications in general differ along several dimensions, such as the degree of state maintained on the server, the class of user, the type of user interface, and the programming language used. Before identifying dimensions for classifying WWW applications (which are a subset of all networked applications), we identify certain features that are shared by all WWW applications:

- All WWW applications are inherently client/server. The WWW client is a Web browser, that communicates with a WWW server using HTTP.1
• The HTTP protocol is inherently stateless, which means that the server does not maintain the state of the client’s application. However, several recent WWW applications involve the downloading of a client (e.g., a chat client) that then establishes a stateful link with its corresponding server (e.g., a chat server).
• The bulk of processing is usually done on the server side, although this is not necessary any more.
• The direction of data is two-way. Multimedia data flows from the WWW server(s) to the client, while alphanumeric data flows from the WWW browser to the WWW server.
• A large percentage of WWW applications are accessed by heterogeneous, naïve users. These features are common to all WWW applications. They are also what makes WWW applications different from networked applications running on some other protocol, such as CORBA (Common Object Request Brokered Architecture) or RPC (Remote Procedure Call).

Next we propose three dimensions along which WWW applications differ: the degree of structure of the WWW pages in the application, the location of processing and finally, the degree of support for interrelated events within an application. WWW applications can be classified along several different dimensions, such as the technology used in building the application, whether the application is transactional or not, or whether the groups accessing the application are naïve or expert. The three dimensions that we propose here serve the purpose of providing insight into the role of different design methodologies that are appropriate for constructing a WWW application.

Degree of Structure of the WWW Pages

This dimension looks at the degree of structure of the pages that make up a WWW application. Values along this dimension include pages following the Hyper Text Markup Language (HTML), pages following the Extensible Markup Language (XML) (Khare & Rifkin, 1997) and pages with completely flexible content, determined by the application. We now explain each of these values.

Most WWW pages today are in HTML format. An HTML page presents a hypertext interface to the user. HTML tags do not convey any meaning as to the structure of the contents of the page. WWW clients simply interpret the tags as display instructions.

The second value along this dimension is XML format. XML is an emerging standard for client browsers, and is a subset of the Standard Generalized Markup Language (SGML). XML allows the definition of the structure of the page using tags that the creator of the document can define at will. It is hoped that using tags for purposes other than merely display, as in HTML, will solve many of the problems that plague HTML pages; e.g., a group of organizations could agree on a set of tags that convey the same content. This will facilitate the development of applications that share information across these organizations by greatly reducing the amount of procedural code that would need to be written to create structure from a flat HTML format.

The third value along this dimension is complete flexibility of user interface. This is now possible by using Java applets that allow a browser to download a document that contains information that allows the execution of a Java applet (Arnold & Gosling, 1996). The applet is stored on the WWW server in bytecode format, and executed on the client machine. The applet can be used to create interfaces that are completely flexible; e.g., different chatroom applets on the WWW present different interfaces to users. Complete flexibility allows pages to be structured and presented in any way desired.

Next, we discuss the second dimension: the location of processing.
The Location of Processing of the WWW Application

This dimension looks at whether the processing (if any) takes place on the server side, or on the client and server side. Hence, we have four values for this dimension: no processing, processing only on the server, processing only on the client, and processing on the client and server. We now explain each of these values.

A large percentage of WWW pages are used for information display only. A WWW application with no processing would consist of a list of linked WWW pages. Note that processing would still be necessary for following the HTTP protocol on both client and server side. However, there is no processing of content.

Processing only on the server arises because of the Common Gateway Interface (CGI) (net.genesis & Hall, 1995). The interface allows a browser to download a WWW page and then to submit alphanumeric data to a WWW server. The WWW server receives the data and passes it to a CGI script. The data is passed using environment variables on UNIX systems, and temporary files on Windows-NT systems. The processing program, which can be in any programming language, reads this data and processes it. The results are passed back to the client, either directly by the program or via the WWW server. The result is usually another page, perhaps generated dynamically. Note that no processing takes place on the client side here. WWW applications using HTML forms that require user input use CGI.

Processing only on the client involves client-side scripting or Java applets or Active X controls. In client side scripting, the page includes programs in an interpreted language such as Javascript or Vbscript; e.g., the function is coded in the HEAD of an HTML page, and then accessed in the BODY of the page. There are a few problems with using client side scripting for large amounts of processing (Bhargava & Krishnan, 1998). First, the script is interpreted and is slower than compiled programs that usually run on the server side. Second, the source code is available to the client, which may be undesirable. Third, increase in size of client-side scripts causes slower downloads. In general, light processing like error checking input is performed using client-side scripting.

Active X controls or Java applets also allow client-side processing. The downloading and execution of an applet in the browser allows a larger amount of processing to be handled by the client than is the case with client-side scripting. Note that if Java applets are used, it is possible to bypass HTTP, and to establish a persistent state network connection using Java’s extensive networking support. Typically, Java applets and Active X controls are used to create user-interfaces. However, they can also be used to perform more processing on the client side.

Processing on both the client and server means that the processing of events in the application is divided between the client and the server. This involves the use of CGI (for processing on the server) and of client-side scripting or Java applets or Active X controls (for processing on the client).

Next, we discuss the third dimension: the degree of support for interrelated events within an application.

The Degree of Support for Interrelated Events

This dimension measures the degree to which the events within the WWW application can be interrelated to each other. We propose four values along this dimension: no events, only independent and idempotent (I&I) events, sets of I&I events interspersed with interrelated events, and sequences of interrelated events. We now explain each value.
No events occur in applications with an absence of processing of any content. This would happen in an application that simply displayed pages, and allowed for hypertext navigation between pages. This is also called a kiosk application (Troyer & Leune, 1998).

Events processed on WWW servers are I&I events because of the stateless nature of HTTP, i.e., the server cannot keep track of events in the application. E.g., if a CGI application requires the client to supply a series of forms that are written to server files, then each time the “submit” button is pressed on the client, an application event is generated on the server. Since HTTP is stateless, each “submit” event from the client is treated without knowledge of any previous submits. There is no way to keep track of the state of how many write-events have been done by a client, or whether a client decided to repeat some write-events by resending some forms of the application.

In a well-designed WWW application of this type, the events that are generated on the WWW server should be idempotent (each event in an application instance can be run multiple times without changing the outcome of the application instance). Also, server events should belong to event sets, i.e., there should be no interdependence between the different events in the application, represented by different pages. This is needed because it is impossible to keep track of the state of the application instance between events. Therefore, in an application of this type, the only solution is to clump all inter-dependent input from users in an application on one page, as one event.

Sets of I&I events interspersed with sequences of interrelated events arise in the case of processing on the client and server side, where the client is a browser, and the server is the WWW server. Note that the client can maintain state of the application. Thus, in a WWW application of this type, interrelated events are processed on the client side, and I&I events are processed on the server side. This kind of application will consist of a sequence of interrelated events (on the client), followed by a set of server (I & I) events, followed by another sequence of client events, etc. An example of this would be performing error checking on an HTML form for correct format masks, permissible value ranges, etc., by a client-side script and then sending the form to the server. The checking at the client-side leads to a sequence of interrelated events, written as a client side script. The submission of the form to the server leads to an I & I event.

Sequences of interrelated events arise in the case of processing on the client and server side, where a special client (e.g., a chat client) can be downloaded on a WWW browser, and can establish a stateful link with its corresponding (chat) server. Once a stateful link is established, the application becomes a sequence of fully interrelated events, since both the (chat) client and the (chat) server can keep track of the state of the (chat) application. WWW applications that employ state maintenance technologies like cookies can also contain sequences of interrelated events.

Our three-dimensional space for classifying WWW applications is shown in Figure 1. An application is classified by a triple that represents values along the three axes. E.g., a WWW application using a HTML pages, with Javascript and CGI sharing the processing would be (HTML, Client and Server, Sequences of interrelated events interspersed with I & I events). A WWW application that displayed HTML pages would be (HTML, no processing, no events). A WWW chat room application that involved downloading a Java applet chat client that connected to a chat server would be (complete flexibility, client and server, sequence of interrelated events).
Insights from the Classification Scheme

The classification scheme provides insight into what models should be used in the analysis and design phase of a WWW application. First, for the degree of support for interrelated events dimension, no model that depicts events is needed for the no events value. In case of the other three values on the dimension, well-known systems analysis and design techniques such as Object-Oriented Analysis and Design (OOAD) (Booch, 1994) can be used. Techniques like OOAD are very suitable for applications that are a sequence of interrelated events. They do not, however, to the best of our knowledge, allow the modeling of the idempotency of each event or a series of events. This is an area for future research. Apart from this, OOAD or other methodologies can be used to design applications along all values of the degree of support for interrelated events dimension, except for applications with no events. There is a large body of literature on using well-known systems analysis and design models with some metric-based feedback to assess the quality of a good design (e.g., Booch, 1994; Bulman, 1991; Jefferey & Stathis, 1996; Martin & Odell, 1992). Examples of well-known metrics for this dimension include lines of code, the function point metric and high-level requirements include the sufficiency, primitiveness and completeness of methods and the coupling and cohesion of classes in OOAD.

Second, for the location of processing dimension, no models that allow the analysis of sharing of processing are needed for no processing. Also, no models are needed for WWW applications where all the processing is done on one machine (only on the WWW client or only on the server). There is a substantial body of literature on designing client/server applications, where processing is distributed between the client and the server (e.g., Boar,
The design methodologies in this literature also provide some metric-based feedback on what a good design is. Many of these design methodologies involve creating discrete event simulation models of the client-server interaction (Deshpande et al., 1996) or analytical queuing models (Drakopoulos, 1995). Examples of well-known metrics along this dimension include CPU utilization of client and server, disk input/output (I/O) utilization of client and server, average wait time for client requests at server, and average run queue length at the server.

Third, for the structure of pages dimension, several models have recently been proposed on how to document HTML applications (e.g., Bichler & Nusser, 1996; Isakowitz et al., 1995; Schwabe, Rossi, & Barbosa, 1996). To the best of our knowledge, most models that have been proposed to model WWW applications actually model only (HTML, no processing, no events) WWW applications. As we point out in Bajaj and Krishnan (1998), these existing models do not provide any metric-based feedback on how good the design of the application is; instead, they focus on documenting the application and facilitating easier maintenance. Several metrics have been identified for hypertext applications (Botafogo, Rivlin, & Shneiderman, 1992) as well as for user-interfaces in general (Nielsen, 1993; Preece, 1994; Shneiderman, 1998). Next, we draw from this previous work and identify a set of high-level metrics that should be supported by a conceptual model that provides metric-based feedback on how good a WWW application is along the structure of pages dimension.

**USABILITY REQUIREMENTS ALONG THE STRUCTURE OF PAGES DIMENSION**

Several high-level requirements have been proposed for hypermedia documents (see Garzotto, Mainetti, & Paolini, 1995, for an extensive listing). We present what we view as a canonical set of four quality requirements, i.e., the set covers most of the abstractions covered in quality requirements proposed by others, and each requirement in the set is reasonably non-overlapping with the other. The high-level requirements we present in this section represent design questions that should be answered by a conceptual model that aims to facilitate the usability design of a WWW application. In a later section, we present CMUWEB, a conceptual model that answers these questions.

The first two quality requirements attempt to measure the readability of the HTML and XML documents in the application. Two factors influence readability: coherence as a positive influence (Thuring, Hannemann, & Hake, 1995) and cognitive overhead (Conklin, 1987) as a negative influence.

**Coherence**

This requirement is used to represent the ease with which readers can form mental models of a possible world from the hypertext document. Local coherence is the degree to which each specific document conveys a mental model. For example, document that uses conjunctions, paragraphs and other writing techniques with appropriate multimedia illustrations provides higher local coherence than one that simply contains free-flowing text. Global coherence deals with the “lost in hyperspace” problem and is the degree to which the reader can form a macro structure across documents. For example, an application that maintains a preservation of context across nodes (perhaps by using HTML frames) is likely to be more
globally coherent than one whose documents are disjoint fragments of context, with no cues in each document as to the pre-context or the post-context. An important determinant of global coherence is the difficulty of navigation. For example, an application that does not permit backward navigation, or that has arbitrary jumps from each page is less easy to navigate than one that supports navigation in both directions and that has a smaller number of jumps with less “cognitive distance” per jump.

**Cognitive Overhead**

The reason for this requirement comes from the limited capacity of human information processing. In hypermedia applications, cognitive overhead is determined primarily by user-interface adjustment (Thuring et al., 1995) and low consistency (Garzotto et al., 1995). For example, an application that presents too many or changing fonts, colors and layouts on each page requires more user interface adjustment than one that presents a uniform appearance between pages with fewer fonts and colors and the same layout. An application that depicts say, sound multimedia objects differently in different pages is less consistent, and would impose greater cognitive overhead on the reader.

The previous requirements related to HTML and XML values along the dimension, since they are based on the fact that the interface for these structures is hypertext.

The next two requirements relate to the actual information presented on the pages of a WWW application.

**Cohesion of Information in a Document**

This requirement represents the need that information in a single page is cohesive in the real-world it represents. For example, if the page contains information on customers alone, then it is more cohesive, and hence better, than a page that contains information on customers as well as sales.

**Coupling of Information Across Documents**

This requirement represents the need that information in a page should be independent of information in other pages in the application. For example, if the customer name and address are duplicated across pages in the application, the coupling will be more, and hence the application will be of lower quality, than if only a key field like the customer number is duplicated across documents.10

The following high-level requirement pertains only to the complete flexibility value on the dimension. It measures the usability of a non-hypertext interface.

**Usability of Non-hypertext Interfaces**

There is a great deal of work on the usability of human interfaces (e.g., Nielsen, 1993; Preece, 1994). We define the following factors as influencing the usability of user interfaces. These factors are borrowed from Nielsen (1993), and represent a synthesis of over 90 published studies in the area of user interface design. The learnability of the interface is the perceived ease of use of the application by novice users. Efficiency of use is the steady-state performance, once the learning curve flattens out. Memorability of the interface is the amount of time that elapses before users slip back on the learning curve.
The sixth-high level requirement comes from the fact that all applications run on a network, and that network delays, slow servers, etc., can lead to long download times. It is hence applicable to all values along the dimension.

**Anticipated Download Times**

This requirement represents the time it will take to download pages from the WWW server. It is determined by endogenous factors like the server capacity, the size of the objects that would appear on each page of the application, and the number of different HTTP requests that need to be sent to create each page of the application. It is also determined by exogenous factors like the network traffic at the time of download and the number of requests to the WWW server. Applications on servers with greater processing and disk input/output capacities, with smaller size objects on each page and requiring fewer HTTP requests per page are likely to have shorter download times.

The high-level requirements are summarized in Figure 2. Note that the requirements we have proposed here stem from our classification scheme, and are for WWW applications, not for models of WWW applications. These requirements represent information that should be derivable from schemas of models that are used to design WWW applications along the structure of pages dimension.

Next, we propose CMU-WEB: a conceptual model for usable Web applications that seeks to facilitate the design of WWW applications by providing metrics that evaluate the fulfillment of requirements identified in this section.

*Figure 2: Usability requirements for values along the degree of structure of pages dimension*
Components and Semantic Rules of CMU-WEB

We define a CMU-WEB schema as a graph with nodes and arcs. The node types are:

1. **Canvas View**: A canvas view (CV) is a full page in a WWW application, with at least one information chunk (defined below). Examples of canvas views are: HTML pages, XML pages, Java forms and subsets of HTML pages if they are anchored (pointed to by a link). A WWW application is structured as a network of canvas views, with hyperlinks between them.

2. **Information Chunk**: An information chunk (IC) is a clearly discernible discrete unit of information that is represented in the application. Examples of ICs include: a textual paragraph describing an idea, an attribute (say, customer name) of a real-world entity (e.g., customer), a .gif file; a .jpeg file and textual information about a real-world entity. Only text-based information can be split into smaller ICs. A photograph, a movie or a piece of sound are considered to be single ICs. The problem of splitting these multimedia files into smaller ICs is for future research.

3. **Information Chunk Depiction**: An information chunk depiction (ICD) is the method of depicting the IC in the application. The same IC may have several different depictions, e.g., a map to a location (.gif) and verbal directions to the location (text).

4. **Hyperlink Within Application**: A hyperlink to within application (HLWA) is a hyperlink from one CV to another CV in the application. HLWAs can be one-way (HLWA1) or two-way (HLWA2: the CV that is pointed to, also points back to the original CV). HLWA2s are implemented as two separate links in a WWW application.

5. **Hyperlink Outside Application**: A hyperlink outside application (HLOA) is a hyperlink to a CV of another application.

The arc types are:

1. **Relationship Between Information Chunks**: A relationship between information chunks (RIC) is any semantic relationship between two ICs that is deemed important for the potential users of the application. Examples of RICs include both_work_for_MIST as an RIC between information_on_bob and info_on_mary, and both_introduce_company as an RIC between a text paragraph describing the company and a photograph of the company location. We note that RICs differ from hyperlinks in that they are conceptual and between ICs, while hyperlinks are actual and used for navigation between CVs.

2. **IC-ICD**: This arc type simply connects the ICD with the relevant IC.

3. **Contained-in**: This arc type connects an IC, an HLWA1, an HLWA2 or an HLOA with each CV that it appears in.

**Informal Rules for a CMU-WEB Schema**

A WWW application is one that consists of at least one Canvas View, and at least one Information Chunk. A CMU-WEB schema shows the RICs between all information chunks, as well as the canvas view locations of all information chunks, HLWA1, HLWA2, HLOA. The graphic representation of components in CMU-WEB is shown in Figure 3. The possible <arc, from-node, to-node> combinations in a schema are shown in Figure 4.
Several other models use one or more of the concepts in CMU-WEB. For example, a widely accepted model of hypertext is a graph of nodes and links. The nodes there are the same as CVs here, and the links are the same as HLWA1s. Most models for developing WWW applications use the concept of a page, which corresponds to a CV here. The IC, ICD and RIC concepts in CMU-WEB are different from other models, but most models use some sort of
primitive for capturing the information on a page, e.g., RMM (Isakowitz, Stohr, & Balsubramanian, 1995) uses a slice as that portion of a real-world entity that appears on a page. The main difference between an IC and a slice is that an IC is not restricted to belonging to some entity, and there is no need in CMU-WEB to aggregate all the chunks of information into entities, before slicing them into pages. This allows the depiction of information like abstract ideas, which may not be easily aggregated into entities. It also eliminates the creation of multiple models, and reduces the time taken to arrive at a schema. CMU-WEB is similar to the simple entity relationship model (Chen, 1976) which consisted of very few components, and where the identification of entities and relationships allows for great flexibility. When creating a CMU-WEB schema, the identification of what a relevant IC is, and what the relevant RICs are, is highly application specific and dependent on the designer of the application.

Another point of structural difference between CMU-WEB and most other current models for WWW applications is the number of components. Most other models contain significantly more components than CMU-WEB, and have significantly more semantic rules for consistency. This makes them more likely to be hard to use than CMU-WEB for modeling real-world applications (Castellini, 1998). For example, RMM has eleven components and several rules for consistency. CMU-WEB has six components (IC, RIC, CV, HLWA1, HLWA2, HLOA) that require cognitive effort to model.

Figures 5, 6 and 7 show a portion of a CMU-WEB schema that we created for an existing Web application. These figures are meant to indicate what a CMU-WEB schema looks like. For convenience, the contained-in arc is assumed. Thus, all the elements in Figure 5 are contained in the CV shown in Figure 5, etc. Since there are no RICs between ICs contained in different CVs, we can do this here for convenience. In general, the contained-in arc should be shown, if RICs exist across CVs.

Figure 5: One CV from a CMU-WEB schema created for an existing WWW application
Figure 6: One CV from a CMU-WEB schema created for an existing WWW application

Figure 7: One CV from a CMU-WEB schema created for an existing WWW application
Next, we show how CMU-WEB can be used to derive values for the high-level metrics, identified in the previous section, that indicate the usability of WWW applications.

**Deriving Values for Usability Metrics From A CMU-WEB Schema**

*Coherence*

Local coherence deals with each CV. The local coherence per CV is increased if there are more RICs between the ICs on the CV. We measure this by the *local coherence due to RIC* (LCRIC) metric:

$$\text{LCRIC} = \frac{\sum \text{RIC}}{\sum \text{IC}}$$

where the summation is over all the RICs and ICs on the CV.

We can use the mean LCRIC, across all CVs in the application, as well as the standard deviation to measure the LCRIC in the application, as a whole. The range of LCRIC is 0 to infinity. A higher LCRIC indicates more local coherence. The mean LCRIC and its standard deviation (std. dev.) can be used to compare applications differing in number of CVs and HLWAs.

In addition to LCRIC, local coherence is also dependent on the well-known principle that the capacity of the human short-term memory is $7 \pm 2$ information chunks. Thus, a CV with more than 9 ICs will have less local coherence, while one with less than 5 ICs is being inefficient. We define a second metric for local coherence called the *local coherence due to short-term memory* (LCSTM).

$$\text{LCSTM} = \sum \text{IC}$$

where the summation is the number of ICs across the page.

This metric ranges from 1 to infinity, but the optimum values are between 5 and 9. We can also get the mean LCSTM across and the standard deviation, to measure the LCSTM in the application as a whole. The mean LCSTM and its standard deviation are comparable across applications of differing sizes.

Global coherence deals with the “lost in hyperspace” problem. Global coherence is higher if the number of HLWAs per CV is higher. We define a metric called *global coherence due to HLWA* (GCHLWA).

$$\text{GCHLWA} = \frac{\sum \text{HLWA1} + 2*\sum \text{HLWA2}}{\sum \text{CV}}$$

where the summation is across the application.

GCHLWA ranges from zero to infinity. The higher the value, the more the global coherence. The metric is comparable across applications of differing sizes.

A second component of global coherence is the *difficulty of navigation*. (Botafogo et al., 1992) propose several low-level metrics that deal with the structure of hypertext documents and permit easier navigation. Since they model a hypertext as a network of nodes and links, we simply note here that their metrics can be easily derived from a CMU-WEB schema, since a node $\equiv$ CV and a link $\equiv$ HLWA1.
We provide a guideline here based on the fact that a hierarchical structure is widely known to be easier to navigate than a network, as long as the number of levels is less than 4 (Vora, 1997). Given that the short-term memory capacity of humans is between 5 and 9 information chunks, this translates to a guideline that applications having less than $9 \times 9 \times 9 = 729$ CVs (other than pure menu-based CVs) should use a hierarchical tree structure. Applications with more than 729 CVs should provide a search engine (Botafogo et al., 1992) or some sort of logical ordering of CVs.

A third component of global coherence is the cognitive jump between two CVs that are hyperlinked. Thus in each direction, an HLWA has a cognitive jump associated with it. This is dependent on the number of RICs between the source and the target CV, and also on the number of ICs in the source CV that are involved in these RICs. If there are more RICs between the source and target CVs, the cognitive jump is lower. However, the jump is higher the more the number of ICs in the source CV that are involved in these RICs. Thus, an ideal case would be where there are a large number of RICs between the two CVs, but only one IC is involved in these RICs, on the source CV. We also note that the number of ICs involved at the source can at most equal the number of RICs between the 2 CVs.

To measure this, we define a metric called global coherence due to cognitive jumps (GCCJ), for each HLWA, along each direction.

$$\text{GCCJ} = \begin{cases} 2 & \text{if the number of RICs between the source and target CV is 0.} \\ \sum \sum \frac{\sum IC}{\sum RIC} & \text{where the summation is for each link.} \end{cases}$$

In the GCCJ metric, IC represents the number of ICs in the source CV that are involved in the RICs, and RIC is the number of RICs between the 2 CVs. The range of GCCJ is between 0 and 2. The lower the value, the lower the cognitive jump. We can also compute the mean GCCJ across each direction of all HLWAs in the application, as well as the standard deviation. The mean GCCJ and its standard deviation are comparable across applications of differing sizes.

Cognitive Overhead

Cognitive overhead consists of user interface adjustment and consistency of the application. User interface adjustment is best determined through empirical testing, since it is highly dependent on the kinds of colors, fonts, and layouts used.

To measure the consistency of the application, we propose a metric called the cognitive overhead due to consistency (COC).

$$\text{COC} = \sum \frac{\sum ICD}{\sum \text{Media}}$$

where the summation is across the different types of media.

Thus, if the application uses three types of media (e.g., text, sound, video) and uses two different formats for sound, and one each for video and text, then the value of COC is $4/3 = 1.33$. The range of the COC metric is 1 to infinity, and the higher the value, the more is the cognitive overhead due to consistency. The COC metric is comparable across applications of differing sizes.
Cohesion

Cohesion looks at how interconnected the information on each CV is. To measure this, we define an IC *cluster* as the ICs on a CV that are reachable from each other, using the RICs as links. Each CV has a minimum of one IC cluster, and a maximum of infinite IC clusters. We define a metric called *cohesion* (COH) to measure cohesion

\[
\text{COH} = \frac{1}{\sum \text{IC clusters}}
\]

where the summation is across at CV.

The range for COH for each CV is 0 to 1. The higher the value, the more cohesive is the CV. We can compute the mean COH across all the CVs, as well as the standard deviation. The mean COH and its standard deviation are comparable across applications of different sizes.

Coupling

This is the degree of commonality of information across CVs. We define a *repetition count* of an IC as the number of occurrences of the IC in different CVs – 1. We define a metric called *Coupling* (COU) to measure coupling.

\[
\text{COU} = \frac{\sum \text{repetition count}}{\sum \text{IC}}
\]

where the summation is across ICs.

The COU metric ranges in value from 0 to infinity. A higher value indicates a higher coupling, which is less desirable. The COU metric is comparable across applications of different sizes.

Download Time

This metric is dependent on several endogenous variables not captured in CMU-WEB, as well as several exogenous variables listed in the previous section, and is thus not measurable from a CMU-WEB schema.

Usability of Completely Flexible Interfaces

The usability metrics for completely flexible interfaces are best tested empirically, since, to the best of our knowledge, so far there is no way to create a conceptual model that can model completely flexible interfaces.

The metrics presented above can be used to evaluate and compare the usability of existing Web applications. Next, we describe how CMU-WEB can be used to design new Web applications, and how it fits in with commonly used heuristics for Web application design.

**UTILIZING CMU-WEB TO DESIGN NEW WEB APPLICATIONS**

From the previous two sections, it follows that a key factor in utilizing CMU-WEB is deciding what RICs between ICs are meaningful. One possible method of determining this is to follow the common design practice of dividing the application into processes that the user will follow. For example, if the application is used to show the static pages of an organization, processes may include: *get_introduced_to_organization,*
**Table 1: Summary information on the usability metrics derivable from CMU-WEB schema**

<table>
<thead>
<tr>
<th>Metric Name</th>
<th>Metric Range</th>
<th>Comparable Across Applications of Different Sizes?</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCRIC</td>
<td>0 to infinity</td>
<td>Mean and std. dev.</td>
</tr>
<tr>
<td>LCSTM</td>
<td>1 to infinity</td>
<td>Mean and std. dev.</td>
</tr>
<tr>
<td>GCHLWA</td>
<td>0 to infinity</td>
<td>Yes</td>
</tr>
<tr>
<td>GCCJ</td>
<td>0 to 2</td>
<td>Mean and std. dev.</td>
</tr>
<tr>
<td>COC</td>
<td>1 to infinity</td>
<td>Yes</td>
</tr>
<tr>
<td>COH</td>
<td>0 to 1</td>
<td>Mean and std. dev.</td>
</tr>
<tr>
<td>COU</td>
<td>0 to infinity</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*find_employee_in_organization* and *learn_about_organization_products*. The processes can then be used to define RICs, so that all the ICs needed for a process are involved in an RIC. Next, we describe how such an application can be designed using the metrics in the previous section.

The LCRIC metric would be increased if all the ICs that are required in a process are portrayed on the same CV. This ties into the common heuristic approach of one screen corresponding to one activity.

LCSTM would be optimized if we chunk the information on a CV in such a way that there are between 5 and 9 chunks of information. Again, this follows the heuristic of chunking information on a screen, and not using an excessive number of chunks.

The GCHLWA metric is increased if we use a frame-based design, where all the processes are listed on every page, and the user can navigate to any process. Furthermore, the CVs associated with each process should be sequentially navigable. Again, we observe that heuristic design practices have evolved so that most Web applications now follow this approach.

The GCCJ metric would be reduced (which is favorable) if we follow the same approach described above for GCHLWA, because the only information chunk that will be related on the source CV will be the process name, which will then be linked to all the ICs on the target CV.

The COC metric would be kept low if we used one format per media throughout the application. We note here that several real-world Web applications use multiple formats for media. The idea is that each user can pick the one format they like and use that throughout the application. This is not undesirable, as long as each instance of a media is available for all the formats (i.e., consistency is maintained).

The COH metric is increased if all the ICs on a CV are connected to the same process, which is again part of the same set of design heuristics described earlier in this section.

The COU metric is increased if the processes are defined so that they are largely exclusive, which means that users would not use the same ICs across processes.
The above discussion lends validity to the CMU-WEB metrics in that common design heuristics move all the metrics in their respective favorable directions. Thus, CMU-WEB is a method of quantifying the usability of an application, and comparing it to other applications.

CONCLUSION AND FUTURE OPPORTUNITIES

In this work, we proposed a three-dimensional classification of Web applications. The classification provided insight into how the different design methodologies interact and can be used to create a WWW application. Specifically, developing a WWW application involves using design methods for each of the three dimensions in the classification.

While design methodologies already exist along both the location of processing and the degree of support for interrelated events dimensions, we identified a need for a conceptual model along the structure of pages dimension that facilitates design along this dimension. To fulfill this need, we first identified design questions that should be answered along this dimension, listing them as high-level metrics. Next, we proposed CMU-WEB—a simple conceptual model that can be used in the analysis phase of a WWW application, as well as to reverse engineer an existing WWW application. We presented a list of seven low-level metrics, whose values can be derived from CMU-WEB schema. This implies that Web applications can be designed for better usability, by using CMU-WEB as the conceptual model along the structure of pages dimension. As in all analytic design methodologies, the advantage gained by using CMU-WEB for designing a proposed WWW application is reduced time for implementation, and a better (more usable) application. We can also use CMU-WEB to compare existing WWW applications for usability, without doing any empirical testing. The approach is simple: create a CMU-WEB schema for each of the WWW applications, and derive the values of the seven metrics for each application.

We have created a CMU-WEB schema for one real-life application so far. Our experience with it has been that it is reasonably simple to use, since it has very few components and semantic rules for consistency. Just as the critical issue in the ER model is the identification of entities and relationships, the critical issue in using CMU-WEB seems to be the identification of ICs and RICs.

CMU-WEB represents, to the best of our knowledge, a first effort to create a conceptual model for designing the usability of WWW applications (versus merely documenting the application). Our future research aims at using CMU-WEB for measuring the usability of more existing WWW applications, at integrating CMU-WEB with other conceptual models, and at promoting the use of CMU-WEB by the academic and practitioner communities.

ENDNOTES

1 Unless specified otherwise, clients in this paper mean “Web browsers” and servers mean “Web servers.”
2 Examples include sound and image contained in a document.
3 Examples include protocol specific data such as GET requests from the client to the server, as well as alphanumeric data that is specific to the application.
4 In this work, we reference a large number of current technologies. Rather than referencing each one, we urge the reader to reference one of several excellent books on technologies of the WWW.
5 The display of XML documents is controlled by an accompanying XSL (extensible style sheet) which allows the same content to be displayed differently.
6 Many applications work around this by using proprietary technologies like cookies.
7 This can happen even if the page was dynamically generated in the CGI application by the server, once it is available in the client cache.
8-9 As an example of using client side processing to maintain state, consider an application that requires a complicated series of inputs from the user, where each input is dependent on the previous ones. A Java applet can take the user through these interrelated activities, and obtain the required input. Once these interrelated (input) events are done, it can then contact the WWW server with the sequence of inputs to perform a server side event.
10 We ignore processing that is part of the HTTP protocol.
11 This, of course, is similar to notions of quality in relational database systems.
12 The application is assumed to be running on a WWW server with a valid address.

REFERENCES


Chapter XIII

Managing Organizational Hypermedia Documents: A Meta-Information System

Woojong Suh
POSCO Research Institute, Korea

Heeseok Lee
Korea Advanced Institute of Science and Technology, Korea

Recently, many organizations have attempted to build hypermedia systems to expand their working areas into Internet-based virtual work places. Increasingly, it becomes more important than ever to manage organizational hypermedia documents (OHDs); metadata plays a critical role for managing these documents. This chapter redefines metadata roles and proposes a metadata classification and the corresponding metadata schema for OHDs. Furthermore, a meta-information system, HyDoMiS (Hyperdocument Meta-information System) built on the basis of this schema is proposed. HyDoMiS performs three functions: metadata management, search, and reporting. The metadata management function is concerned with workflow, documents, and databases. The system is more likely to help implement and maintain hypermedia information systems effectively.

INTRODUCTION

Today, hypermedia documents are growing explosively in many organizations because of a large number of attempts to develop systems employing intranets or extranets for enhancing their business performance. Through these systems, significant progress has been made in communication and collaboration among employees in many organizations (Lai, 2001). These systems include hypermedia documents (hyperdocuments) for supporting organizational tasks. Hyperdocuments employed for such tasks are referred to as organizational hyperdocuments (OHDs). They typically play a critical role in business, in the form of, for example, invoices, checks, or orders. The maintenance of OHD is becoming a burdensome task; managing their needs is as important to economic success as is software.
Managing Organizational Hypermedia Documents: A Meta-Information System

A hypermedia document—a special type of digital document—based on the inter-linking of nodes, such as multimedia components, etc. (Nielsen, 1993); i.e., it is an application of hypertext technologies employing multimedia components (Fluckiger, 1995). In contrast to other digital documents, a hyperdocument has links to various nodes, “hyperlinks,” that are used as a path for the navigation.

Most of the previous studies on metadata for digital documents have investigated the topic from a technical perspective, such as information discovery. However, corporate digital documents are closely related to business tasks in an organization. In this context, OHDs typically have complex relationships with both information and business processes. The OHDs can impact the speed of communications and the productivity of business processes. Accordingly, OHDs should be designed to support collaboration among workers in business processes. This aspect needs to be considered in defining metadata of the OHDs for their effective management. Furthermore, such documents should be also considered from a technical aspect. The system resources used by OHDs are a considerable part of the organizational assets.

The two objectives of this chapter are (1) to propose metadata classification and metadata schema for OHDs, and (2) to implement a meta-information system on the basis of the schema. The system was designed to support the maintenance of OHDs. Our research is rooted in previous studies on the various types of multimedia documents so as to capture the more generic perspective of metadata.

METADATA AND META-INFORMATION SYSTEM

Metadata is generally known as data about data (or information about information). Metadata for digital documents has been explored from various research perspectives: mixed media (Chen, Hearst, Kupiec, Pederson, & Wilcox, 1994), multimedia representations (Kashyap & Sheth, 1996), document objects (Sutton, 1996), and networked information resources (Dempsey & Weibel, 1996). Much past research has concentrated on the use of metadata to support access to media- and application-specific documents. This metadata describes various system properties, such as video (Jain & Hampapur, 1994; Hunter & Armstrong, 1999), images (Anderson & Stonebraker, 1994; Kiyoki, Kitagawa, & Hayama, 1994), or speech and text document (Glavitsch, Schauble, & Wechsler, 1994). In contrast to these, it has been suggested that media-integrated metadata should be developed for the management of documents with heterogeneous properties. There have been attempts to do this (Mena, Illarramendi, Kashap, & Sheth, 1996; Shklar, Sheth, Kashyap, & Shah, 1995).

These studies have described metadata roles in various technical aspects from the perspective of document types or system environments. Efficiency in document access control or interoperability of heterogeneous documents has been discussed as the prime problems of these systems. A set of hyperdocument metadata, the Dublin Core (Dublin Metadata Core Element Set) (Dempsey & Weibel, 1996; Weibel, Godby, Miller, & Daniel, 1995; Weibel & Iannellla, 1997; Bearman, Miller, Rust, Trant, & Weibel, 1999; Weibel & Koch, 2000), has also focused on the information discovery; it has a limitation in managing OHDs (Murphy, 1998) like other metadata sets (Lang & Burnett, 2000; Li, Vu, Agrawal, Hara, & Takano, 1999; Karvounarakis & Kapidakis, 2000) that are developed for discovering or controlling information resources on the Internet.

Metadata of OHDs should be considered beyond the technical aspects by including an organizational aspect toward organizational memory (OM) because they are a major source
of organizational memory (Meier & Sprague, 1996; Murphy, 1998; Sprague, 1995). The concept of OM has many facets but most authors agree that OM must support decisions by using OM techniques for managing an organization’s information or knowledge of the past (Shum, 1997; Stein & Zwass, 1995; Wijnhoven, 1998). In this context, a meta-information system for OHDs can evolve toward managing OM by extending their metadata scope to capture their history in terms of business functions, communication mechanisms, or technical artifacts, beyond focusing on contents discovery. These memories may provide knowledge to support various decisions for controlling communication mechanisms in a business process, linking to the previous responsible workers, or maintaining the hypermedia applications. Metadata roles can be summarized in three levels—operation, system, and organization—as shown in Table 1.

A meta-information system can be characterized by information resources (to be controlled) or supported services; they service three types of domains: application-oriented, hybrid, or management-oriented.

Application-oriented meta-information systems use metadata to support the application functions. Therefore, metadata schemas are primarily determined on the basis of system requirements. Examples include Web search engines such as AltaVista, Excite, and Lycos that provide services based on the metadata collected from the resources on the Internet. The metadata of such systems is usually determined by focusing on the service functions for effective information discovery. The main users of this system may be application end-users.

In contrast, management-oriented meta-information systems play a major role in supporting the reuse and maintenance of managerial resources. Such systems should be developed by a metadata-driven approach from the managerial perspective on organizational resources; the metadata should be determined first and then its system functions are designed for handling the metadata. In a document-oriented environment, these systems should serve managerial capabilities for the system- and business-related information or knowledge, through the management of the metadata on organizational documents; major users may be system analysts, information managers, or system administrators.

Hybrid domain systems pay attention to metadata for the managerial purposes, as well as specific application functions. Accordingly, the major users may not only include application managers but also end-users or application specialists. Examples of this domain include EDMS (Electronic Document Management System), which requires metadata as an essential component for document handling (Sutton, 1996).

<table>
<thead>
<tr>
<th>Level</th>
<th>Metadata Roles</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operation</td>
<td>• Easy and fast access</td>
</tr>
<tr>
<td></td>
<td>• Increased accuracy</td>
</tr>
<tr>
<td>System</td>
<td>• Interoperability under heterogeneous environment</td>
</tr>
<tr>
<td></td>
<td>• Document maintenance</td>
</tr>
<tr>
<td></td>
<td>• Document distribution</td>
</tr>
<tr>
<td>Organization</td>
<td>• Increased reusability of information and knowledge resources</td>
</tr>
<tr>
<td></td>
<td>• Increased capacity of business management</td>
</tr>
<tr>
<td></td>
<td>• Increased organizational memory</td>
</tr>
</tbody>
</table>

Table 1: Metadata roles
METADATA CLASSIFICATION AND ELEMENTS FOR HYPERDOCUMENTS

Metadata Classification

Metadata classification can be perceived as a fundamental framework for providing metadata elements. The roles of the elements are determined by the classification coverage. Bohm and Rakow (1994) proposed a metadata classification for multimedia documents. It focuses on the representation of media type, content, relationships among document components, history, and location. Another metadata classification is specified by the dependency on the content (Kashyap & Sheth, 1996; Mena et al., 1996; Shklar et al., 1995). On the basis of these two kinds of classification, metadata for managing medical documentation using hypermedia (Consorti, Merialdo, & Sindoni, 1996) and quality of service in distributed multimedia systems (Kerherv, Pons, Bochmann, & Hafid, 1996) has also been developed.

This study proposes the following metadata classification for organizational hyperdocuments:

- **Content-dependent metadata**: This metadata is used to enable understanding of the content of documents. The metadata includes information that depends on (1) the content directly, and (2) semantic meanings based on the content of the document indirectly.

- **Workflow-dependent metadata**: This metadata provides information about workflow related to an organizational hyperdocument.

- **Format-dependent metadata**: This metadata describes information on formats related to organizational hyperdocuments, as well as hypermedia components, such as nodes and interface sources.

- **System-dependent metadata**: This metadata provides information concerned with storage- and software-related information on system resources, such as hyperdocuments, interface sources, and databases.

- **Log-dependent metadata**: This metadata describes information on the history and the status of organizational hyperdocuments.

Workflow-dependent metadata is concerned with process-related factors such as workers, tasks, or business rules. Generally, corporate documents are produced in undertaking an organizational process (Uijlenbroek & Sol, 1997); furthermore, most businesses are based on, or driven by, document flow (Sprague, 1995). Thus, documents and business processes may be considered simultaneously in the analysis of a corporate information system (Frank, 1997). Accordingly, workflow-dependent metadata is required for the effective management of OHDs in an information system.

Format-dependent metadata is concerned primarily with the artifacts related to a hyperdocument, such as nodes, anchors, interface sources, or database attributes. The meta-information of formats can provide an understanding of the hypermedia features in terms of structures and operational mechanisms, so that it can be useful in the technical maintenance of hyperdocuments. The system-dependent metadata can also play a critical role in technical maintenance by providing information on hardware and location, and software technologies applied to the hyperdocuments. This meta-information is essential for sharing and reusing system resources. Finally, log-dependent metadata may contribute to organizational memories. Thus, the metadata in this category should be specified in order to capture the history of OHDs.
**Metadata Elements**

Metadata elements for digital documents have typically been determined differently according to the characteristics of documents and purposes of their systems. Most of the OHDs in business applications typically perform complex functions that are often connected with a corporate database for business tasks in a workflow. This chapter focuses on OHD maintenance in consideration of processes and system artifacts. From this perspective, detailed metadata elements may be specified under the classification suggested in this chapter, as shown in Table 2.

Content-dependent classification consists of elements that enable users to understand the content of the hyperdocuments. The document domain may be in terms of content and roles. The conceptual attributes, as data items represented on a hyperdocument, are connected to a corporate database. Interface sources are primarily multimedia components, such as image or animation that are represented on interfaces.

A node, an essential factor of hypermedia, has been defined as the fundamental unit of hypertext (Nielsen, 1993), fragments of information (Fluckiger, 1995), or basic information containers (Schwabe & Rossi, 1994). This chapter defines a node as any navigational object with hyperlinks. An object may be a type of media, such as image, sound, animation, or a hyperdocument itself. Nodes may be categorized into two types from the perspective of their properties: document nodes and data nodes. Nodes are also of two types from the perspective of link directions: source and destination. The fundamental definitions for nodes are summarized in Table 3.

*Table 2: Metadata elements of organizational hyperdocuments*

<table>
<thead>
<tr>
<th>Classifications</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-dependent</td>
<td>[Document] Title, Description, Document Domain Name, Conceptual Attribute</td>
</tr>
<tr>
<td></td>
<td>Name [Anchor] Name [Data Node] Title [Interface-Source] Name</td>
</tr>
<tr>
<td>Workflow-dependent</td>
<td>Task Domain Name, Task, Agent Domain Name, Agent Object Name, Business</td>
</tr>
<tr>
<td></td>
<td>Rule</td>
</tr>
<tr>
<td></td>
<td>[Interface-Source] Property [DB] Physical Attribute Type</td>
</tr>
<tr>
<td>System-dependent</td>
<td>[Document] File Name, H/W Name, Location Path, S/W Technology [Data Node]</td>
</tr>
<tr>
<td></td>
<td>File Name, H/W Name, Location Path [Interface-Source] File Name, Storage,</td>
</tr>
<tr>
<td></td>
<td>Location Path [Database] Name, H/W Name, Location Path, Table Name, Table</td>
</tr>
<tr>
<td></td>
<td>Name, Table Type, Physical Attribute Name, DBMS Name</td>
</tr>
<tr>
<td>Log-dependent</td>
<td>Document Number, Version Number, Loading Date, Withdrawal Date, Update</td>
</tr>
<tr>
<td></td>
<td>Date, Update Description, Director, Operator</td>
</tr>
</tbody>
</table>

*Table 3: Types of nodes*

<table>
<thead>
<tr>
<th>Perspectives</th>
<th>Types</th>
<th>Descriptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Document Node</td>
<td>A unit of an HTML document, which may be a whole interface or a part of it.</td>
</tr>
<tr>
<td></td>
<td>Data Node</td>
<td>A unit of multimedia data that may be accessed from a document node.</td>
</tr>
<tr>
<td>Link Direction</td>
<td>Source Node</td>
<td>Nodes that can access a current node.</td>
</tr>
<tr>
<td></td>
<td>Destination Node</td>
<td>Nodes to which a current node can access.</td>
</tr>
</tbody>
</table>
An interface may consist of one or more hyperdocuments. Accordingly, a document node, a hyperdocument, can be either only a part of an interface or an interface itself. From these definitions, the element of node type in format-dependent metadata may take a document node or data node as its value.

The information of a hyperdocument in terms of a process can be obtained effectively by the use of a document-based workflow concept. The workflow concept typically includes common essential factors in terms of a unit of a work, a tool of a work, and a person for a work. In the document-based workflow approach, an OHD is regarded as a tool of a work. A task, as a work unit consisting of a workflow, may be described as operations or descriptions of human actions with a hyperdocument. An agent refers to a person who performs the task, and is expressed by hierarchical status in an organization. An agent domain can be defined as a group of agent objects having common tasks or organizational objectives. The agent domain can be typically conceived as a department of an organization. The task domain is a set of tasks corresponding to an agent domain. This metadata can be captured effectively by the use of a document-based workflow model proposed in WHDM (Workflow-Based Hypermedia Development Methodology) (Lee & Suh, 2001).

The format-dependent metadata is concerned with type or properties of hyperdocuments, anchors, nodes, interface sources, and databases. The types of anchors can be static or dynamic depending on their value. The definitions of these types are as follows:

- **Static anchor**: One fixed in a hyperdocument.
- **Dynamic anchor**: One generated by data stored in a database; i.e., it refers to data transformed into and represented as an anchor when the data is accessed by a hyperdocument according to any event that occurs as a function or another anchor.

The types of OHDs can be categorized into three: control, processing, and referential, according to their roles in a hypermedia application. These types are defined as follows:

- **Control Type**: Hyperdocuments that typically guide users to other hyperdocuments of processing or referential types. Homepages or index pages are examples of this type.
- **Processing Type**: Hyperdocuments that typically contain data attributes connected with a database in the style of a form.
- **Referential Type**: Hyperdocuments that provide supplementary information about work instructions, business rules, news, or products.

Properties of interface sources are multimedia properties such as images or animation. The properties of data nodes are the same as those of interface sources. The physical attribute type of a database implies the data properties of the attribute.

System-dependent metadata focuses on storage-related information. The storage-related information can be found in various applications, but they are not integrated, so it is difficult to create a synergetic effect. However, if metadata of all the components of hypermedia systems, such as hyperdocuments, data nodes, interface sources, and databases, are integrated into a repository, it is possible to manage a hypermedia system effectively. Software technology is a major factor in determining the capacity and characteristics of a system. Recently, for example, a variety of emerging software technologies, such as ASP (Active Server Page), Java scripts, Visual Basic scripts, or Perl, have had a considerable impact on the improvement of hypermedia systems. Accordingly, the information on software technologies applied to a hyperdocument may contribute to the maintenance of a hypermedia application.

Log-dependent metadata is used for tracing the history of hyperdocuments for the maintenance of their system. Although there may be log information captured automatically by an operating system or an application program, it is typically separated, so it is difficult
to obtain a synergetic effect in maintenance. Furthermore, it is insufficient to maintain a hypermedia system effectively. Therefore it is necessary to capture the information about changes of hyperdocuments synthetically. Some hyperdocuments may be operated temporarily, depending on their purposes. Accordingly, version- or time-related information should be managed. The loading date is a date pushing a hyperdocument into its system. The withdrawal date is a date removing a hyperdocument from the system for the expiration of its periodic role or for updating. Information on responsible operators and directors for a hyperdocument may be required for responsible management, or questions by new staff members.

**HYDOMIS ARCHITECTURE**

This section introduces the prototype of a meta-information system for organizational hyperdocuments (OHD), called Hyperdocument Meta-information System (HyDoMiS), included in a management-oriented meta-information domain. HyDoMiS was constructed so as to manage OHDs effectively through their metadata. This system may affect economic success in maintaining organizational hypermedia applications based on intranet or extranet.

HyDoMiS consists of two main modules: metadata management and a supporting module. These modules have their sub-modules, as shown in Figure 1. The metadata management module is responsible for metadata handling such as creating, editing, or deleting. The supporting module serves two types of functions - searching an OHD and reporting its meta-information. These functions are based on a hyperdocument metadata database.

*Figure 1: HyDoMiS architecture*
Workflow information enables us to understand the hyperdocuments’ roles in a business process. This information is concerned with workflow domains, agents, tasks, and business rules of an OHD. The document metadata management module is composed of five sub-modules, a hypermedia system, data attributes, navigation, interface sources, and log information. System-related information focuses on the hardware, and the software technologies. Attribute information is concerned with data provided by a corporate database. This sub-module, therefore, can provide complete information, as long as database-related information is provided from the database information sub-module. The navigation information sub-module provides meta-information in terms of two kinds of nodes (destination node and source node). That is, for a certain hyperdocument, we can get not only information of source nodes that can go to the hyperdocument, but also information about destination nodes, which are differentiated into two types, document node and data node. The interface source information is useful to increase the reusability of multimedia components represented on hyperdocuments. The database information module deals with information about databases connected to hyperdocuments. The information produced in this module can be used for the control of the connection between hyperdocuments and a database in the document metadata management module.

The search module provides the numbers and titles of searched hyperdocuments as a result. The search module employs two approaches: drill-down search and keyword search. The drill-down search uses a mechanism to narrow the domain to find a hyperdocument. This search can be performed by the use of various domains in terms of workflow, database, navigation, interface source, and log information. The keyword search uses a typed keyword complying with the selected items, such as workflow name, document title, abstract, or anchor name. The result produced in a search module can be transferred to a reporting module automatically, in order to generate a report on a document search. The report can provide meta-information whose nature will depend on which reporting domains are selected.

The metadata schema is produced based on metadata elements proposed in Table 2. The schema was designed as an E-R diagram for implementing a metadata database for HyDoMiS as shown in Figure 2. The schema was centered on document entity.

This schema represents complex relationships among the components employed in hyperdocument operations, and contains managerial factors for the control of task or system. Such schema content can be captured effectively through the processes of the hypermedia development methodology, WHDM (Workflow-Based Hypermedia Development Methodology) (Lee & Suh, 1999) rather than other methodologies such as VHDM (View-Based Hypermedia Design Methodology) (Lee, Kim, Kim, & Cho, 1999) or SOHDM (Scenario-Based Object-Oriented Hypermedia Design Methodology) (Lee, Lee, & Yoo, 1999). WHDM employs a document-based workflow model to capture the requirements for OHDs to be implemented.

A CASE AND A SYSTEM IMPLEMENTATION

HyDoMiS was constructed as a Web server based on Internet Information Server (IIS) 4.0 for multi-users such as developers or system administrators. These users can access the HyDoMiS through Web browsers that belong to the client. The Visual Basic script based on ASP technology was used primarily for implementing functions for dynamic navigation, metadata controls (creating, editing, and deleting), search and reporting. The metadata DB was developed with Microsoft SQL Server 6.5.
In this section, each module of HyDoMiS is illustrated by using a real-life case for a bank in South Korea. The case is concerned with OHDs in a workflow for individual loans that require insurance policies that are issued by a professional organization for credit insurance. This workflow requires rigorous approval procedures through several steps, because it is important to investigate an applicant’s ability to repay, as well as his credit.

**Metadata Management**

This module takes responsibility not only for storing and retrieving metadata for hyperdocuments but also for providing useful meta-information based on the queries using the metadata. This module includes three sub-modules: workflow, OHDs, and databases.

**Workflow Metadata Management**

Workflow metadata management module deals with meta-information on constituent factors of workflow, such as workers, tasks, and business rules related to an OHD. Moreover, this information can be reported on the basis of the relationships generated automatically through the procedure of metadata creation. The workflow meta-information is managed at two levels: domain and specification. The domain level manages information on workflows, agents, tasks, and hyperdocuments, which can be guided to their modules from Screen A of Figure 3. Among these modules, workflow domain information in Screen B can be created in Screen C, and can be edited or deleted in Screen D. Screen B shows the private loan workflow that is the case already explained. Screen C is linked to the icon of a pencil in Screen B, and Screen D is accessed from dynamic anchors generated as workflow domain names in Screen B. Most of the other sub-modules in metadata management were implemented in this manner for reporting and controlling the meta-information.
Figure 3: Screens for workflow domain metadata management

The task domain management module generates the relationships between task domains and the other domains concerned with a workflow and agents that are already produced, as reported in Screen A of Figure 4. This report makes clear which workflow domain is processed by which agent domains related to which task domains. Screen A lets us see the workflow–individual loan–and task domains; it is controlled by the telemarketing department and loan center. This information may be used to control the roles of OHDs in a business application. In this screen, task domain names are also, like dynamic anchors, linked to their editing screen. Furthermore, the data listed in each domain column can be sorted out by clicking their titles.

The module of workflow specification manages detailed information about OHDs, in terms of agent objects, and tasks, on the basis of the domain information. In this module, the task specification management module provides full specification related to a workflow, as shown in Screen B of Figure 4. From the report of Screen B, for example, in individual loan workflow, LP-1112, “Approval Confirmation,” is used by a head (manager) for the task of approving a loan, according to the business rule checking the application information and confirming the applicant’s ability to repay within the criteria. Such business rules related to an OHD can be applied differently, depending on agent object’s tasks. The information concerning the history of the business rules applied to the agent objects may become a valuable part of organizational memory, and may contribute to improvement of the business productivity. The LP-1112 document is used by credit staffs for the issue of a form submitted to the insurance company in order to apply for an insurance policy required to guarantee the
loan application. To provide this task-specific information, the other metadata on agent objects and documents should be created in advance.

**Document Metadata Management**

Document metadata management concentrates on the information of hyperdocuments themselves in terms of their content, system-related resources and managerial history. Its module is made up of five sub-modules concerned with storage and software, attributes, navigation, interface sources, and logs. The information provided by these sub-modules is helpful in understanding content, managing system resources, and controlling the navigation relationships of hyperdocuments. Therefore, the maintenance of a hypermedia system can be supported effectively by the use of such information.

Among the sub-modules, a system specification module manages information concerned with storage and software technologies. The information on document number, title, document domain, document type, file name, hardware name, and location path is managed in the module. This information is needed essentially to provide access to a hyperdocument, and to use software technologies for the maintenance of hyperdocuments.

Second, the document attribute management module is implemented for the purpose of controlling the connection of a hyperdocument with a database. Accordingly, this module provides the information needed to manage the relationships between conceptual attributes and database information, including tables and physical attributes. The database-related information is supported by the database metadata management module (See next section).

Third, the navigation management sub-module is responsible for managing nodes and hyperlinks. The hyperlinks, as the most essential component of hypermedia along with nodes, provide navigation paths between nodes (Fluckiger, 1995; Nielsen, 1993). Accordingly, for the effective management of hyperdocuments, it is essential to manage information on hyperlinks and their node specifications. This sub-module provides detailed information on nodes from two perspectives: properties and link direction. The information about source node may be useful if a hyperdocument should be changed or removed. Screen A of Figure 5 lets us see information on anchor names and document file locations of hyperdocuments.
that can access LC-11—the homepage of the loan center. Accordingly, if the content of LC-11 is changed or the document should be removed, we can edit the anchors of the source node without missing important information. That is, with the help of source node information, system managers can perform maintenance activities effectively through the efficient control of relationships among hyperdocuments.

Screen B of Figure 5 shows the information on document nodes where LC-11 can go, while Screen C lets us know the information on data nodes that LC-11 can call for. This module provides location information of the nodes, which may help users control the navigation relationships among nodes.

Fourth, the module managing interface source is included in the document metadata management module. Interface sources are typically multimedia data represented on a hyperdocument for its rich semantics. They may have properties of some multimedia types, such as image, or animation. It is important to manage interface sources for increasing the representation effectiveness of hypermedia information. This module provides information about source file names and their locations, as well as their properties.

Fifth, the log management module supports management of a document history in terms of time, changed specification, or a person in charge of the document. The information about these considerations is managed through version control. It is generally conceived that hyperdocuments have an advantage of flexibility in accommodating rapidly changing business requirements. Furthermore, the fast development of hypermedia technology has a great impact on motivation for the change of hypermedia applications. Accordingly, more complete log management is required for more effective maintenance of a hypermedia system.

Figure 5: Screens for navigation information management
Database Metadata Management

Database metadata management module manages information on a database that supports data transaction of an organizational hyperdocument. The information of a database and its location created in this module is used for providing meta-information of relationships between OHDs and a database related to them. This module consists of two sub-modules: database information and table information.

Search and Reporting

In a hypermedia system, if some components in terms of interface sources, such as logo, database schema, or processes should be changed, it is not easy to find all the hyperdocuments related to the components in a short time. Accordingly, the search function can play a critical role in maintaining hyperdocuments.

These modules were developed for the purpose of direct maintenance of hyperdocuments by using the metadata stored in a metadata database. The search function was implemented in two ways: drill-down search and keyword search. The drill-down search derives nested attributes from the selected attributes, and the query is based on the attributes chosen from the latest nested ones. The keyword search gets an input word, and finds hyperdocuments related to the word. As a search result, hyperdocument numbers and titles are listed, then if a user selects a hyperdocument, meta-information can be reported on the document depending on the reporting domain chosen.

Drill-down search provides a user with a mechanism to explore hyperdocument lists that belong to search domains by narrowing them down. This search method can provide nested attributes resulting from the query about attributes already selected. As an example, if a workflow domain is selected, then agent domains, task domains and document domains (which are nested in the workflow domain) are queried and listed in combo boxes, as shown in Screen A of Figure 6. The other drill-down domains, which cover all the meta-information domains of an OHD, can be seen in Screen A if the screen is scrolled up.

Agent objects or tasks may be queried in the same way. The selected items are listed in the above drill-down search interface. Search results can be represented as the lists in Screen B of Figure 6. If a document is selected from the result lists in B of the above Figure, then the number of the document is passed to the report module and appears automatically in the input box in the report on Screen C. The reporting module provides the options for the searched domain. Screen D shows meta-information about LP-1113, depending on the searched domain option, “All Information.” Besides the basic information and system specification in Screen D, the other information in terms of workflow, data attributes, navigation, interface sources, and log can be seen if the screen is scrolled up.

A keyword search may be more useful in search of a hyperdocument by use of a keyword that users already know, even if the keyword is incomplete. If the keyword, for example, is “logo,” the information on interface sources that include “logo” in their names can be listed as two kinds, bank-logo and center-logo. Interface sources may be used repeatedly in many documents. Accordingly, if an interface source is changed or should be removed, the query to find documents including the source may be a useful tool for system maintenance. The keyword search is performed depending on the search options, such as workflow name, table name, document name, anchor name, or operator name. The results of a keyword search can be used in generating a report on searched documents in the same manner as the results in the drill-down search.
Implications
Metadata has been employed as a critical means for managing system resources. The need for managing compound multimedia documents using metadata has been pointed out. Hyperdocuments, as a type of compound multimedia document, may have common metadata with other types of multimedia documents. However, for more effective management of hyperdocuments, it is necessary to investigate metadata to capture its own generic features such as the links. Hyperdocument metadata proposed by past research has a limitation in managing OHDs that support organizational tasks. Accordingly, task-related metadata needs to be incorporated into their metadata schema. The HyDoMiS proposed in this chapter was developed from this perspective in order to provide meta-information for maintaining OHDs technically and supporting organizational process management.

Practical benefits of the HyDoMiS can be summarized as follows: First, it can efficiently support the changes of OHDs corresponding to the organizational tasks that need to be changed. HyDoMiS can provide information to analyze business tasks and OHDs simultaneously. Second, complex hyperlinks of OHDs can be controlled effectively. Most changes of hypermedia information systems (HISs) are more likely to require handling hyperlinks. Accordingly, the ability to control the hyperlinks is important to maintain HIS. Third, it is possible to effectively control the coordination between hyperdocuments and a database. Meta-information on cooperative relationships between a database and hyperdocuments enables us to take efficient action for the requirements for changing
business data. Fourth, system resources related to OHDs can be reused and managed effectively, which can reduce many programmers’ efforts for maintenance. For example, if it is necessary to change an interface component, all the hyperdocuments that contain the component can be found by search and reporting functions. Fifth, organizational memory can be improved from the technical and business perspectives. The history of business requirements and the responses of HIS to these requirements can be accumulated. This history can help new members understand a system and contact past responsible staff members for maintenance.

In spite of this usefulness, HyDoMiS still has a weakness related to automatic problems. For higher efficiency of HyDoMiS, it is necessary to improve the automatic capability for obtaining metadata in two ways: one is to make an automatic coordination possible with a CASE tool that supports development of database applications, and the other is to make possible automatic changes of metadata resulting from the change of OHDs.

**CONCLUSIONS**

Recently, many organizations have expanded their business by the use of Internet technologies. Organizational hyperdocuments are critical resources for such organizations. Managing the documents may impact the success of business.

In this chapter, we propose a meta-information system, HyDoMiS (Hyperdocument Meta-information System), on the basis of a metadata database for the effective management of hypermedia resources. In order to generate a metadata schema for HyDoMiS, a metadata classification for hyperdocuments is studied, and metadata elements are thus specified. The metadata schema is developed from an organizational perspective in terms of processes and technical artifacts. HyDoMiS is constructed as a Web server for a variety of users, such as system analysts, information managers, or system administrators. HyDoMiS is expected to become a repository system for organizational memory, in the long term.

Another contribution of this chapter is that it redefines the complex relationships among the components employed in hyperdocument operations, and it captures the definitions in a metadata schema for the HyDoMiS metadata database.

On the basis of the current research, we are planning to develop the mechanisms for automatic generation of metadata by the use of hyperdocument creating/editing functions of a case tool supporting WHDM. Another research challenge is to apply HyDoMiS to a futuristic knowledge repository.

**REFERENCES**


Chapter XIV

Changing the Face of War Through Telemedicine and Mobile E-Commerce

James A. Rodger
Indiana University of Pennsylvania, USA

Parag C. Pendharkar and Mehdi Khosrowpour
Penn State at Harrisburg, USA

INTRODUCTION

The general perspective of this chapter is designed to relate the rationale used by the Department of Defense (DoD) for the military to adapt the principles of e-commerce to Telemedicine to meet increasing global crises, and to find ways to more effectively manage manpower and time. A mobile telemedicine package has been developed by the Department of Defense to collect and transmit near-real-time, far-forward medical data and to assess how this Web-based capability enhances medical management of the battlespace. Telemedicine has been successful in resolving uncertain organizational and technological military deficiencies and in improving medical communications and information management. The deployable, mobile teams are the centerpieces of this telemedicine package. These teams have the capability of inserting essential networking and communications capabilities into austere theaters and establishing an immediate means for enhancing health protection, collaborative planning, situational awareness, and strategic decision-making through Web-based Internet applications.

This chapter is designed to relate the rationale used by the Department of Defense and the Test and Evaluation (T&E) Integrated Product Team, in order to determine the military utility of the Joint Medical Operations–Telemedicine Advanced Concept Technology Demonstration (JMO-T ACTD). In order to meet increasing global crises, the U.S. military must find ways to more effectively manage manpower and time. Joint Medical Operations–Telemedicine (JMO-T) was developed by the Department of Defense to collect and transmit near-real-time, far-forward medical data and to assess how this improved capability enhances...
medical management of the battlespace. JMO-T has been successful in resolving uncertain organizational and technological military deficiencies and in improving medical communications and information management. The deployable, mobile Telemedicine Teams are the centerpieces of JMO-T. These teams have the capability of inserting essential networking and communications capabilities into austere theaters and establishing an immediate means for enhancing health protection, collaborative planning, situational awareness, and strategic decision-making.

The objectives of the chapter are focused on developing a holistic model of transformation. The model synthesizes current thinking on transformation into a holistic model and also explains the integrative influence of vision on the other four model components—environment, people, methodology, and IT perspective. The model was tested by the testing and evaluating team of the JMO-T ACTD. JMO-T ACTD has developed a very successful training program and is very aware of the importance of planned change. Top military officials, such as the Commander-in-Chief (CINC) are actively involved in change and are committed to people development through learning. The model served an applied purpose by allowing us to see how well the military organization fit current theory. The model also fit a theoretical purpose by organizing a holistic, comprehensive framework. Accordingly, we have organized and synthesized the literature into five interrelated components that act as a fundamental guide for research. The model also helped us to identify a theoretical link and apply it to the internal operations of the military and its adaptation of mobile e-commerce principles to more effectively deliver telemedicine benefits to military personnel.

BACKGROUND OF TELEMEDICINE AND ISSUES

Telemedicine is an approach of providing care for patients that are geographically separated from a doctor. Telemedicine allows a doctor and a patient to interact with each other using computer networks. Telemedicine, when used in military, has the potential to treat patients in the war zone where doctors may not be readily available. The U.S. national strategy for military pre-eminence is based on technological superiority. Through new discoveries in advanced science and technology, the goal of the Department of Defense under Joint Vision 2010 (JV 2010) is to develop the ability to directly and decisively influence events ashore and at sea—anytime, anywhere—to meet current and future challenges.

To successfully meet these challenges, the DoD must continue to move forward in its effort to incorporate telemedicine into its prime mission—to keep every service member healthy and on the job, anywhere in the world, to support combat operations, as well as humanitarian, peacekeeping, and disaster relief missions.

Telemedicine supports the DoD’s goal by electronically bringing the specialist to the primary provider who directly cares for service members in austere, remote, and isolated environments (Floro, Nelson, & Garshnek, 1998). Telemedicine also creates an opportunity to provide rapid, accurate diagnosis and therapeutic recommendations (Garshnek & Burkle, 1998). The end result is that telemedicine helps to maintain the health of service personnel and their ability to quickly return to duty, minimizing logistically burdensome, inconvenient, and expensive transportation to distant specialty care (Bangert, Doktor, & Warren, 1998).
For telemedicine methods to be successful, however, their operational effectiveness, suitability, and importance to the warfighters’ mission must continuously be tested, evaluated, and proven (Oliver, Sheng, Paul, & Chih, 1999). In 1997, the U.S. Army, in partnership with the Navy and Air Force, was tasked to develop exercises to explore the integration of advanced technologies with existing systems and architectures to meet the requirements established under JV2010.

These technologies are all aligned with the Joint Vision 2010 concepts of Dominant Maneuver, Precision Engagement, Focused Logistics and Full-Dimensional Protection. The technology initiatives utilize dedicated, small mobile teams, with a sophisticated IT infrastructure, to provide telemedicine capabilities wherever they are needed in the medical battlespace (Mann, 1997). This IT Infrastructure includes novel Medical Equipment Sets (MES) with digital capture devices such as digital cameras, digital scopes, digital blood and urine laboratories, physiological monitors, advanced digital radiography, and digital ultrasound (Perednia & Allen, 1995). Other, associated items of equipment include novel software, such as the Pacific Virtual Health Care System. This package offers electronic medical record archiving capability that enables automated, standardized teleconsultation by forward medics to higher echelon physicians (Rodger & Pendharkar, 2000).

The Joint Medical Operations-Telemedicine Advanced Concept Technical Design (JMO-T ACTD) has charged itself with operating within the concept of Focused Logistics and Full-Dimensional Protection. It is, therefore, pertinent to understand just how this ACTD can accomplish its missions/objectives and meet the operational concepts of JV2010. This operationalization is embodied in the following quote: “To protect the force, the Army will rely on a technically advanced, operationally simple network of multi-component intelligence sources capable of detecting and locating forces, active and passive obstacles, in-flight aircraft, ballistic and cruise missiles and their launch sites, chemical and biological agents, electronic jamming sources and a host of still-developing threats.”

One technology that is mentioned in the document that applies to this ACTD is the use of “advanced soldier technologies.” It is necessary for this ACTD to fit within this concept and provide the warfighter with information that identifies, early on, those countermeasures that can be used to defeat medical threats (Dardelet, 1998). It is also important to recognize other actions that may be used to defeat enemy deployment of weapons of mass destruction (WMD), especially biological agent dispersal.

Focused Logistics makes only one mention of telemedicine. “For the Army, Focused Logistics will be the fusion of logistics and information technologies, flexible and agile combat service support organizations, and new doctrinal support concepts to provide rapid crisis response to deliver precisely tailored logistics packages directly to each level of military operation.” The document portrays medical support to Focused Logistics in the form of “internet triage” and “telemedicine” in order to enhance the survivability of the joint force (Zajtchuk, 1995).

**MAIN THRUST OF CHAPTER**

Achieving 21st century medical support capability demands significant advances in the military’s ability to provide force health care and medical protection and to deploy medical communications and information management in tactical operations (Institute of Medicine, 1996). The broad mission of Telemedicine in the military is to assess advanced mobile applications that can potentially meet such demands (Paul, Pearson, & McDaniel, 1999).
U.S. military has adapted a suite of software, databases, and architecture standards to provide deployable medical information management (Tanriverdi & Venkatraman, 1998). The Theater Medical Core Services (TMCS) is a database that stores data locally and is capable of sending encrypted email to several redundant database servers via store-and-forward (Rasberry, 1998). The database servers aggregate information and store it in databases for distribution. Web servers supply data to medical personnel as customized encrypted reports.

The Medical Workstation (MeWS) is a network-based workstation equipped with portable medical devices, clinical support capabilities, medical information support, and a graphical user interface. The MeWS will support multi-patient monitoring, interface with the patient’s clinical record, and provide access to a searchable database. It will also provide full Personal Information Carrier (PIC) read and write implementation. MeWS collects, stores, and forwards medical device data and images. By utilizing a Global Positioning System (GPS), a MeWS has the capability to enter the patient’s geographical location. The various software components of the MeWS help to facilitate clinical data entry, acquisition and retrieval. MeWS enables the generation of medical facility status reports, the monitoring of disease surveillance, the updating of supplies, and tracking of evacuation requirements.

The Field Medical Surveillance System (FMSS) is an expert system that systematically detects and monitors epidemiological trends and profiles patient populations. FMSS integrates patient information to the Global Infectious Disease and Epidemiology Network (GIDEON) knowledge base. Demographic and symptomatic information is used to arrive at a presumptive diagnosis or classify the patient using discriminate analysis. FMSS is also capable of providing incidence and prevalence trends for infectious diseases.

The Libretto is a Commercial-Off-The-Shelf (COTS) hand-held computer, manufactured by Toshiba. It has the capability to automate field medic PIC card software by reading a service member’s demographic information from the PIC into the software. It can also write GPS medical encounter information to the PIC and store the information as a pre-formatted message for transmission.

Tactical medical communications require updating of the existing IT infrastructure. The previously mentioned novel hardware, software, and interfaces were implemented in order to enable this change and facilitate the transmission of medical-unique information over the existing communications hardware and Command, Control, Communication, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) networks. However, telecommunications from the operational area of responsibility (AOR) to the medical sustaining base uses the existing Defense Information Systems Network (DISN).

The technologies described above have been assembled into an exportable capability that is specifically tailored to meet the Medical Information Management (IM) and Information Technology (IT) needs of the unit it is supporting. This assemblage of technologies is referred to as the Capability Package. The capability package must work in concert with the unit’s infrastructure, communications, tactical situation, and logistical constraints if the military is to realize its full potential in meeting today’s global crises.

For such technologies to be successful, however, their operational effectiveness, suitability, and importance to the Telemedicine mission must continuously be tested, evaluated, and proven. To perform this task, the military established a Test and Evaluation Integrated Product Team (T&E-IPT) to evaluate candidate mobile models and architectures. These technologies are examined in a rigorous test and evaluation (T&E) environment with extensive user participation as a means of assessing their mobile applications. The T&E-IPT has leveraged and optimized existing communications technologies to transmit medical data.
Database technologies for mobile technologies are utilized for epidemiological and trend analyses utilizing data mining of these data warehouses.

The initial concept of operations (CONOPS) was to employ a tailored Joint Task Force (JTF) to accomplish missions in controlled environment demonstrations. The first series of demonstrations tested communication methodologies, functionality, and the field utility of collecting and sending patient data from the forward edge of the battlefield. As the information and results were obtained, the CONOPS was expanded to use additional activities. These activities are as follows:

- The deployment of mobile technologies and agents, called Theater Telemedicine Teams (TTTs), to medical treatment facilities (MTFs) to establish and conduct telemedicine operations; coordinate with signal and Command, Control, Communications, Computers, and Intelligence (C4I) assets to establish and maintain tactical medical networks; receive, verify, and log Command information provided from lower echelons;
- The use of advanced mobile information management models and technologies, such as software, databases, and architecture standards, that were adapted to provide deployable medical information management for advanced mobile applications;
- Two radio frequency (RF) networking technologies that were enhanced for user interface design in a battlefield setting; and
- Modeling and simulation (M&S) capabilities provided through advanced mobile application software during training exercises.

All of these capabilities are being evaluated by the military. The goal of this approach is to first establish effective, interoperable mobile communications in the early stages of the exercises and to then implement more robust mobile database technology capabilities as the application matures. This chapter will provide the following details of this advanced mobile application:

- Types of mobile technologies that were identified and tested as potential candidates for enhancing Telemedicine capabilities;
- Objectives of each mobile agent in the field;
- Methods and applications of these mobile technologies;
- Performance results of these mobile database technologies;
- Recommendations, lessons learned, and feedback received from actual mobile users; and
- Overall findings and results of Telemedicine mobile field agents.

HOLISTIC MODEL-SOLUTIONS AND RECOMMENDATIONS

For this project, the authors applied a holistic model to the DoD’s mobile e-commerce re-engineering strategy. Strong evidence from prior case studies shows that holism offers a viable management model for successful transformation, or re-engineering (Clark, Cavanaugh, Brown, & Sambamurthy, 1997). Our model consists of five interdependent components—environment, people, methodology, information technology (IT) perspective, and vision (Paper, Rodger, & Pendharkar, 2000).

Environment

Basic environmental factors that lead to structural change include top management support, risk disposition, organizational learning, compensation, information sharing, and
resources (Amabile, 1997; Lynn, 1998; O’Toole, 1999). Innovation can come from any level of an organization, but environmental change originates at the top (Paper, 1999; Cooper & Markus, 1995). When employees actually see top managers initiating process improvement changes, they perceive that their work is noticed and that it is important to the organization (Paper & Dickinson, 1997; Paper, 1999).

It has been argued that the fear of failure must be limited and risk taking promoted for innovation to thrive (Nemeth, 1997). Many organizations make the mistake of trying to manage uncertainty with creative projects by establishing social control; however, it is the freedom to act that provokes the desire to act (Sternberg, O’Hara, & Lubart, 1997).

The ability to learn as an organization dictates whether and how fast it will improve (Harkness, Kettinger, & Segars, 1996). Knowledge exchange between and among teams appears to give some organizations a distinctive competitive advantage (Lynn, 1998). Learning as part of the environment enables top management to disseminate its change message to the people who do the work (Gupta, Nemati, & Harvey, 1999).

Compensation has been attributed as a means of motivating employees to perform better (Pfeffer, 1998). Being rewarded for one’s work sends the message to employees that their contributions to the organization are valued. It seems logical to conclude that people who are well compensated for risk taking, innovation, and creativity will continue that behavior (Paper, Rodger, & Pendharkar, 2000).

Information sharing enables people to better understand the business and what it requires to be successful (Paper, & Dickinson, 1997; Harkness et al., 1996). Restricting information, on the other hand, inhibits change.

Resources can be viewed as a source for providing a variety of services to an organization’s customers (Kangas, 1999). According to Barney (1991), an organization’s resources can include all assets, capabilities, organizational processes, attributes, information, and knowledge that enable the organization to develop and implement strategies that improve its efficiency and effectiveness.

People

Transformation success hinges on people and their knowledge, creativity, and openness to change (Cooper & Markus, 1995). Real change will not occur without mechanisms in place to help people transform processes. Such mechanisms include training and education, challenging work, teamwork, and empowerment.

“Education and training is the single most powerful tool in cultural transformation” (Wong, 1998). It raises people’s awareness and understanding of the business and customer. Training helps develop creativity, problem solving, and decision-making skills in people previously isolated from critical roles in projects that potentially impact the entire enterprise. Business education is equally important in that people need to know how the business works in order to add value to business processes (Paper, 1999; Paper & Dickinson, 1997).

When work is challenging, people are more motivated, satisfied, and often more productive (Hackman, Oldham, Janson, & Purdy, 1975). Challenge allows people to see the significance of and exercise responsibility for an entire piece of work (Cummings & Oldham, 1997). Challenge stimulates creativity in people and gives them a sense of accomplishment (Amabile, 1997).

People cannot reach their creative potential unless they are given the freedom to do so (Pfeffer, 1998). Management, therefore, needs to be sensitive to and aware of its role in creating a workplace that allows people freedom to act on their ideas.
Methodology

Methodology keeps people focused on the proper tasks and activities required at a specific step of a transformation project. It acts as a rallying point for cross-functional teams, facilitators, and managers as it informs them about where the project is and where it is going (Paper & Dickinson, 1997). It allows people to challenge existing assumptions, recognize resistance to change, and establish project buy-in (Kettinger et al., 1998). Of critical importance in the beginning stages is the buy-in and direction from top management, which is essential to identifying information technology opportunities, informing stakeholders, setting performance goals, and identifying BPR opportunities. Direction is important because large-scale re-engineering spans functional boundaries in which people from across the organization are involved (Paper, 1998).

Information Technology (IT) Perspective

The perspective of IT professionals toward change is critical because technology implementation is an organizational intervention (Markus & Benjamin, 1996). As such, IT can either strengthen or weaken an organization’s competitiveness (Kangas, 1999).

As introduced by Markus and Benjamin (1996), the three fundamental models of IT change agency are traditional, facilitator, and advocate. Each model offers the dominant belief system or perspective of IT professionals toward the goals and means of work that shape what they do and how they do it. IT professionals with the traditional perspective believe that technology causes change. IT professionals with the facilitator perspective believe that people create change. IT professionals with the advocate perspective also believe that people create change; however, they believe that the advocate and the team are responsible for change and performance improvements. The facilitator perspective best characterizes the philosophy adopted at the DoD Telemedicine project.

Consistent with the change-agency theory, IT perspective is categorized rather than measured. IT perspective cannot really be measured because one has one belief system or another. The facilitator perspective views change as influenced by the people who do the work. Managers facilitate and guide the process. However, they do not control the process in any way. People control the processes, set their own goals, and are responsible for the consequences. However, managers share goal-setting tasks with the group, champion the effort, and are jointly responsible for the consequences.

Mata, Fuerst, and Barney’s (1995) findings reinforce the facilitator model, and suggest that two factors effectively contribute to an organization’s competitive advantage: 1) Developing methods for strategy generation involving information resources management that emphasizes and enforces the learning of these skills across the entire organization, and 2) Developing shared goals within the entire organization. This facilitator attitude toward common business processes and systems has been adopted by many organizations, including General Motors (Schneberger & Krajewski, 1999).

Transformation (Change) Vision

Vision offers a means of communicating the re-engineering philosophy to the entire organization to push strategic objectives down through the process level and align the project with business goals. If the change vision is holistic, work is viewed as part of the whole system (Teng, Jeong, & Grover, 1998). The underlying goal of a holistic change vision is to align employee goals with those of the organization and vice versa (Drucker, 1989).
Change management, however, is very difficult because people tend to react negatively to it (Topchick, 1998). Hence, a top-down vision is imperative because it helps people understand the reasons for change. If people believe that change will benefit them or the organization, negativity is reduced. Top management has in its power the ability to influence how the organization perceives environment, people, IT, and methodology.

The vision can help open communication channels between IT and top management. One cannot be successful without frequent interactions between top management and IT change advocates (Markus & Benjamin, 1996). Open communication can help inform top management of political obstacles, training issues, and budget problems before they stymie the project. It can also help top management disseminate information about the business and BPR progress across the organization. The more informed people are about the business, the better they feel about what they do.

It is well known that organizations need information in order to compete (Ives & Jarvenpaa, 1993). The source for the following comments is the briefing Army Vision 2010 (Briefing is on the Web at URL www.army.mil/2010/introduction.htm). This document and the efforts underway to achieve its objectives shape the Army’s vision for the year 2010 and beyond. In the aggregate, the Army is seeking to “lighten up the heavy forces” and to “heavy up the capabilities of the light forces.” From mission receipt through deployment, operations and transition to follow-on operations, Army elements will execute their responsibilities through a deliberate set of patterns of operation. These patterns are:

- Project the Force,
- Protect the Force,
- Shape the Battlespace,
- Decisive Operations,
- Sustain the Force, and
- Gain Information Dominance.

These patterns are all aligned with the Joint Vision 2010 concepts of Dominant Maneuver, Precision Engagement, Focused Logistics and Full-Dimensional Protection, and illustrated in Figure 1 in the appendix. The technology initiatives utilize dedicated, small mobile teams with a sophisticated IT infrastructure to provide telemedicine capabilities wherever they are needed in the medical battlespace (Mann, 1997). This IT infrastructure includes novel Medical Equipment Sets (MES) with digital capture devices such as digital cameras, digital scopes, digital blood and urine laboratories, physiological monitors, advanced digital radiography, and digital ultrasound (Perednia & Allen, 1995). Other, associated items of equipment include novel software, such as the Pacific Virtual Health Care System. This package offers electronic medical record archiving capability that enables automated, standardized teleconsultation by forward medics to higher echelon physicians.

This ACTD has charged itself with operating within the concept of Focused Logistics and Full Dimensional Protection. It is, therefore, pertinent to understand just how this ACTD can accomplish its missions/objectives and meet the operational concepts of JV2010. This operationalization is embodied in the following quote: “To protect the force, the Army will rely on a technically advanced, operationally simple network of multi-component intelligence sources capable of detecting and locating forces, active and passive obstacles, in-flight aircraft, ballistic and cruise missiles and their launch sites, chemical and biological agents, electronic jamming sources and a host of still-developing threats.”

One technology that is mentioned in the document that applies to this ACTD is the use of “advanced soldier technologies.” It is necessary for this ACTD to fit within this concept
and provide the warfighter with information that identifies, early on, those countermeasures that can be used to defeat medical threats. It is also important to recognize other actions that may be used to defeat enemy deployment of weapons of mass destruction (WMD), especially biological agent dispersal. A graphical representation is depicted in Figure 2.

Focused Logistics makes only one mention of telemedicine. “For the Army, Focused Logistics will be the fusion of logistics and information technologies, flexible and agile combat service support organizations, and new doctrinal support concepts to provide rapid crisis response to deliver precisely tailored logistics packages directly to each level of military operation.” The document portrays medical support to Focused Logistics in the form of “internet triage” and “telemedicine” in order to enhance the survivability of the joint force (Zajtchuk, 1995). This ACTD will best support this concept by demonstrating the ability to:

- capture the data,
- see the data,
- use the data,
- use decision tools to plan and prioritize,
- model and simulate, and
- utilize the GSSS strategy to accomplish the above.

That strategy is to develop the hardware, software, database, and network solutions that impact the computer-based patient record, medical threat identification, and command and control of medical units. This will be accomplished through management of information and information technologies, deployed throughout the battlespace. Most logisticians consider medical under their purview. Therefore, logistics organizations will be streamlined and “right-sized” to allow the delivery of service in a balance between “just in time” and “just in case = just enough.” The operatives in the impact of Focused Logistics are “reduced footprint” and “tailoring on the fly” of units. This will provide for rapid crisis response, the tracking and shifting of assets while en route, and the delivery of tailored logistics packages and sustainment directly at the operational and tactical levels of operation. The JMO-T ACTD will tailor forces using novel Modeling and Simulation packages.

The GCSS Strategy is shown in Figure 3. The most important facet of all of the JV2010 concepts is that the enablers and technologies will empower soldiers and not replace them. The enablers listed for Focused Logistics are germane to this ACTD as well. These are:

- Integrated Maneuver & Combat Service Support Systems Command and Control;
- Total Asset Visibility;
- Modular Organization;
- Movement Tracking System; and
- Wireless Information Management Systems.

**Measurement of Issues and Findings**

A series of measurements were conducted to test mobile communications methodologies and functionality. The field utility of collecting and transmitting near-real-time, far-forward medical data was examined and assessed as to how this improved capability enhanced medical management of the battlespace. This phase was also used to expand and improve the techniques for testing and evaluating the proposed mobile technologies and software enhancements.

The mobile technologies were operated by typical users who performed their intended mission tasks at the projected levels of workload within a realistic operational environment. Included were the use of dedicated, small, mobile teams with associated items of equipment.
to provide telemedicine capabilities when and where needed in the medical battlespace. These items included novel medical equipment sets (MES) with digital data capture devices, as well as novel software that enables automated, standardized teleconsultation by forward medics and corpsmen to rearward physicians with an electronic medical record archiving capability. A suite of software, medical databases, and architecture standards were adapted to provide deployable medical information management.

In addition, two radio frequency (RF) networking technologies were also tested and fielded. These included the Lucent Wireless WaveLAN II system, a commercial wireless networking capability that was enhanced for military applications, and the Joint Internet Controller (JINC), a tailored set of software and firmware that is geared toward providing lower bandwidth data networking capabilities to existing military field radio systems.

The medical play in several of the demonstrations was robust enough to provide a rich opportunity to observe how these mobile technologies provided support to the user in an operational setting. These results were then used as a baseline for follow-up demonstrations and exercises.

Both the WaveLAN and JINC demonstrated their primary intended functions of mobile tactical networking capacity. The WaveLAN system provided superior bandwidth and full wireless local area network (LAN) capabilities, and the JINC provided tactical networking over low bandwidth military radio systems.

Among the outcomes, it was found that mobile technologies could successfully replace wired LANs with wireless LANs and that mobile database technology software development and refinement should be continued.

The exercises demonstrated the following capabilities:

- Theater Medical Core Services (TMCS) system—a mobile database application used to provide medical reports;
- Medical Workstation (MeWS)—a mobile, functionally configured, network-based workstation designed to support the clinical and information support requirements of forward echelon providers ashore and afloat;
- Toshiba Libretto end user terminal (EUT)—a lightweight, handheld computer capable of reading, storing, and transmitting the soldiers’ demographic information in the field;
- Desert Care II (DC II) Theater Clinical Encounter Application (TCEA)—a Web-based application that facilitates the user interface design, on the browser workstation, for mobile providers or medical technicians to record, view, and report patient encounter information in the field;
- Personal Information Carrier (PIC)—a small, portable storage device containing demographic and medical information pertaining to the soldier who is wearing or carrying the device;
- Theater Telemedicine Prototype Program (T2P2)—a Web-based delivery system of consultive care that gives healthcare providers from remote locations the ability to access the expertise of a regional facility for medical specialty consultations;
- Theater Telemedicine Team (TTT)—a mobile team composed of a leader with a clinical background, a visual systems operator, and an information systems operator who provide telemedicine capability to select, deployed MTFs; and
- Aeromedical Evacuation (AE) Suitcase—a mobile system that provides critical voice and data communications to the AE mission of the U.S. Air Force (USAF) Air Mobility Command (AMC).

The tasks needed to achieve the objectives of the demonstration were carried out. These included the ability to collect and forward healthcare data in DC II and TMCS
Lightweight Data Entry Tool (LDET), transmit it over existing communications [high frequency (HF) and International Maritime Satellite (INMARSAT)], extract it to a medical situational awareness system (TMCS), view those data in a Web environment on the TMCS server at Systems Center, San Diego (SSC SD), and conduct long-range clinical consultations. Although technical difficulties were experienced, the lessons learned from these exercises were evaluated, and solutions to these problems were incorporated into the next exercise. One good example of a lesson learned was the use of the wireless LAN to track patients within the MTF.

The exercises also indicated that essential data transport requirements of these mobile technologies can be met consistently, reliably, and cost effectively. Specific technologies were examined relative to each other for specific operational requirements of data throughput, transmission distance, time to setup, time to train, and actual costs to acquire, maintain and dispose. Among the significant achievements was the employment of the five-person mobile TTT, which successfully conducted clinical reachback capability.

Several parameters were not measured directly by the field exercise. These parameters can be determined through future exercises and battle laboratory testing and evaluation methods. For example, analysis still is not complete on the availability of mobile HF and very high frequency (VHF) radios, the overall reliability of the mobile laptops demonstrated, the software reliability of several of the communication modules, and the sustainability of several of the software database applications, hardware components, networks, and databases used in the exercise. As new data become available through future exercises and battle laboratory testing, a more complete picture of these advanced mobile applications of telemedicine will evolve.

Testing and evaluation of mobile Telemedicine applications have produced tangible evidence for the military utility of these technologies. Results from the field indicate that the essential data collection and dissemination requirements of these mobile technologies can be met consistently, reliably, and cost effectively.

The mobile models and architectures demonstrate the potential to enhance data collection and dissemination of information through the use of quality database software and robust, mobile communications infrastructure. Through its efforts, these mobile agents have developed a consistent pattern of progression. From an initial state of uncoordinated, service-unique solutions to the building of an overall mobile framework, this architectural solution is being developed and refined by several different technological concepts. These concepts have been and will continue to be assessed for operational and technical feasibility. The results from these operational and technical assessments will ultimately lead to the development and insertion of an emerging architecture, which will encompass these advanced mobile applications.

This first series of phases was conducted to test communications methodologies, functionality, and the field utility of collecting and transmitting near-real-time, far-forward medical data and to assess how this improved capability enhanced medical management of the battlespace. This phase was also used to expand and improve the techniques for testing and evaluating the proposed technologies and software enhancements specified in the exercises.

The technologies demonstrated were operated by typical users who performed their intended mission tasks at the projected levels of workload within a realistic operational environment. These technologies included the use of dedicated, small, mobile teams with associated items of equipment to provide telemedicine capabilities when and where needed in the medical battlespace. Associated items of equipment included novel MES with digital
data capture devices (e.g., digital cameras/scopes, physiological monitors, and advanced digital radiography), as well as novel software (e.g., Theater Telemedicine Prototype Project) that enables automated, standardized teleconsultation by forward medics and corpsmen to rearward physicians with an electronic medical record archiving capability. A suite of software, medical databases, and architecture standards were adapted to provide deployable medical IM.

In addition, two RF networking technologies were also tested and fielded during the exercises. These included: Lucent Wireless WaveLAN II system and JINC.

The WaveLAN system was developed and maintained from commercial-off-theshelf (COTS) wireless networking capabilities for the exercise. All JMO-T participation in this exercise was predicated on the work accomplished by the engineers to enhance the Lucent WaveLAN II system for military applications. In this regard, the WaveLAN represented an extension of a LAN via wireless means at data rates in excess of 2 million bits per second (Mbps).

The JINC system is a tailored set of software and firmware that is geared toward providing lower bandwidth—i.e., 2.4–64 kilobytes per second (Kbps)—data networking capabilities to existing military field radio systems. The basic concept behind JINC system development was to field a “programmable,” mobile tactical networking system capable of exchanging digital data between ships, aircraft, combat vehicles, and individual soldiers in the field. The JINC system was enhanced from an existing COTS product to allow data connectivity between any two existing military radio systems without reliance on satellite communications (SATCOM). The intent behind this configuration was to avoid having the ACTD become involved in procuring and installing new generation radio systems.

The JINC is composed of three elements operating together: the host computer, Team Care Automation System (TCAS) software, and a Micro-INC data controller device.

The TCAS software installed on the JINC computer host provided automated network connectivity for distributed facilities, remote users, and individual units all interconnected using existing military communications media. TCAS software is based on object-oriented technology to enhance data exchange at low bandwidths. Fundamentally, TCAS software operates in two basic modes. The first mode emulates any specified data package as an “object” in an object-oriented database structure. Using a common database distributed throughout the entire JINC network, the software takes the “objects” and compresses them using a proprietary compression scheme and then transmits the “object” across the RF network. At the receiving node, the object is decompressed and translated back into its original protocol stack prior to delivery; thus, hosts on either end of a JINC-supported RF network see the expected data format in the form it was transmitted. Using this object compression scheme, JINC is able to deliver near full use of available low bandwidth data links with very little administrative network overhead.

The Micro-INC (MINC) data controller provided the conversion from RS-232 serial data to a synchronous MIL-STD-1880-114 data stream. Each Micro-INC data controller can support up to two radio systems simultaneously. This data controller is normally located near the Single-Channel Ground and Airborne Radio System (SINCGARS) radio installation to reduce the length of the synchronous cable run. The controller requires no external or manual operation to function. All MINC functions are controlled by TCAS software.

**Technologies Demonstrated**

For this demonstration, a Mobile Medical Monitor (B) (M3B) computer system simulating a MeWS was connected to a SINCGARS via the TCAS. A Libretto system running
TCAS was connected to a second Libretto via the WaveLAN Personal Computer Memory Card International Association (PCMCIA) wireless networking devices. Abbreviated discharge summary documents in Microsoft Word format were prepared on the M3B based on input from the various sensors attached to the M3B. This message was transmitted as a file attachment to a TCAS freetext email from the M3B to the first Libretto via SINCGARS. The Libretto then ported the data, via preset forwarding rules, from the SINCGARS net over the WaveLAN net to the second Libretto using the socket interface.

The computer systems selected for the exercise consisted of Libretto 110CT-NT computers, which were similar to the 100CT Libretto EUTs. The principal difference was that JMO-T Librettos required the Windows NT 4.0 operating system to support the TMCS system. The Librettos used in the exercise generally used Windows 95/98. In addition to the basic computer system, each JMO-T EUT was provided with a Quatech four-port serial expander PCMCIA card, which allowed the connection of the PIC reader along with the Garmin 12XL Global Positioning System (GPS) device. The second PCMCIA slot on the Libretto was occupied by the WaveLAN II 803.11 PCMCIA wireless network card.

During this exercise, far-forward Hospital Corpsman (HM) transmitted medical information from four far-forward first responder sites to the medical command onboard the USS Coronado. Data was entered via the Libretto 110CT-NT, which was equipped with a PIC Reader and TMCS LDET software. Three stationary sites were located at Area 41 in Camp Pendleton, California, and one mobile platform, a High-Mobility, Multipurpose Wheeled Vehicle (HMMWV), traveled to Yuma, Arizona. Because no specific medical exercise took place during the ELB phase, each user was given a set of preprogrammed PICs to scan into the system. The data were then periodically transmitted.

Initially, the Joint Medical Semi-Automated Forces (JMedSAF) simulation was to be used in conjunction with the scenario played out on the ground to give the staff onboard the USS Coronado a more robust “picture” of the battlespace; however, early in the exercise, it became apparent that bandwidth was at a premium on the network. The demonstration manager, therefore, elected to “shut down” the JMedSAF feed to the USS Coronado to keep essential data feeds open to the Enhanced Combat Operations Center (ECOC). As a result, very little data generated from the simulation runs made its way to the TMCS database. Furthermore, the scenario of the “real” battlespace was disconnected from the “virtual” battlespace.

Results

JMO-T operations consisted of sending over 120 patient encounters via the TMCS LDET to the TMCS server located in the ECOC on the USS Coronado. Three nodes were operated by JMO-T personnel during ELB:

- HMMWV Mobile Node,
- Area 41 Node, and
- Yuma Node.

Two basic WaveLAN modes of operation were used. The first (and most commonly used) was the “standard” mode, which allowed the EUTs to communicate with the rest of the WaveLAN network via a WavePoint router connection that translated the packets for use by the rest of the network. Because the power output of the individual WaveLAN card was only 25 milliwatts, the JMO-T EUT had to be located within 1,000 feet of a WavePoint in order
to access the network. In practice, this range was extended to as much as 2,000 feet at Area 41, but this was due primarily to a high antenna mast (about 40 feet) for the Area 41 WavePoint antenna.

The other method of operation was called the “ad hoc demo” mode, which was accessed by selecting an option on the WaveLAN card “properties” window. When activated, this allowed the EUTs to communicate with each other (i.e., for training) without the need for a WavePoint.

**OPERATIONAL OBSERVATIONS BY SYSTEM NODE**

**HMMWV Mobile Node**

The intent behind JMO-T operations from the HMMWV node was to demonstrate the ability to send medical data from a highly mobile platform. In practice, this actually involved JMO-T personnel going to the location where the HMMWV was parked and joining the net from that location. One corpsman from 1/5 Marines accompanied JMO-T engineering personnel to the HMMWV location at Green Beach. The corpsman transmitted nine patient records with transmission times as follows in Table 1.

Although additional periods of JMO-T participation were scheduled with the HMMWV node, revisions to the exercise schedule resulted in postponement and eventual cancellation of other HMMWV JMO-T operations, because the HMMWV was needed elsewhere in the exercise.

**Area 41 Node**

The majority of JMO-T operations occurred at the main exercise Area 41 operations center and relay node. Results of patient record transmissions are provided in Tables 2 and 3.

After the completion of the videoconference, JMO-T personnel and 1/5 Marines corpsmen experienced dramatically improved network performance. LDET transmission times for all patient encounters were in the 3-5-second range. In addition, the ECOC TMCS server was able to be viewed and browsed from the Libretto EUTs (something that had not

<table>
<thead>
<tr>
<th>Patient Number</th>
<th>Transmission Time (min:sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1:45</td>
</tr>
<tr>
<td>2</td>
<td>2:30</td>
</tr>
<tr>
<td>3</td>
<td>17:00</td>
</tr>
<tr>
<td>4</td>
<td>1:00</td>
</tr>
<tr>
<td>5</td>
<td>0:20</td>
</tr>
<tr>
<td>6</td>
<td>1:10</td>
</tr>
<tr>
<td>7-9</td>
<td>0:25</td>
</tr>
</tbody>
</table>
been possible previously due to videoconferencing network delays). The corpsmen passed all required MSEL data and then resent all previous MSEL data at the request of ELB authorities. The entire evolution was smooth and successful. In addition, all required imagery files, including four 2.35 megabyte (MB) images, were successfully transmitted. The 2.35-MB files took 2-3 minutes to transmit, and all were viewed on the ECOC TMCS server.

**Yuma Node**

JMO-T participation at Yuma demonstrated far-forward message reach-back capability. JMO-T was assigned to operate from a WavePoint assigned to a Naval Research Lab (NRL) mobile commercial SATCOM system mounted in a HMMWV. This SATCOM link provided a 2-Mbps relay directly back to Area 41 at Camp Pendleton. EUT operational modification only required an IP change.

As in Area 41, all JMO-T messaging was handled by a 1/5 Marines corpsman. The system was operated from the back of a vehicle within 200 feet of the NRL SATCOM HMMWV. Individual patient encounter messages were transmitted within 5-10 seconds. The ECOC TMCS server was able to be browsed to confirm delivery.

<table>
<thead>
<tr>
<th>Table 2: Area 41 transmission times</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSEL Event</strong></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>401</td>
</tr>
<tr>
<td>341</td>
</tr>
<tr>
<td>408</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 3: Area 41 transmission times (continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MSEL Event</strong></td>
</tr>
<tr>
<td>-----------------</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>359</td>
</tr>
<tr>
<td>406</td>
</tr>
<tr>
<td>412</td>
</tr>
<tr>
<td>368</td>
</tr>
</tbody>
</table>
Five additional images, including two 1.35 MB images, were transmitted via File Transfer Protocol (FTP). Small files were transmitted in 10-20 seconds, and large files took 2:20 each. The only operational problem noted was a tendency for the Global Positioning System unit to stop sending position information when requested. This was traced to a loose cable on the Quatech serial port card; however, the cable was tightened, and the system returned to normal operation.

**FUTURE TRENDS**

ELB technology provided a number of excellent options for medical communications. When the network was not overwhelmed by the demands of videoconferencing, it provided an excellent method of collecting medical data—both TMCS textual data and images. During these times, Engineering Integrated Product Team (E-IPT) personnel working with data senders reported that TMCS data was sent in milliseconds, and the large files were transmitted in no more than five seconds. Data senders were able to use the handheld computers with ease. JMO-T participated in the longest leg of the exercise network by successfully sending TMCS and large data files from Yuma, Arizona.

The Libretto systems running Windows NT using 64 MB RAM performed satisfactorily; however, when the LDET, TCAS, Serial TCAS, and Medical Messaging Service (MMS) server were all running on one computer, the operation slowed significantly. One solution was to allow TCAS to speak TMCS (or Wave or any other medical software) in its native mode as a C++ object. Based on this experience, a more effective device for Echelon I use is a Windows CE computer, which weighs less than one pound, can easily fit into a Battle Dress Utilities (BDU) pocket, and provides resident software, a user-friendly screen, and a long-life, inexpensive battery.

**KB Prime (CG-1)**

KB Prime (CG-1) consisted of an amphibious assault exercise with a robust medical activity imbedded inside the main training action. The deployed forces consisted of three Regimental landing force size units supported by appropriate Level III medical care both ashore and afloat. The medical force during the CG-1 phase included two BASs, two STPs, one SC, the USS Essex (LHD-2), which served as a Casualty Receiving Treatment Ship (CRTS), the USNS Mercy (AH-19) hospital ship, and a Fleet Hospital (FH). Plans were for roughly 500 total casualties in five days.

The medical play in KB Prime was robust enough to provide a rich opportunity to observe how the technologies provided support to the user in an operational setting. These results were then used as a baseline for follow-up exercises. Both the WaveLAN and JINC demonstrated their primary intended functions of mobile tactical networking capacity. The WaveLAN system provided superior bandwidth and full wireless LAN capabilities. The JINC provided tactical networking over low bandwidth military radio systems. The primary objectives and results are provided in Tables 4 and 5.

**Recommendations**

Based on achievement of the stated objectives, the following recommendations are provided for continued wireless networking development:

- WaveLAN technology appears sufficiently mature to warrant use as a replacement for wired networking at field MTFs; and
- A prototype network configuration to support an SC should be devised and prepared for testing.
The following recommendations are provided for continued TCAS software development:

- As demonstrated in the exercise, TCAS was based on a C++ Windows 95/98/NT executable program. Operational experience with the Libretto NT system at Echelon I showed the need for a smaller, lighter computing system to support this highly mobile group. The Windows CE operating environment appears most suited to this requirement. Port TCAS software into the CE environment is recommended.

- The greatest asset (and liability) of the TCAS/J software is its flexibility. Programming the various communications servers, forwarding rules, and message formats is similar to programming a full-featured network router. This implies that a TCAS operator must be both computer literate and network knowledgeable. Simplification of the user interface, perhaps with more graphical network connection screens, appears necessary. In addition, the software should feature some type of “system lock” that will keep all settings under a password-controlled environment so that an inexperienced operator cannot change them by accident.

- Continued developmental work is needed to incorporate the full range of medical database-specific messages into TCAS. Message delivery in the exercise was achieved via a complicated process involving multiple serial port data exchange and encoding. This process can be streamlined by the provision of a medical system communications server to the TCAS software developers so that they can test their message servers directly.

Testing and evaluation of the JMO-T ACTD have produced tangible evidence for the military utility of mobile telemedicine. Results from Demonstration I indicate that the essential

---

**Table 4: Networking objectives and results for KB 99**

<table>
<thead>
<tr>
<th>Objective</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Achieve medical in-transit patient visibility through the use of the WaveLAN network</td>
<td>Achieved. When the WaveLAN network was in full operation, the delivery of LDET messages occurred in 3-5 seconds. Other ELB functions (i.e., VTC) significantly slowed network operations.</td>
</tr>
<tr>
<td>Achieve medical imagery file transfer using WaveLAN technology</td>
<td>Achieved. Multiple images were transferred using standard FTP programs.</td>
</tr>
<tr>
<td>Achieve medical in-transit patient visibility through the use of JINC network and tactical radio systems</td>
<td>Not achieved. While LDET messages were delivered between units on the network, LDET input did not reach the TMCS master server at Space and Warfare (SPAWAR). As a result, full patient in-transit visibility was not achieved.</td>
</tr>
<tr>
<td>Achieve MeWS medical messaging and file transfer capability through the use of JINC network and tactical radio systems</td>
<td>Achieved. Two test files were transferred between SC and USNS Mercy.</td>
</tr>
<tr>
<td>Demonstrate internetting for units on Different RFs</td>
<td>Partially achieved. Messages received over SINCGARS net were forwarded via HF net but required manual intervention for delivery. SINCGARS-to-WaveLAN automated delivery was accomplished during the UW exercise.</td>
</tr>
</tbody>
</table>
data collection and dissemination requirements of JMO-T can be met consistently, reliably, and cost effectively.

The ACTD promises the potential to demonstrate technology-enhanced data collection and dissemination of information through the use of quality software and robust communications infrastructure. Through its efforts, the JMO-T ACTD has developed a consistent pattern of progression. From an initial state of uncoordinated, service-unique solutions to the building of an overall architectural framework, this architectural solution is being developed and refined by several different concepts. These concepts have been and will continue to be assessed for operational and technical feasibility throughout Demonstration II, which begins with Cobra Gold in April–May 2001 and FOAL Eagle in the Fall. The results from these operational and technical assessments will ultimately lead to the development and insertion of an emerging JMO-T architecture, which will encompass the “web” phase of the JMO-T ACTD e-commerce effort.

Operational Effectiveness

The effectiveness of the systems used in the exercise was demonstrated by typical users, who operated them in a realistic operational environment. The Capability Package demonstrated the ability to collect both patient encounter and Annex Q-type information; however, it did not meet the threshold values established by the Performance Integrated Product Team (P-IPT) for transmitting that information to the theater medical command. The purpose of the exercise was to move patients through the evacuation system, and most decisions that needed to be made could be made without referring to the information stored on the TMCS server. In fact, most of the decisions did not require the type of information that was reported, and therefore, the staff instead used other data. As stated in the feedback questionnaires, the Marine Expeditionary Force (MEF) and Third Fleet Surgeons neither relied on the data provided by TMCS nor trusted its timeliness or reliability.

CONCLUSIONS AND LESSONS LEARNED

Our re-engineering project with mobile telemedicine provided many insights into how the military actually deals with BPR on an enterprise-wide basis. The project uncovered the ideological methodologies used to guide BPR efforts and the technologies used to help implement mobile e-commerce applications for telemedicine. The military radically redesigned T&E processes to improve overall performance. At the same time, it used technology and data warehouse methods to decentralize data management for increased information sharing, easier access data by those who need it, and more timely delivery of data, products, and services. Thus, the BPR strategy uses an approach to process improvement with information technology and mobile e-commerce applications as a complementary support mechanism.

It is realized that JMO-T ACTD must continue to provide telemedicine service to its military personnel, improve strategic awareness of the battlespace, and provide excellent information services to commanders and end-users during times of both peace and war. The literature on BPR has not helped us in this regard. In addition, it provides little insight into the complexities of dealing with military re-engineering and information re-engineering simultaneously. Each branch of the armed forces has a different set of problems to deal with. Books and periodicals can only provide basic ideas; therefore, we believe that we must develop our own methodology for dealing with change and process improvement.
Lesson One

Military commanders, such as the CINC, should be knowledgeable and interact with the operations of JMO-T ACTD. The T&E members believe that all of the functional areas have a hands-on approach to IT. JMO-T ACTD used IT to redefine its business processes and adopt mobile e-commerce principles to telemedicine. They found that it is much easier to teach the CINC and top military commanders the fundamentals of technology than it is to teach technology people about strategic management of the battlespace. In addition, JMO-T ACTD also serves the medical needs of the military’s internal personnel.

Lesson Two

If business processes are dependent on timely, accurate, and complete information, business re-engineering should be approached with a strategy that includes information re-engineering. In the contemporary military environment, information is especially important because it is very information intensive; hence, T&E choose a dual strategy of business and information re-engineering as JMO-T ACTD’s ideological approach to BPR.

Lesson Three

BPR should be adopted based on a military need and not because “everyone else is doing it.” T&E chose to redesign JMO-T ACTD processes because they were concerned about JMO-T ACTD’s reputation with the top military commanders and the Senate Committees that fund its operations. Before re-engineering, no method for tracking soldiers’ medical care in the field or during medical evacuation existed. The costs associated with medical evacuations was prohibitive, both in terms of lives and money. Re-engineering the process through JMO-T ACTDs adoption of mobile e-commerce technologies allows for on-site medical treatment without the fear of helicopter medical evacuations under enemy fire and/or during poor weather conditions. Medical evacuations lead to a long cycle time from receipt of the wounded in the field until they could reach proper medical care. Long cycle times translate into increased mortality and morbidity for military personnel. Because JMO-T ACTD allows “real-time” treatment, T&E feels that telemedicine and mobile e-commerce technologies provide an edge for treating casualties. T&E believes that BPR has given the military that edge by decreasing cycle times and improving information sharing.

Lesson Four

T&E believes that JMO-T ACTD must develop an independent JTF capability package in order to lead the IT re-engineering effort. JMO-T ACTD clients are the entire military. Because the IT capability package manages information flow throughout the military battlespace, it must be able to work with military commanders and end-users to “show them the way.” In other words, IT people in the JTF must provide a data view of the entire military organization. They know how the information is distributed to all departments and operational areas and are in an ideal position to work with information users as changes in business processes occur. This is a full-time job that requires individuals who are dedicated to carrying out this mission.

Lesson Five

We feel that BPR projects require support from top commanders and those involved along the process path to succeed. If top military management does not visibly support the
BPR effort of JMO-T ACTD, politics will destroy the project. Most people are afraid of change, and given the opportunity to resist change, many will do just that. Moreover, changing the way that business is conducted will not be tolerated without top-level approval because top military officials are in charge. T&E believes that if those involved in the process are not part of the project, they will resist changes and most likely sabotage the BPR effort. After all, they are the ones who will most likely be affected by these changes.

Lesson Six

T&E found that very few military personnel or officers know the overall military operational process; however, T&E believes that the JTF capability package must support an innovative approach to telemedicine improvement projects if it to serve all members of the military. T&E concluded, therefore, that top military management should form a JTF department and help it to gain knowledge about the military operations that it serves. The best strategy is to assign top military officers to the JTF department to add operational and strategic knowledge and experience.

Lesson Seven

T&E believes that it is important to choose a project that must work, so that its success can be sold to the rest of the company. Success is hard to resist. If a project is very successful, it will be much easier to get other departments and operational areas involved in BPR. Because the JMO-T ACTD project worked, it allowed the military to decentralize its information processing. Medical information processing was taking too long and negatively impacting soldier well-being; therefore, T&E took action and decided to embark on a major BPR project to rethink the existing medical information and inventory system and to decentralize medical treatment in the battlespace. This was a critical process and a risky venture, but the military had no choice. The JMO-T ACTD project succeeded because the potential for excellent results far outweighed the risk.

DISCUSSION AND CONCLUSIONS

This research focused on developing a holistic model of transformation. The model synthesizes current thinking on transformation into a holistic model and also explains the integrative influence of vision on the other four components of the model. The model was tested by T&E on the JMO-T ACTD. JMO-T ACTD has developed a very successful training program and is very aware of the importance of planned change. Top military officials are actively involved in change and are committed to people development through learning. The model served an applied purpose by allowing us to see how well the military organization fit current theory. The model also fit a theoretical purpose by organizing a holistic, comprehensive framework. Accordingly, we have organized and synthesized the literature into five interrelated components that act as a fundamental guide for research. The model also helped us to identify a theoretical link and apply it to the internal operations of mobile e-commerce and telemedicine in the military.

The views expressed in this paper are those of the authors and do not reflect the official policy or position of the Department of the Army, Department of the Navy, Department of Defense, or the U.S. Government.
## APPENDIX-ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTD</td>
<td>Advanced Concept Technology Demonstration</td>
</tr>
<tr>
<td>AE</td>
<td>Aeromedical Evacuation</td>
</tr>
<tr>
<td>AEF</td>
<td>Air Expeditionary Force</td>
</tr>
<tr>
<td>AELT</td>
<td>Aeromedical Evacuation Liaison Team</td>
</tr>
<tr>
<td>AFB</td>
<td>Air Force Base</td>
</tr>
<tr>
<td>AFFOR</td>
<td>Air Force Forces</td>
</tr>
<tr>
<td>ALE</td>
<td>Automatic Link Establishment</td>
</tr>
<tr>
<td>AMC</td>
<td>Air Mobility Command</td>
</tr>
<tr>
<td>AoA</td>
<td>Analysis of Alternatives</td>
</tr>
<tr>
<td>AOR</td>
<td>Area of Responsibility</td>
</tr>
<tr>
<td>ARFOR</td>
<td>Army Forces</td>
</tr>
<tr>
<td>ASMB</td>
<td>Area Support Medical Battalion</td>
</tr>
<tr>
<td>ASTS</td>
<td>Air Medical Staging Squadron</td>
</tr>
<tr>
<td>ATH</td>
<td>Air Transportable Hospital</td>
</tr>
<tr>
<td>BAS</td>
<td>Battalion Aid Station</td>
</tr>
<tr>
<td>BDU</td>
<td>Battle Dress Utilities</td>
</tr>
<tr>
<td>CG-1</td>
<td>KB Prime</td>
</tr>
<tr>
<td>CIA</td>
<td>Care in the Air</td>
</tr>
<tr>
<td>CINC</td>
<td>Commander-in-Chief</td>
</tr>
<tr>
<td>CJTF</td>
<td>Commander Joint Task Force</td>
</tr>
<tr>
<td>COA</td>
<td>Course of Action</td>
</tr>
<tr>
<td>COI</td>
<td>Critical Operational Issue</td>
</tr>
<tr>
<td>COIC</td>
<td>Critical Operational Issues Criteria</td>
</tr>
<tr>
<td>CONOPS</td>
<td>Concept of Operations</td>
</tr>
<tr>
<td>COP</td>
<td>Common Operating Picture</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-off-the-shelf</td>
</tr>
<tr>
<td>CPX</td>
<td>Command Post Exercise</td>
</tr>
<tr>
<td>CRTS</td>
<td>Casualty Receiving Treatment Ships</td>
</tr>
<tr>
<td>CSH</td>
<td>Combat Support Hospital</td>
</tr>
<tr>
<td>CSI</td>
<td>Command System, Incorporated</td>
</tr>
<tr>
<td>CSS</td>
<td>Combat Service Support</td>
</tr>
<tr>
<td>C2</td>
<td>Command and Control</td>
</tr>
<tr>
<td>C4I</td>
<td>Command, Control, Communications, Computers and Intelligence</td>
</tr>
<tr>
<td>DAMA</td>
<td>Demand Assigned Multiple Access</td>
</tr>
<tr>
<td>DC</td>
<td>Direct Current</td>
</tr>
<tr>
<td>DC II</td>
<td>Desert Care II</td>
</tr>
<tr>
<td>DEPMEDS</td>
<td>Deployable Medical Systems</td>
</tr>
<tr>
<td>DNBI</td>
<td>Diseases and Non-Battle Injuries</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
<tr>
<td>DII COE</td>
<td>Defense Information Infrastructure, Common Operating Engine</td>
</tr>
<tr>
<td>ECOC</td>
<td>Enhanced Combat Operations Center</td>
</tr>
<tr>
<td>E-IPT</td>
<td>Engineering Integrated Product Team</td>
</tr>
<tr>
<td>ELB</td>
<td>Extending the Littoral Battlespace</td>
</tr>
<tr>
<td>EMEDS</td>
<td>Expeditionary Medical Service</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>EMT</td>
<td>Emergency Medical Treatment</td>
</tr>
<tr>
<td>ENT</td>
<td>Ear, Nose, and Throat</td>
</tr>
<tr>
<td>EUT</td>
<td>End User Terminal</td>
</tr>
<tr>
<td>FBE-E</td>
<td>Fleet Battle Experiment - Echo</td>
</tr>
<tr>
<td>FH</td>
<td>Fleet/Field Hospital</td>
</tr>
<tr>
<td>FMSS</td>
<td>Field Medical Surveillance System</td>
</tr>
<tr>
<td>FSMC</td>
<td>Forward Support Medical Company</td>
</tr>
<tr>
<td>FST</td>
<td>Forward Surgical Team</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>FTX</td>
<td>Field Training Exercise</td>
</tr>
<tr>
<td>FY</td>
<td>Fiscal Year</td>
</tr>
<tr>
<td>GiCOD</td>
<td>Good Idea Cut-Off Date</td>
</tr>
<tr>
<td>GOTS</td>
<td>Government-off-the-shelf</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HF</td>
<td>High Frequency</td>
</tr>
<tr>
<td>HM</td>
<td>Hospital Corpsman</td>
</tr>
<tr>
<td>HMMWV</td>
<td>High-Mobility Multipurpose Wheeled Vehicle</td>
</tr>
<tr>
<td>HQ</td>
<td>Headquarters</td>
</tr>
<tr>
<td>ICU</td>
<td>Intensive Care Unit</td>
</tr>
<tr>
<td>IM</td>
<td>Information Management</td>
</tr>
<tr>
<td>INMARSAT</td>
<td>International Maritime Satellite</td>
</tr>
<tr>
<td>IPT</td>
<td>Integrated Product Team</td>
</tr>
<tr>
<td>ISDN</td>
<td>Integrated Services Digital Network</td>
</tr>
<tr>
<td>JHSS</td>
<td>Joint Health Service Support</td>
</tr>
<tr>
<td>JINC</td>
<td>Joint Internet Controller</td>
</tr>
<tr>
<td>JMedSAF</td>
<td>Joint Medical Semi-Automated Forces</td>
</tr>
<tr>
<td>JMeWS</td>
<td>Joint Medical Workstation</td>
</tr>
<tr>
<td>JMO-T</td>
<td>Joint Medical Operations-Telemedicine</td>
</tr>
<tr>
<td>JSIMS</td>
<td>Joint Simulation Systems Acquisition Program</td>
</tr>
<tr>
<td>JTF</td>
<td>Joint Task Force</td>
</tr>
<tr>
<td>JTTP</td>
<td>Joint Tactics, Techniques, and Procedures</td>
</tr>
<tr>
<td>JV 2010</td>
<td>Joint Vision 2010</td>
</tr>
<tr>
<td>kb</td>
<td>kilobyte</td>
</tr>
<tr>
<td>Kbps</td>
<td>Kilobytes per second</td>
</tr>
<tr>
<td>KB 99</td>
<td>Kernel Blitz 99</td>
</tr>
<tr>
<td>LAN</td>
<td>Local Area Network</td>
</tr>
<tr>
<td>LDET</td>
<td>Lightweight Data Entry Tool</td>
</tr>
<tr>
<td>MARFOR</td>
<td>Marine Forces</td>
</tr>
<tr>
<td>MASF</td>
<td>Mobile Air Staging Facility</td>
</tr>
<tr>
<td>MB</td>
<td>Megabyte</td>
</tr>
<tr>
<td>Mbps</td>
<td>Million bits per second</td>
</tr>
<tr>
<td>MCM</td>
<td>Military Command</td>
</tr>
<tr>
<td>MEDEVAC</td>
<td>Medical Evacuation</td>
</tr>
<tr>
<td>MEF</td>
<td>Marine Expeditionary Force</td>
</tr>
<tr>
<td>MES</td>
<td>Medical Equipment Sets</td>
</tr>
<tr>
<td>MeWS</td>
<td>Medical Workstation</td>
</tr>
<tr>
<td>MHS</td>
<td>Military Health System</td>
</tr>
<tr>
<td>MHz</td>
<td>Megahertz</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>MIEP</td>
<td>Medical Information Engineering Prototype</td>
</tr>
<tr>
<td>MILNET</td>
<td>Military Network</td>
</tr>
<tr>
<td>mm</td>
<td>millimeters</td>
</tr>
<tr>
<td>MMS</td>
<td>Medical Messaging Service</td>
</tr>
<tr>
<td>MOE</td>
<td>Measures of Effectiveness</td>
</tr>
<tr>
<td>MOP</td>
<td>Measures of Performance</td>
</tr>
<tr>
<td>MTF</td>
<td>Medical Treatment Facility</td>
</tr>
<tr>
<td>M&amp;S</td>
<td>Modeling and Simulation</td>
</tr>
<tr>
<td>MSEL</td>
<td>Master Scenario Events List</td>
</tr>
<tr>
<td>M3B</td>
<td>Mobile Medical Monitor (B)</td>
</tr>
<tr>
<td>NAVFOR</td>
<td>Navy Forces</td>
</tr>
<tr>
<td>NIPRNET</td>
<td>Unclassified Internet Protocol Router Network</td>
</tr>
<tr>
<td>NRL</td>
<td>Naval Research Lab</td>
</tr>
<tr>
<td>OR</td>
<td>Operating Room</td>
</tr>
<tr>
<td>OTH</td>
<td>Over-the-Horizon</td>
</tr>
<tr>
<td>PACOM</td>
<td>Pacific Command</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PCMCIA</td>
<td>Personal Computer Memory Card International Association</td>
</tr>
<tr>
<td>PIC</td>
<td>Personal Information Carrier</td>
</tr>
<tr>
<td>P-IPT</td>
<td>Performance Integrated Product Team</td>
</tr>
<tr>
<td>PROFIS</td>
<td>Professional Filler System</td>
</tr>
<tr>
<td>PM 99</td>
<td>Patriot Medstar 99</td>
</tr>
<tr>
<td>PW 99</td>
<td>Pacific Warrior 99</td>
</tr>
<tr>
<td>RF</td>
<td>Radio Frequency</td>
</tr>
<tr>
<td>RFTA</td>
<td>Reserve Forces Training Area</td>
</tr>
<tr>
<td>SAIC</td>
<td>Science Applications International Corporation</td>
</tr>
<tr>
<td>SATCOM</td>
<td>Satellite Communications</td>
</tr>
<tr>
<td>SC</td>
<td>Surgical Company</td>
</tr>
<tr>
<td>SINCGARS</td>
<td>Single-Channel Ground and Airborne Radio System</td>
</tr>
<tr>
<td>SPAWAR</td>
<td>Space and Warfare</td>
</tr>
<tr>
<td>SSC SD</td>
<td>Systems Center, San Diego</td>
</tr>
<tr>
<td>STP</td>
<td>Shock Trauma Platoon</td>
</tr>
<tr>
<td>TAMC</td>
<td>Tripler Army Medical Center</td>
</tr>
<tr>
<td>TCAS</td>
<td>Team Care Automation System</td>
</tr>
<tr>
<td>TCEA</td>
<td>Theater Clinical Encounter Application</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>T&amp;E</td>
<td>Test and Evaluation</td>
</tr>
<tr>
<td>T&amp;E-IPT</td>
<td>Test and Evaluation Integrated Product Team</td>
</tr>
<tr>
<td>TMCS</td>
<td>Theater Medical Core Services</td>
</tr>
<tr>
<td>TMIP</td>
<td>Theater Medical Information Program</td>
</tr>
<tr>
<td>TTT</td>
<td>Theater Telemedicine Team</td>
</tr>
<tr>
<td>T2P2</td>
<td>Theater Telemedicine Prototype Program</td>
</tr>
<tr>
<td>UHF</td>
<td>Ultra-high frequency</td>
</tr>
<tr>
<td>USAF</td>
<td>U.S. Air Force</td>
</tr>
<tr>
<td>USFK</td>
<td>U.S. Forces, Korea</td>
</tr>
<tr>
<td>UW</td>
<td>Urban Warrior</td>
</tr>
<tr>
<td>VHF</td>
<td>Very High Frequency</td>
</tr>
<tr>
<td>VTC</td>
<td>Video Teleconference</td>
</tr>
</tbody>
</table>
REFERENCES


telemedicine: Technology-management implications. IEEE Transactions on Engi-
Review, 2(40), 96-124.
Rasberry, M. S. (1998). The theater telemedicine prototype project: Multimedia email in the
pacific. Proceedings of the 31st Hawaii International Conference on System Sciences
(HICSS-31), Maui, Hawaii.
Communications of the ACM, 43(2).
80-98.
Motors of Canada. Journal of Information Technology Cases and Applications, 1(3),
45-60.
Management Review, 40.
Planning, 26(6), 22-40.
model using telemedicine as a case. Proceedings of the 31st Hawaii International
Conference on System Sciences (HICSS-31), Maui, Hawaii.
Communications of the ACM, 6(41), 96-102.
of environment. Total Quality Management, 4(9), 241-245.
Section IV

Applications of Unified Modeling Language
Chapter XV

How Complex Is the Unified Modeling Language?

Keng Siau
University of Nebraska–Lincoln, USA

Qing Cao
University of Missouri–Kansas City, USA

Unified Modeling Language (UML) has emerged as the software industry’s dominant modeling language. It is the de facto modeling language standard for specifying, visualizing, constructing, and documenting the components of software systems. Despite its prominence and status as the standard modeling language, UML has its critics. Opponents argue that it is complex and difficult to learn. Some question the rationale of having nine diagramming techniques in UML and the raison d’être of those nine techniques in UML. Others point out that UML lacks a comprehensive methodology to guide its users, which makes the language even more convoluted. A few studies on UML can be found in the literature. However, no study exists to provide a quantitative measure of UML complexity or to compare UML with other object-oriented techniques. In this research, we evaluate the complexity of UML using complexity metrics. The objective is to provide a reliable and accurate quantitative measure of UML complexity. A comparison of the complexity metrical values of UML with other object-oriented techniques was also carried out. Our findings suggest that each diagram in UML is not distinctly more complex than techniques in other modeling methods. But as a whole, UML is very complex—2-11 times more complex than other modeling methods.

INTRODUCTION

Unified Modeling Language (UML) is a visual modeling language for modeling system requirements, describing designs, and depicting implementation details. Grady Booch, Jim Rumbaugh, and Ivars Jacobson, known collectively as the “three Amigos” at Rational Software Corp, spearheaded development of UML in the mid-1990s. Unified Modeling Language (UML) borrows concepts from a large number of different methods, and is tailored specifically for object-oriented systems development. Soon after its
inception, UML emerged as the software industry’s dominant modeling language. UML is not only the de facto modeling language standard for specifying, visualizing, constructing, and documenting the components of software systems, but it has also been accepted by the Object Management Group (OMG) as a standard language for object-oriented analysis and design. In addition, UML has been proposed for standardization by the International Standards Organization (ISO) and approval is anticipated sometime in 2001.

Many of the language’s supporters claim that UML’s simplicity is its chief benefit (Kobryn, 1999) and argue that UML uses simple, intuitive notations that are understandable by nonprogrammers. This is in line with a recent study by Fedorowicz and Villeneuve (1999) who surveyed vendors and developers regarding OO systems analysis, design, and programming. Their results suggested that the OO approach is intuitive and easy to use/comprehend for vendors and developers alike. By offering a common blueprint language, UML also relieves developers of the proprietary ties that are so common in this industry. Major vendors including IBM, Microsoft, and Oracle are brought together under the UML umbrella. If developers, customers, and implementers can all understand a single modeling language instead of the few dozen OO techniques that existed before UML (Siau, 1999), they are more likely to agree on the intended functionality of the modeling techniques and constructs, thereby improving the communication process among the stakeholders and enhancing their chances of creating an application that truly addresses business problems.

UML, nevertheless, has its fair share of critics. Halpin and Bloesch (1999) pointed out that although UML facilitates the transition to object-oriented code, its implementation concerns render it less suitable for developing and validating a conceptual model with domain experts. Beckwith and Moore (1998) studied the use of deployment diagrams in architecture modeling and discovered problems with multi-tasking. In contrast to Kobryn (1999) who stated that UML is simple to use, Douglass (1998) argued that UML is somewhat large and complex, which can be daunting to novice users. He called for future research to examine the questions: (1) How easy is it to understand UML? and (2) How easy is it to use UML?

This study investigates the complexity of UML. Complexity is a key measure of the effectiveness of a language because complexity directly affects the learn-ability and ease-of-use of the language. As mentioned by Booch, Rumbaugh, and Jacobson (1999a), UML is still in its infancy and there is a need for improvement in the analysis of UML and in understanding its use and functionality.

The rest of the chapter is organized as follows: the following section reviews UML and is followed by a section discussing various approaches to measuring complexity, including empirical studies and different complexity metrics. The complexity of UML is then analyzed using the complexity metrics proposed by Rossi and Brinkkemper (1994). The research findings are then discussed and a comparison is then made between the complexity of UML and other modeling techniques. The last section summarizes the research, lists the limitations of the study, and suggests future research directions.

OBJECT ORIENTATION AND UML

Object-orientation (OO) is the new paradigm for systems development (Johnson & Hardgrave, 1999). The first few years of the 1990s saw the blossoming of nearly 50 different object oriented methods. The abundance of OO methods caused great confusion among users. Standardization of OO was needed to curtail the chaos and the Unified Modeling Language (UML) was the result.
The Unified Modeling Language (UML) includes nine modeling diagrams. They are class diagrams, use-case diagrams, state-chart diagrams, activity diagrams, sequence diagrams, collaboration diagrams, object diagrams, components diagrams, and deployment diagrams (Booch et al., 1999a). These diagrams model both the static view and the dynamic view of the system. While class diagrams, object diagrams, component diagrams, and deployment diagrams model the static view of the system, statechart diagrams, activity diagrams, sequence diagrams, and collaboration diagrams depict the dynamic view of the system. Each diagram shows a different aspect of the system, and is by design incomplete.

A class diagram shows a set of classes, interfaces, and collaborations and their relationships. An object diagram depicts static “snapshots” of the elements within a system, showing objects’ structure, attributes, and relationships to one another. An activity diagram shows the flow of control from one activity to the next, and a use-case diagram illustrates how elements outside the system use the system. For instance, the internal workings of a new payroll system would be shown in an activity diagram, whereas external actors, such as the mail order department, would appear in a use-case diagram. Sequence and collaboration diagrams show interactive processes; i.e., developers see not only objects and classes, but also the messages that pass between them. A statechart diagram shows a state machine, consisting of states, transitions, events, and activities. Finally, component and deployment diagrams show the physical or implementation view of the system (including executables, libraries, and interfaces).

In this study, we measure the complexity of each diagram in UML and the complexity of UML as a whole.

**MEASURING COMPLEXITY**

Siau and Rossi (1998) completed a comprehensive review of the approaches for evaluating modeling methods. They broadly categorized evaluation methods into two categories—empirical and non-empirical. Empirical studies include those conducted on ease-of-use of data modeling methods (e.g., Chan & Lim, 1998; Chan, Siau, & Wei, 1998; Siau, Chan, & Wei, 1995; Chan, Wei, & Siau, 1993; Batra, Hoffer, & Bostrom, 1990). For instance, Chan et al. (1993) conducted an experiment to compare the ER and relational models using textual query languages specifically designed for each model respectively. Siau et al. (1995) compared the relational and ER models using visual query languages. Non-empirical evaluation approaches include feature comparison, metamodeling, paradigmatic analysis, and metrics analysis (e.g., Siau, 2000; Siau, 1999; Rossi & Brinkkemper, 1994; Wand & Weber, 1993). In this study, we use the metrics analysis to evaluate UML.

**Complexity Measures**

The quality of a complexity measure rests on its explanatory power and applicability. Explanatory power refers to the measure’s ability to explain the interrelationships among complexity, quality, and other programming and design parameters. Applicability refers to the degree to which the measure can be applied to improve the quality of work during the design, coding, and testing stages. There are several major complexity measures in the literature.

**Lines of Code**

Lines of code (LOC) is a count of instruction statements. Because LOC count represents the program size and complexity, it is not a surprise that the more lines of code there are in a
program, the more defects should be expected. A concave relationship between number of defects and module size was suggested. Withrow (1990) examined modules written in Ada for a large project at Unisys and confirmed the concave relationship between defect density and module size. The author argued that there might be an optimal program size that could lead to the lowest defect rate. As module size becomes large, the complexity increases to a level beyond a programmer’s immediate span of control and total comprehension.

**Halstead’s Software Science**

Halstead (1977) developed a system of equations expressing the total vocabulary, overall program length, potential minimum volume for an algorithm, program level (a measure of software complexity), program difficulty, and other features such as development effort and projected number of faults in the software. Halstead metrics are static metrics, ignoring the huge variations in fault rates observed in software products and among modules.

**Cyclomatic Complexity**

The cyclomatic complexity measure by McCabe (1976) is designed to indicate a program’s testability and understandability. The measure provides the classical graph theory cyclomatic number, indicating the number of regions in a graph. The cyclomatic complexity metric is additive. The complexity of several graphs that form a group is equal to the sum of the individual graph’s complexities.

**Structure Metrics**

Lines of code, Halstead’s software science, and McCabe’s cyclomatic complexity metrics that measure module complexity assume implicitly that each program module is a separate entity (Kan, 1995). Structure metrics try to take into account the interactions between modules in a product or system and quantify such interactions. Weyuker (1988) examined and compared four complexity measures—Lines of code, Halstead’s software science, McCabe’s cyclomatic complexity metrics, and structure metrics. She discussed the proper use of structure metrics in object-oriented software evaluation and called for a more rigorous study on complexity measures, which would lead to the definition of good meaningful software complexity measures. Many approaches in structure metrics have been proposed. Some examples include system-partitioning measures by Belady and Evangelisti (1981), information flow metrics by Henry and Kafura (1981), and stability measures by Yau and Collofello (1980). More recently, Rossi and Brinkkemper (1994) introduced seventeen complexity metrics for systems development methods and techniques.

In this study, we apply the structural metrics proposed by Rossi and Brinkkemper (1994) to evaluate UML. The reason we choose complexity metrics proposed by Rossi and Brinkkemper is two-fold. First, the metrics are by far one of the most comprehensive complexity metrics proposed. Second, the metrics have been used to evaluate the ease-of-use of object-oriented techniques. For example, Rossi and Brinkkemper applied the metrics to a number of modeling techniques. The availability of other modeling techniques’ metrical values enables us to compare UML with other modeling approaches.

**COMPLEXITY OF UML**

In this section, we will discuss how to apply Rossi and Brinkkemper’s (1994) complexity metrics to UML. First, we explain a common method modeling language—OPRR
(Object, Property, Relationship, Role) method modeling language. Second, we briefly describe the seventeen metrics proposed by Rossi and Brinkkemper. Finally, we apply the seventeen complexity metrics to evaluate UML.

The Method Modeling Language

According to Rossi and Brinkkemper (1994), a model of a technique can be defined as a six-tuple $M = \{O, P, R, X, r, p\}$ based on OPRR (Object, Property, Relationship, Role) method modeling language, where

- $O$ is a finite set of object types—*Object* is a thing that exists on its own. Examples of objects are classes, components, etc.
- $P$ is a finite set of property types—*Properties* are characteristics of meta-types. Examples of properties are class names, component names, attributes, etc.
- $R$ is a finite set of relationship types—*Relationship* is an association between two or more objects. Examples of relationships are associations between classes, generalizations, dependencies, etc.
- $X$ is a finite set of role types—*Role* is the name of a behavior of an object participating in a relationship.
- $r$ is a mapping $r : R \rightarrow \{x \mid x \in \wp(\wp(O) - \{\emptyset\)) \land n(x) \geq 2\}$ where $n(x)$ is the multiplicity of $x$ and $\wp(O)$ is the powerset of set $O$—$r$ maps a relationship type to a member of the set of powersets of role types and powersets of objects.
- $p$ is a partial mapping $p : NP \rightarrow \wp(P)$ where $NP = \{O \cup R \cup X\}$ is the set of non-property types—$p$ is the partial mapping from the non-property types to all subsets of property types.

Definitions of $\{O, P, R, X, r, p\}$ for UML diagrams were derived accordingly. Because of the length of the definitions, they are not included as part of this chapter.

Complexity Metrics

Rossi and Brinkkemper (1994) proposed a set of seventeen complexity metrics. The seventeen metrics were divided into three categories, i.e., independent measures, aggregate metrics, and method-level metrics.

**Independent Metrics**

The independent measures/metrics describe the individual characteristics of techniques. There are nine metrics in this category.

**Definition 1:** $n(O_t)$ is the count of object types per technique.

This metric demonstrates the number of individual object types used to specify object systems.

**Definition 2:** $n(R_t)$ is the count of relationship types per technique.

It indicates the number of concepts that are used for describing connections between objects.

**Definition 3:** $n(P_t)$ is the count of property types per technique.

This metric shows properties used by diagramming techniques.

**Definition 4:** $P_o(M_t, o) = n(P_t(o))$

It is the count of the number of properties for a given object type.
Definition 5: \[
\overline{P}_O(M_T) = \frac{1}{n(O_T)} \sum_{o \in O_T} P_O(M_T, o)
\]

This metric is the average number of properties per object type.

Definition 6: \[
P_R(M_T, e) = n(p_T(e)) + \sum_{x \in R(e)} n(p(role(x))), \text{ where } e \in R_T
\]

Definition 6 is the number of properties of a relationship type and its accompanying role types.

Definition 7: \[
\overline{P}_R(M_T) = \frac{1}{n(R_T)} \sum_{e \in R_T} P_R(M_T, e)
\]

Definition 7 counts the average number of properties per relationship type. It shows the complexity of the interface between object types.

Definition 8: \[
R_O(M_T, o) = n \left\{ e \in R_T \bigg| o \in \bigcup_{x \in R(e)} \text{objects}(x) \right\}, \text{ where } o \in O_T
\]

Definition 8 gives the number of relationship types that can be connected to a certain object type.

Definition 9: \[
\overline{R}_O(M_T) = \frac{1}{n(O_T)} \sum_{o \in O_T} R_O(M_T, o)
\]

The last independent measure gives the average number of relationship types that can be connected to a given object type.

Aggregate Metrics

While independent measures described the individual characteristics of techniques, aggregate metrics are used to measure the overall complexity of the technique. This category consists of three metrics.

Definition 10: \[
C(M_T, o) = \frac{P_O(M_T, o)}{\sum_{e \in A} P_R(M_T, e)}, \text{ where } A = \left\{ x \in R_T \bigg| o \in \bigcup_{y \in R(e)} \text{objects}(y) \right\}
\]

The quotient indicates the division of work in this technique (i.e., are things described by their internal properties or by external connections).

Definition 11: \[
\overline{C}(M_T) = \frac{1}{n(O_T)} \sum_{o \in O_T} C(M_T, o)
\]

This metric shows the average complexity for the whole technique.
**Definition 12:** \( C'(M_T) = \sqrt{n(O_T)^2 + n(R_T)^2 + n(P_T)^2} \)

This metric gives the total conceptual complexity of a technique.

**Method-Level Metrics**

Methods here are treated as collections of individual techniques. For example, UML, as a method, consists of nine diagramming techniques. Definitions 13, 14, 15 give the cumulative complexities of a method in terms of object, relationship and property types.

**Definition 13:** \( n(O_M) = \sum_{T \in M} n(O_T) \)

The value for this metric is found by summing the object types in all the techniques in the method.

**Definition 14:** \( n(R_M) = \sum_{T \in M} n(R_T) \)

The value for this metric is computed by summing the relationship types in all the techniques in the method.

**Definition 15:** \( n(P_M) = \sum_{T \in M} n(P_T) \)

The value is derived by summing the property types in all the techniques in the method.

**Definition 16:** \( \overline{C}(M) = \frac{1}{n(O_M)} \sum_{T \in M} \sum_{o \in O_T} C(M_T, o) \)

This metric gives the division of work between objects and relationships in the whole method. It is the summation of individual objects and their properties and relationships in each technique.

**Definition 17:** \( C'(M) = \sqrt{n(O_M)^2 + n(R_M)^2 + n(P_M)^2} \)

The cumulative complexity can be defined as the cumulative value of each individual technique’s complexity.

**Complexity Metrics for UML**

The complexity of UML was computed using the Rossi and Brinkkemper (1994) metrics. The following two tables depict the values for UML.

Some of the more interesting results from Tables 1 and 2 are:

- Statechart diagrams possess the largest number of object types among the UML diagrams (i.e., metric 1). None of the diagrams in UML has more than 10 object types.
- Class diagrams have the largest number of counts for relationship types (i.e., metric 2). This is followed by component diagrams. The number of relationship types per technique varies from 1 to 18.
### Table 1: UML values for independent metrics

<table>
<thead>
<tr>
<th>UML Diagrams</th>
<th>Metric 1</th>
<th>Metric 2</th>
<th>Metric 3</th>
<th>Metric 4</th>
<th>Metric 5</th>
<th>Metric 6</th>
<th>Metric 7</th>
<th>Metric 8</th>
<th>Metric 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n (O₂)</td>
<td>n (R₂)</td>
<td>n (P₂)</td>
<td>Po (M₂, O)</td>
<td>P₀ (M₂, e)</td>
<td>P₀(M₂, o)</td>
<td>R₀(M₂, o)</td>
<td>R₀(M₂, e)</td>
<td></td>
</tr>
<tr>
<td>Class Diagrams</td>
<td>7</td>
<td>18</td>
<td>18</td>
<td>12</td>
<td>1.71</td>
<td>22</td>
<td>1.22</td>
<td>18</td>
<td>2.57</td>
</tr>
<tr>
<td>Use-Case Diagrams</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.83</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>Activity Diagrams</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>0.75</td>
<td>1</td>
<td>0.20</td>
<td>5</td>
<td>0.63</td>
</tr>
<tr>
<td>Sequence Diagrams</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>0.67</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>0.17</td>
</tr>
<tr>
<td>Collaboration Diagrams</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>8</td>
<td>1</td>
<td>0.25</td>
</tr>
<tr>
<td>Object Diagrams</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>1.67</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>0.33</td>
</tr>
<tr>
<td>State-Chart Diagrams</td>
<td>10</td>
<td>4</td>
<td>11</td>
<td>10</td>
<td>1</td>
<td>2</td>
<td>0.5</td>
<td>4</td>
<td>0.40</td>
</tr>
<tr>
<td>Component Diagrams</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>8</td>
<td>1</td>
<td>13</td>
<td>3.6</td>
<td>10</td>
<td>1.25</td>
</tr>
<tr>
<td>Deployment Diagrams</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>8</td>
<td>1.14</td>
<td>7</td>
<td>1.40</td>
</tr>
</tbody>
</table>

### Table 2: UML values for aggregate and method-level metrics

<table>
<thead>
<tr>
<th>UML Diagrams</th>
<th>Metrics 10</th>
<th>Metrics 11</th>
<th>Metrics 12</th>
<th>Metrics 13</th>
<th>Metrics 14</th>
<th>Metrics 15</th>
<th>Metrics 16</th>
<th>Metrics 17</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C (M₂, o)</td>
<td>C (M₂)</td>
<td>C (M₂)</td>
<td>Total n (O₂)</td>
<td>Total n (R₂)</td>
<td>Total n (P₂)</td>
<td>C (M)</td>
<td>C (M)</td>
</tr>
<tr>
<td>Class Diagrams</td>
<td>0.67</td>
<td>0.10</td>
<td>26.40</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Use-Case Diagrams</td>
<td>1</td>
<td>0.17</td>
<td>10.39</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Activity Diagrams</td>
<td>1</td>
<td>0.13</td>
<td>11.18</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Sequence Diagrams</td>
<td>0.80</td>
<td>0.13</td>
<td>7.87</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Collaboration Diagrams</td>
<td>0.57</td>
<td>0.14</td>
<td>8.12</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Object Diagrams</td>
<td>1</td>
<td>0.33</td>
<td>5.92</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>State-Chart Diagrams</td>
<td>0.91</td>
<td>0.09</td>
<td>15.39</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Component Diagrams</td>
<td>0.89</td>
<td>0.11</td>
<td>15.65</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
<tr>
<td>Deployment Diagrams</td>
<td>1</td>
<td>0.20</td>
<td>9.95</td>
<td>57</td>
<td>53</td>
<td>72</td>
<td>0.14</td>
<td>106</td>
</tr>
</tbody>
</table>

- Class, statechart, and component diagrams have more property types per technique than the rest of the UML diagrams (i.e., metric 3). Since component diagrams are special kinds of class diagrams (Booch, 1999), their high property type counts is not surprising.
- Class diagrams have a very high count for metric 6 compared to other diagrams in UML. In other words, class diagrams have the highest count for the number of properties of a relationship type and its accompanying role types.
- For metric 8, the number of relationship types that can be connected to a certain object type, class diagrams again have the largest count (i.e., 18).
- In terms of the complication in selecting the right connection between object types (i.e., metric 9), class diagrams have the highest count—double that of use-case, component and deployment diagrams, and over 10 times more complicated than sequence and collaborations diagrams. This is in line with some studies that found users having difficulty in selecting a relationship type between objects/entities (e.g., Siau et al., 1995, 1996, 1997).
- As for the complexity of a technique, Table 2 shows that class diagrams are the most complex to use, followed by component and state-chart diagrams (i.e., metric 12). The main reason for the high complexity value of class diagrams is their very high relationship type and property type counts compared to other diagramming techniques in UML. Again, the complexity of component diagrams is not surprising because they are special cases of class diagrams.
DISCUSSION

In this section, we discuss the results of the research by analyzing the complexity values we derived for UML, and more importantly, by comparing the complexity values of UML with the complexity values of other modeling methods derived by Rossi and Brinkkemper (1994).

Table 3 lists the complexity values of 36 modeling techniques from 14 different modeling methods calculated by Rossi and Brinkkemper (1994) and the nine complexity values for UML diagrams.

<table>
<thead>
<tr>
<th>Method</th>
<th>Technique</th>
<th>n (O)</th>
<th>n (R)</th>
<th>n (P)</th>
<th>(\bar{P}_O) (M)</th>
<th>(\bar{P}_R) (M)</th>
<th>(\bar{R}_O) (M)</th>
<th>(\bar{C}) (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demeter</td>
<td>Demeter</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>.5</td>
<td>1.33</td>
</tr>
<tr>
<td>Express</td>
<td>Express-G</td>
<td>5</td>
<td>6</td>
<td>13</td>
<td>2.20</td>
<td>4</td>
<td>2.67</td>
<td>.14</td>
</tr>
<tr>
<td>FUSION</td>
<td>Object Interaction Graph</td>
<td>2</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>3</td>
<td>3.75</td>
<td>7.35</td>
</tr>
<tr>
<td>FUSION</td>
<td>Object Model</td>
<td>4</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>2.25</td>
<td>.75</td>
<td>9.80</td>
</tr>
<tr>
<td>Goldkuhl</td>
<td>Activity Model</td>
<td>5</td>
<td>5</td>
<td>8</td>
<td>1.60</td>
<td>2.80</td>
<td>20</td>
<td>.55</td>
</tr>
<tr>
<td>JSP</td>
<td>Data Structure Diagram</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3.32</td>
<td></td>
</tr>
<tr>
<td>JSP</td>
<td>Program Structure Diagram</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>2.50</td>
<td>1.50</td>
<td>1.75</td>
<td>5.74</td>
</tr>
<tr>
<td>MOSES</td>
<td>Event Model</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>.66</td>
</tr>
<tr>
<td>MOSES</td>
<td>O/C Model</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>4.50</td>
<td>6</td>
<td>1.88</td>
<td>.67</td>
</tr>
<tr>
<td>OMT</td>
<td>Class Diagram</td>
<td>5</td>
<td>15</td>
<td>14</td>
<td>2</td>
<td>4.40</td>
<td>4</td>
<td>.91</td>
</tr>
<tr>
<td>OMT</td>
<td>Data Flow Diagram</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>1.67</td>
<td>3</td>
<td>2</td>
<td>.19</td>
</tr>
<tr>
<td>OMT</td>
<td>State Diagram</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1.5</td>
<td>1</td>
<td>2</td>
<td>.50</td>
</tr>
<tr>
<td>OOAD</td>
<td>Object-Oriented Analysis and Design</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>2.33</td>
<td>4</td>
<td>1</td>
<td>.37</td>
</tr>
<tr>
<td>OOAD</td>
<td>Object State Diagram</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>OOAD</td>
<td>Service Chart</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>2.67</td>
<td>2</td>
<td>.50</td>
<td>8.70</td>
</tr>
<tr>
<td>OODA</td>
<td>Class Diagram</td>
<td>3</td>
<td>9</td>
<td>10</td>
<td>4.67</td>
<td>5</td>
<td>1.44</td>
<td>.70</td>
</tr>
<tr>
<td>OODA</td>
<td>Module Diagram</td>
<td>10</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1.50</td>
</tr>
<tr>
<td>OODA</td>
<td>Object Diagram</td>
<td>1</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>4</td>
<td>.17</td>
</tr>
<tr>
<td>OODA</td>
<td>Process Diagram</td>
<td>2</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>OODA</td>
<td>State Transition Diagram</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3.25</td>
</tr>
<tr>
<td>OSA</td>
<td>Object Behavior Model</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>.38</td>
</tr>
<tr>
<td>OSA</td>
<td>Object Interaction Model</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2.50</td>
<td>.14</td>
</tr>
<tr>
<td>OSA</td>
<td>Object Relationship Model</td>
<td>5</td>
<td>7</td>
<td>11</td>
<td>1.80</td>
<td>2.20</td>
<td>1.57</td>
<td>.75</td>
</tr>
<tr>
<td>RTSA</td>
<td>Entity Relationship Attribute Diagram</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>2.33</td>
<td>1.67</td>
<td>.50</td>
<td>1.44</td>
</tr>
<tr>
<td>RTSA</td>
<td>Real-Time Structured Analysis</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>2.2</td>
<td>4.20</td>
<td>1</td>
<td>.30</td>
</tr>
<tr>
<td>RTSA</td>
<td>State Transition Diagram</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>.40</td>
</tr>
<tr>
<td>RTSA</td>
<td>Structure Chart</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>.67</td>
</tr>
<tr>
<td>Shlaer/Mellor</td>
<td>Action Data Flow Diagram</td>
<td>2</td>
<td>4</td>
<td>7</td>
<td>2.5</td>
<td>3</td>
<td>1.75</td>
<td>.30</td>
</tr>
<tr>
<td>Shlaer/Mellor</td>
<td>Information Structure Diagram</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>2.5</td>
<td>2</td>
<td>5</td>
<td>1.06</td>
</tr>
<tr>
<td>Shlaer/Mellor</td>
<td>State Transition Diagram</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>.40</td>
</tr>
<tr>
<td>SSA</td>
<td>Entity Relationship Attribute Diagram</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>2.33</td>
<td>1.67</td>
<td>.50</td>
<td>1.44</td>
</tr>
<tr>
<td>SSA</td>
<td>Structured Systems Analysis</td>
<td>5</td>
<td>2</td>
<td>10</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>.50</td>
</tr>
<tr>
<td>Yourdon</td>
<td>Data Flow Diagram</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>1.67</td>
<td>2</td>
<td>1</td>
<td>.42</td>
</tr>
<tr>
<td>Yourdon</td>
<td>Entity Relationship Attribute Diagram</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>2.33</td>
<td>1.67</td>
<td>.50</td>
<td>1.44</td>
</tr>
<tr>
<td>Yourdon</td>
<td>State Transition Diagram</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>.40</td>
</tr>
<tr>
<td>Yourdon</td>
<td>Class Diagram</td>
<td>7</td>
<td>18</td>
<td>18</td>
<td>1.71</td>
<td>1.22</td>
<td>2.57</td>
<td>.10</td>
</tr>
<tr>
<td>UML</td>
<td>Use Case Diagrams</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>0.83</td>
<td>1</td>
<td>.17</td>
</tr>
<tr>
<td>UML</td>
<td>Activity Diagrams</td>
<td>8</td>
<td>5</td>
<td>6</td>
<td>0.75</td>
<td>0.20</td>
<td>0.63</td>
<td>.13</td>
</tr>
<tr>
<td>UML</td>
<td>Sequence Diagrams</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>0.67</td>
<td>6</td>
<td>0.17</td>
<td>.13</td>
</tr>
<tr>
<td>UML</td>
<td>Collaboration Diagrams</td>
<td>4</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>8</td>
<td>0.25</td>
<td>.14</td>
</tr>
<tr>
<td>UML</td>
<td>Object Diagrams</td>
<td>3</td>
<td>1</td>
<td>5</td>
<td>1.67</td>
<td>3</td>
<td>0.33</td>
<td>.33</td>
</tr>
<tr>
<td>UML</td>
<td>StateChart Diagrams</td>
<td>10</td>
<td>4</td>
<td>11</td>
<td>1</td>
<td>0.5</td>
<td>0.40</td>
<td>.09</td>
</tr>
<tr>
<td>UML</td>
<td>Component Diagrams</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>3.6</td>
<td>1.25</td>
<td>0.11</td>
<td>13.65</td>
</tr>
<tr>
<td>UML</td>
<td>Deployment Diagrams</td>
<td>5</td>
<td>7</td>
<td>5</td>
<td>1.14</td>
<td>1.40</td>
<td>0.20</td>
<td>9.95</td>
</tr>
</tbody>
</table>
From Table 3, it appears that the various diagrams in UML are not distinctly different from the techniques in other modeling methods. Although UML has more diagramming techniques when compared to other modeling methods, each of the diagramming techniques is no more complex than techniques found in other modeling methods. In fact, for most of the metrics, most UML diagrams rank in the middle (other than class diagrams).

Table 4 shows the average metrics values for each modeling method. This again shows that UML diagrams are not markedly different from other modeling methods. On average, diagramming techniques in UML are comparable to techniques in other modeling methods in terms of complexity.

Table 5 shows the complexity values at the method-level. UML stands out noticeably! It has the largest values in all method-level metrics except \( \overline{C} (M) \). The reason for this exception is that \( \overline{C} (M) \) shows the division of work between objects and relationships in the whole method. Since UML has the largest number of object types among all the modeling methods, it has the smallest \( \overline{C} (M) \).

As can be seen from Table 5, UML is indeed large and complex compared to other modeling methods. Compared to other modeling methods, UML consists of 3-19 times more object types, 2-17 times more relationship types, and 2-9 times more property types. As a result, UML is 2-11 times more complex than other modeling methods.

The complexity of UML can be daunting to novice users. In Siau (1999), the author pointed out the importance of considering human limitations when engineering methods. For example, it is well known that our working memory limitations place an upper bound of the number of chunks of information we can work with simultaneously. The UML notation includes nearly two hundred symbols (i.e., 57 object types, 53 relationship types, and 72 property types). Simply memorizing them would take hours or even days. As such, it is almost impossible for novice users to comprehend such a large and diverse set of constructs. If end-users cannot understand the constructs in the techniques, the communication process between analysts and end-users during requirements analysis will be severely impaired. This is consistent with the concern of Halpin and Bloesch (1999) who pointed out that UML might not be suitable for developing and validating a conceptual model with domain experts.

### Table 4: Average complexity values for UML and other modeling methods

<table>
<thead>
<tr>
<th>Method</th>
<th>( n (O_2) )</th>
<th>( n (R_2) )</th>
<th>( n (P_2) )</th>
<th>( \overline{PO} (M_1) )</th>
<th>( \overline{PR} (M_1) )</th>
<th>( \overline{RO} (M_1) )</th>
<th>( \overline{C} (M_1) )</th>
<th>( \overline{C}' (M_1) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUSION</td>
<td>3</td>
<td>2.50</td>
<td>7.50</td>
<td>2.5</td>
<td>1.63</td>
<td>1.88</td>
<td>.65</td>
<td>8.58</td>
</tr>
<tr>
<td>JSP</td>
<td>1.50</td>
<td>1.50</td>
<td>4</td>
<td>2.75</td>
<td>1.25</td>
<td>2.38</td>
<td>4.53</td>
<td></td>
</tr>
<tr>
<td>MOSES</td>
<td>2.5</td>
<td>4.50</td>
<td>8.00</td>
<td>3.75</td>
<td>3.50</td>
<td>2.94</td>
<td>.64</td>
<td>9.79</td>
</tr>
<tr>
<td>OMT</td>
<td>4</td>
<td>6.33</td>
<td>8.67</td>
<td>1.72</td>
<td>2.80</td>
<td>2.67</td>
<td>.53</td>
<td>11.97</td>
</tr>
<tr>
<td>OOAD</td>
<td>2.33</td>
<td>2.67</td>
<td>5.33</td>
<td>2.33</td>
<td>2.33</td>
<td>.83</td>
<td>.75</td>
<td>6.52</td>
</tr>
<tr>
<td>OODA</td>
<td>3.80</td>
<td>3.60</td>
<td>6.40</td>
<td>3.53</td>
<td>2.80</td>
<td>2.49</td>
<td>.72</td>
<td>9.23</td>
</tr>
<tr>
<td>OSA</td>
<td>3</td>
<td>4.33</td>
<td>7</td>
<td>1.93</td>
<td>2.73</td>
<td>1.69</td>
<td>.42</td>
<td>8.78</td>
</tr>
<tr>
<td>RTSA</td>
<td>3</td>
<td>2.75</td>
<td>4.75</td>
<td>2.13</td>
<td>1.97</td>
<td>1.88</td>
<td>.70</td>
<td>6.66</td>
</tr>
<tr>
<td>Shlaer/Mellor</td>
<td>3.33</td>
<td>3</td>
<td>6.67</td>
<td>2.33</td>
<td>3.85</td>
<td>3.58</td>
<td>.59</td>
<td>7.80</td>
</tr>
<tr>
<td>SSA</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>2.67</td>
<td>1.83</td>
<td>1.25</td>
<td>.97</td>
<td>8.82</td>
</tr>
<tr>
<td>Yourdon</td>
<td>2.5</td>
<td>1.5</td>
<td>5.25</td>
<td>2</td>
<td>1.42</td>
<td>1.88</td>
<td>.73</td>
<td>6.04</td>
</tr>
<tr>
<td>UML</td>
<td>6.33</td>
<td>5.89</td>
<td>8.00</td>
<td>1.09</td>
<td>2.72</td>
<td>0.89</td>
<td>0.16</td>
<td>12.32</td>
</tr>
</tbody>
</table>
The large number of constructs in UML is partly due to its diverse diagramming techniques. Constructs (e.g., object types, relationship types, property types) in use case diagrams are very different from constructs in class diagrams. Class diagrams are also very different from activity diagrams or state-chart diagrams. This diversity causes problems. First, there are more constructs to learn. Second, the knowledge and experience gained in using one diagramming technique cannot be transferred to another diagramming technique in UML. Sometimes, a construct can have different semantics in different diagrams. For example, a transition in an activity diagram represents a flow of control whereas a transition in a state-chart diagram symbolizes a change of state. This can be very confusing to novice users as both are transitions. Many constructs in UML need refinement or clarification.

With a large number of diagramming techniques and a huge number of constructs, a systematic way of guiding the users through the systems development process is required. Unfortunately, this is another shortcoming of UML – it lacks a comprehensive and systematic way of navigating through the technique- and construct-filled jungle. There are different views about how to use UML to develop software systems. The Rational Unified Process is an attempt to provide a standard. However, the Rational Unified Process, in a nutshell, consists of four simple ideas – use-case driven, architecture-centric, iterative, and incremental. It is doubtful that these four ideas will form the silver bullet to resolve the complexity of UML and to provide a roadmap for using UML in systems development.

Despite the problems with UML, having a standard modeling language is definitely a step in the right direction. At the very least, the research community can now focus its attention on improving one modeling language instead of a few dozen OO languages.

**LIMITATIONS AND CONCLUSIONS**

In this study, we apply the complexity metrics proposed by Rossi and Brinkkemper (1994) to UML. We also compare the UML complexity values with the values of other modeling techniques and methods. The research produces some interesting results. For the diagramming techniques in UML, each diagram of UML, taken individually, is not distinctly

---

**Table 5: Complexity values (method-level metrics)**

<table>
<thead>
<tr>
<th>Method</th>
<th>n (O_t)</th>
<th>n (R_t)</th>
<th>n (P_t)</th>
<th>(\bar{C} ) (M)</th>
<th>C'(M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSP</td>
<td>3</td>
<td>3</td>
<td>8</td>
<td>2.17</td>
<td>9.06</td>
</tr>
<tr>
<td>FUSION</td>
<td>6</td>
<td>5</td>
<td>15</td>
<td>.61</td>
<td>16.91</td>
</tr>
<tr>
<td>SSA</td>
<td>6</td>
<td>4</td>
<td>16</td>
<td>.97</td>
<td>17.55</td>
</tr>
<tr>
<td>MOSES</td>
<td>5</td>
<td>9</td>
<td>16</td>
<td>.66</td>
<td>19.03</td>
</tr>
<tr>
<td>OOAD</td>
<td>7</td>
<td>8</td>
<td>16</td>
<td>.68</td>
<td>19.21</td>
</tr>
<tr>
<td>Shlaer/Mellor</td>
<td>7</td>
<td>9</td>
<td>20</td>
<td>.56</td>
<td>23.02</td>
</tr>
<tr>
<td>OSA</td>
<td>9</td>
<td>13</td>
<td>21</td>
<td>.56</td>
<td>26.29</td>
</tr>
<tr>
<td>OMT</td>
<td>12</td>
<td>19</td>
<td>26</td>
<td>.59</td>
<td>34.37</td>
</tr>
<tr>
<td>Yourdon</td>
<td>10</td>
<td>6</td>
<td>21</td>
<td>.75</td>
<td>24.02</td>
</tr>
<tr>
<td>RTSA</td>
<td>12</td>
<td>11</td>
<td>19</td>
<td>.64</td>
<td>25.02</td>
</tr>
<tr>
<td>OODA</td>
<td>19</td>
<td>18</td>
<td>32</td>
<td>1.05</td>
<td>41.34</td>
</tr>
<tr>
<td><strong>UML</strong></td>
<td><strong>57</strong></td>
<td><strong>53</strong></td>
<td><strong>72</strong></td>
<td><strong>0.14</strong></td>
<td><strong>106</strong></td>
</tr>
</tbody>
</table>
more complex than other modeling techniques. However, UML, as a whole, is very much more complex than other modeling methods. This indicates that future work on UML needs to focus on reducing the complexity in using the numerous techniques in UML.

This research, like others, has its limitations. The Object, Property, Relationship, Role (OPRR) method modeling language used in this study models mainly the static aspects of the techniques. OPRR is not capable of dealing appropriately with interconnected techniques. Also, the definitions of \{O, P, R, X, r, p\} are subjected to different interpretations. Since we were going to compare UML complexity values to other modeling methods derived by Rossi and Brinkkemper (1994), we tried to follow their derivation approach as much as possible. The use of metrics has its own shortcomings. One of the pitfalls of metrics is that it produces only numbers, not insight. Combining complexity metrics approach with empirical studies will enable us to understand the strengths and weaknesses of UML more fully.

ACKNOWLEDGMENT

This research is partly supported by a research funding from the University of Nebraska-Lincoln (LWT/06-185-92501).

ENDNOTE

1 Some modeling methods, Demeter, Express, and Goldkuhl, have only one technique. We exclude these three modeling methods from the computation.

REFERENCES


Since its adoption by the Object Management Group as a language for object-oriented analysis and design, the Unified Modeling Language (UML) has become widely used for designing object-oriented code. However, UML has had only minimal adoption among practitioners for the purposes of information analysis and database design. One main reason for this is that the class diagrams used in UML for data modeling provide only weak, and awkward, support for the kinds of business rules found in data-intensive applications. Moreover, UML’s graphical language does not lend itself readily to verbalization and multiple instantiation for validating data models with domain experts. These defects can be remedied by using a fact-oriented approach for information analysis, from which UML class diagrams may be derived. Object-Role Modeling (ORM) is currently the most popular fact-oriented modeling approach. This chapter examines the relative strengths and weaknesses of UML and ORM for conceptual data modeling, and indicates how models in one notation can be translated into the other.

INTRODUCTION

The Unified Modeling Language (UML) was adopted by the Object Management Group (OMG) in 1997 as a language for object-oriented (OO) analysis and design. At the time of writing, the latest approved specification for UML is version 1.4 (OMG, 2001), with various proposals for version 2.0 under consideration. Our discussion of UML is confined to version 1.4. For a simple introduction to UML, see Fowler (1997). Detailed treatments of UML are provided by Booch, Rumbaugh, and Jacobson (1999), and Rumbaugh, Jacobson, and Booch (1999). In-depth discussions of UML for database design are given by Muller (1999) and, with a slightly different notation, by Blaha and Premerlani (1998).
UML can be used for this task also, since a class diagram can be viewed as an extended Entity-Relationship (ER) notation, assuming it is annotated with user-defined constraints for database constructs (e.g., key declarations).

The UML notation includes hundreds of symbols, from which various diagrams may be constructed to model different perspectives of an application (e.g., use-case diagrams, class diagrams, object diagrams, state-charts, activity diagrams, sequence diagrams, collaboration diagrams, component diagrams and deployment diagrams). This chapter focuses on conceptual data modeling, so considers only the static structure (class and object) diagrams. Class diagrams are used for the data model, and object diagrams provide a limited means to discuss data populations.

UML is suitable for OO code design, covering both data and behavioral aspects, and allowing OO-implementation details to be directly expressed (e.g., attribute visibility and directional navigation across associations). UML may be also used for analysis by ignoring its implementation features. Although not widely used for modeling database applications, analysis versions of UML class diagrams effectively provide an Entity-Relationship (ER) notation, assuming they are annotated with additional user-defined constraints (e.g., uniqueness declarations for attributes).

In practice, however, UML static structure diagrams are often unsuitable for information analysis, since they can make it awkward to capture and validate data concepts and business rules with domain experts, and to cater for structural changes in the application domain. These problems can be remedied by using a fact-oriented approach for information analysis, where communication takes place in simple sentences; each sentence type can easily be populated with multiple instances, and attributes are eschewed in the base model. At design time, a fact-oriented model can be used to derive a UML class model or a logical database model.

Object Role Modeling (ORM) is the main exemplar of the fact-oriented approach, and though less popular than UML, is used productively in many countries and is supported by CASE tools from companies such as Ascaris and Microsoft. ORM harmonizes well with UML, since both approaches provide direct support for association roles, n-ary associations and objectified associations. ORM pictures the world simply in terms of objects (entities or values) that play roles (parts in relationships). For example, you are now playing the role of reading, and this chapter is playing the role of being read. Overviews of ORM may be found in Halpin (1998a, 1998b) and a detailed treatment in Halpin (2001a).

The rest of this chapter is structured as follows. The next section provides a comparative overview of UML class diagrams and ORM, based on linguistic design criteria. These design principles are then used to examine the relative strengths and weaknesses of UML and ORM for information analysis, focusing first on data structures, and then moving on to constraints, outlining how models in one notation can be translated into the other. An example of schema transformation and optimization is then used to underscore the importance of constraint expressibility. We then briefly compare textual language support for constraints, derivation rules and queries in the two approaches. The conclusion summarizes the main points and identifies topics for future research.

CONCEPTUAL MODELING LANGUAGE CRITERIA

A modeling method comprises both a language and a procedure to guide modelers in using the language to construct models. A language has associated syntax (marks), semantics
(meaning) and pragmatics (use). Written languages may be graphical (diagrams) and/or textual. The terms “abstract syntax” and “concrete syntax” are sometimes used to distinguish underlying concepts (e.g., class) from their representation (e.g., named rectangle). Conceptual modeling portrays the application domain at a high level, using terms and concepts familiar to the application users, ignoring logical and physical level aspects (e.g., the underlying database or programming structures used for implementation) and external level aspects (e.g., screen forms used for data entry).

Drawing upon lessons from linguistics and semiotics, the ORM language was designed from the ground up to meet the following design criteria: expressibility; clarity; learnability (hence, orthogonality, parsimony and convenience); semantic stability (minimize the impact of change); semantic relevance (scope views to just the currently relevant task); validation mechanisms; abstraction mechanisms; and formal foundation.

The expressibility of a language is a measure of what it can be used to say. Ideally, a conceptual language should be able to completely model all details about the application domain that are conceptually relevant. This is called the 100% Principle (ISO, 1982). ORM is a method for modeling and querying an information system at the conceptual level, and for mapping between conceptual and logical levels. Although various ORM extensions have been proposed for object-orientation and dynamic modeling (e.g., Barros, ter Hofstede, & Proper, 1997; De Troyer & Meersman, 1995; ter Hofstede & van der Weide, 1994), the focus of ORM is on data modeling, since the data perspective is more stable and it provides a formal foundation on which operations can be defined. In this sense, UML is generally more expressive than standard ORM, since its use-case, behavior and implementation diagrams model aspects beyond static structures. An evaluation of such additional modeling capabilities of UML and ORM extensions is beyond the scope of this chapter. We show later that ORM diagrams are graphically more expressive than UML class diagrams.

The clarity of a language is a measure of how easy it is to understand and use. To begin with, the language should be unambiguous. Ideally, the meaning of diagrams or textual expressions in the language should be intuitively obvious. At a minimum, the language concepts and notations should be easily learned and remembered. Semantic stability is a measure of how well models or queries expressed in the language retain their original intent in the face of changes to the application. The more changes one makes to a model or query to cope with an application change, the less stable it is. Semantic relevance requires that only conceptually relevant details need be modeled. Any aspect irrelevant to the meaning (e.g., implementation choices, machine efficiency) should be avoided. This is called the conceptualization principle (ISO, 1982).

Validation mechanisms are ways in which domain experts can check whether the model matches the application. For example, static features of a model may be checked by verbalization and multiple instantiation, and dynamic features may be checked by simulation. Abstraction mechanisms allow unwanted details to be removed from immediate consideration. This is very important with large models (e.g., wall-size schema diagrams). ORM diagrams tend to be more detailed and larger than corresponding UML models, so abstraction mechanisms are often used. For example, a global schema may be modularized into various scopes or views based on span or perspective (e.g., a single page of a data model, or a single page of an activity model). Successive refinement may be used to decompose higher-level views into more detailed views. Tools can provide additional support (e.g., feature toggles, layering, and object zoom). Such mechanisms can be used to hide and show just that part of the model relevant to a user’s immediate needs (Campbell, Halpin, & Proper, 1996; Halpin, 2001a). With minor variations, these
techniques can be applied to both ORM and UML. ORM also includes an attribute abstraction procedure to generate an ER diagram as a view.

A formal foundation ensures unambiguity and executability (e.g., to automate the storage, verification, transformation and simulation of models), and allows formal proofs of equivalence and implication between alternative models (Halpin & Proper, 1995b). Although ORM’s richer, graphical constraint notation provides a more complete diagrammatic treatment of schema transformations, use of textual constraint languages can partly offset this advantage. For their data modeling constructs, both UML and ORM have an adequate formal foundation. Since ORM and UML are roughly comparable with regard to abstraction mechanisms and formal foundations, our following evaluation focuses on the criteria of expressibility, clarity, stability, relevance and validation.

Language design often involves a number of trade-offs between competing criteria. One well-known trade-off is that between expressibility and tractability (Levesque, 1984): the more expressive a language is, the harder it is to make it efficiently executable. Another trade-off is between parsimony and convenience: although ceteris paribus, the fewer concepts the better (cf. Occam’s razor), restricting ourselves to the minimum number of concepts may sometimes be too inconvenient. For example, two-valued propositional calculus allows for four monadic and 16 dyadic logical operators. All 20 of these operators can be expressed in terms of a single logical operator (e.g., nand, nor), but while useful in building electronic components, it is simply too inconvenient for direct human communication. For example, “not \( p \)” is far more convenient than “\( p \) nand \( p \).” In practice, we use several operators (e.g., not, and, or, if-then) since the convenience of using them directly far outweighs the parsimonious benefits of having to learn only one operator such as nand. When it comes to proving meta-theorems about a given logic, it is often convenient to adopt a somewhat parsimonious stance regarding the base constructs (e.g., treat “not” and “or” as the only primitive logical operators), while introducing other constructs as derived (e.g., define “if \( p \) then \( q \)” as “not \( p \) or \( q \)”). Similar considerations apply to modeling languages.

One basic question relevant to the parsimony-convenience trade-off is whether to use the attribute concept as a base-modeling construct. The main structural difference between ORM and UML is that ORM eschews attributes in its base models whereas UML models typically make extensive use of attributes. Implicitly, attributes may be associated with roles in a relationship. For example, Employee.birthdate is modeled in ORM as the second role of the fact type: Employee was born on Date. Traditional ER supports single-valued attributes, while UML supports both single-valued and multi-valued attributes. We argue later that ORM’s parsimonious avoidance of attributes in its base models is convenient and advantageous for validating and evolving conceptual models.

### DATA STRUCTURES

Table 1 summarizes the main correspondences between conceptual data constructs in ORM and UML. An uncommented “—” indicates no predefined support for the corresponding concept, and “†” indicates incomplete support. This comparison indicates that ORM’s built-in symbols provide greater expressive power for capturing conceptual constraints in data models. In this section we consider the top section of the table (data structures).

In UML and ORM, objects and data values are both instances. Each object is a member of at least one type, known as class in UML and an object type in ORM. ORM classifies objects into entities (UML objects) and values (UML data values—constants such as
Table 1: Comparison of the main conceptual data constructs in ORM and UML

<table>
<thead>
<tr>
<th>ORM</th>
<th>UML</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data structures:</strong></td>
<td><strong>Data structures:</strong></td>
</tr>
<tr>
<td>object type: entity type; value type</td>
<td>object class</td>
</tr>
<tr>
<td>— { use association }</td>
<td>data type</td>
</tr>
<tr>
<td>unary association</td>
<td>attribute</td>
</tr>
<tr>
<td>2-ary association</td>
<td>— { use Boolean attribute }</td>
</tr>
<tr>
<td>objectified association (nesting)</td>
<td>2-ary association</td>
</tr>
<tr>
<td>co-reference</td>
<td>association class</td>
</tr>
<tr>
<td>Predefined Constraints:</td>
<td>qualified association †</td>
</tr>
<tr>
<td>internal uniqueness</td>
<td>Predefined Constraints:</td>
</tr>
<tr>
<td>external uniqueness</td>
<td>multiplicity of ..1 †</td>
</tr>
<tr>
<td>simple mandatory role</td>
<td>— { use qualified association } †</td>
</tr>
<tr>
<td>disjunctive mandatory role</td>
<td>multiplicity of 1..* †</td>
</tr>
<tr>
<td>frequency: internal; external value</td>
<td>—</td>
</tr>
<tr>
<td>subset and equality</td>
<td>multiplicity †; —</td>
</tr>
<tr>
<td>exclusion</td>
<td>enumeration, and textual subset †</td>
</tr>
<tr>
<td>subtype link and definition</td>
<td>xor †</td>
</tr>
<tr>
<td>ring constraints</td>
<td>subclass discriminator, etc. †</td>
</tr>
<tr>
<td>join constraints</td>
<td>—</td>
</tr>
<tr>
<td>object cardinality</td>
<td>—</td>
</tr>
<tr>
<td>— { use uniqueness and ring } †</td>
<td>class multiplicity</td>
</tr>
<tr>
<td><strong>User-defined textual constraints</strong></td>
<td>User-defined textual constraints</td>
</tr>
<tr>
<td>† = incomplete coverage of corresponding concept</td>
<td></td>
</tr>
</tbody>
</table>

character strings or numbers). In UML, entities are identified by oids, but in ORM they must have a reference scheme for human communication (e.g., employees might be referenced by social security numbers). UML classes must have a name, and may also have attributes, operations (implemented as methods) and play roles. ORM object types must have a name and play roles. Since our focus is on the data perspective, we avoid any detailed discussion of operations, except to note that some of these may be handled in ORM as derived relationship types. A relationship instance in ORM is called a link in UML (e.g., Employee 101 works for Company “Microsoft”). A relationship type in ORM is called an association in UML (e.g., Employee works for Company). Object types in ORM are depicted as named ellipses, and simple reference schemes may be abbreviated in parentheses below the type name. Classes in UML are depicted as named rectangles to which attributes and operations may be added.

Apart from object types, the only data structure in ORM is the relationship type. In particular, attributes are not used at all in base ORM. This is a fundamental difference between ORM and UML (and ER for that matter). Wherever an attribute is used in UML, ORM uses a relationship instead. This omission was deliberately designed to avoid fuzzy and unstable distinctions about whether a feature should be modeled as an attribute or an association (Falkenberg, 1976). Although this advantage is enjoyed by some other semantic modeling approaches, such as OSM (Embley, 1998), the price paid is that attribute-free diagrams usually consume more space. This disadvantage is easily outweighed by the many advantages of the attribute-free approach.
Firstly, attribute-free models and queries are more stable, because they are free of changes caused by attributes evolving into other constructs (e.g., associations), or vice versa. For example, suppose we model car as an attribute of Employee. If we later decide that employees may drive many cars, we need to replace this attribute by an association (Employee drives Car) or a multi-valued attribute (cars). If we decide to record data about cars (e.g., carmodel) a Car class must be introduced. If attributes are replaced, queries and constraints based on them also need replacing. Since we can’t be sure about our future modeling needs, use of attributes in the base model decreases semantic stability.

An ORM model is essentially a connected network of object types and relationship types. The object types are the semantic domains that glue things together, and are always visible. This connectedness reveals relevant detail and enables ORM models to be queried directly, traversing through object types to perform conceptual joins (Bloesch & Halpin, 1997; Halpin, 2001a). In addition, attribute-free models are easy to populate with multiple instances, facilitate verbalization (in sentences), are simpler and more uniform, avoid arbitrary modeling decisions, and facilitate constraint specification (see later).

Attributes, however, have two advantages: they often lead to a more compact diagram, and they can simplify arithmetic derivation rules. For this reason, ORM includes algorithms for dynamically generating attribute-based diagrams as views (Campbell et al., 1996; Halpin, 2001a). These algorithms assign different levels of importance to object types depending on their current roles and constraints, redisplaying minor fact types as attributes of the major object types. Elementary facts are the fundamental conceptual units of information and are uniformly represented as relationships. How they are grouped into structures is not a conceptual issue.

ORM allows relationships of any arity (number of roles), each with at least one reading or predicate name. An n-ary relationship may have up to n readings (one starting at each role), to more naturally verbalize constraints and navigation paths in any direction. ORM also allows role names to be added. A predicate is an elementary sentence with holes in it for object terms. These object holes may appear at any position in the predicate (mixfix notation), and are denoted by an ellipsis “…” if the predicate is not infix-binary. In support of our clarity criterion, mixfix notation enables natural verbalization of sentences in any language (e.g., in Japanese, verbs come at the end of sentences). ORM includes procedures to assist in the creation and transformation of models. A key step in its design procedure is the verbalization of relevant information examples, such as sample reports expected from the system. These “data use cases” are in the spirit of UML use cases, except the focus is on the underlying data.

ORM sentence types (and constraints) may be specified either textually or graphically. Both are formal, and can be automatically transformed into the other. In an ORM diagram, roles appear as boxes, connected by a line to their object type. A predicate appears as a named, contiguous sequence of role boxes. Since these boxes are set out in a line, fact types may be conveniently populated with tables holding multiple fact instances, one column for each role. This allows all fact types and constraints to be validated by verbalization as well as sample populations. Communication between modeler and domain expert takes place in a familiar language, backed up by population checks.

UML uses Boolean attributes instead of unary relationships, but allows relationships of all other arities. Each association has at most one name. Binary associations are depicted as lines between classes, with higher arity associations depicted as a diamond connected by lines to the classes. Roles are simply association ends, but may optionally be named. Verbalization into sentences is possible only for infix binaries, and then only by naming the
association with a predicate name (e.g., “employs”) and using an optional marker “{ }” to denote the direction. Since roles for ternaries and higher arity associations are not on the same line, directional verbalization is ruled out. This non-linear layout also makes it impractical to conveniently populate associations with multiple instances. Add to this the impracticality of displaying multiple populations of attributes, and it is clear that class diagrams do not facilitate population checks (see later). UML does provide *object diagrams* for instantiation, but these are convenient only for populating with a *single* instance. Hence, “the use of object diagrams is fairly limited” (OMG, 2001). We conclude that ORM surpasses UML on the validation mechanism criterion.

Both UML and ORM allow associations to be objectified as first-class object types, called *association classes* in UML and *nested-object types* in ORM. UML requires the same name to be used for the association and the association class, impeding natural verbalization, in contrast to ORM nesting based on linguistic nominalization (a verb phrase is objectified by a noun phrase). UML allows objectification of n:1 associations. Currently ORM forbids this except for 1:1 cases, since attached roles are typically best viewed as played by the object type on the “many” end of the association. However, ORM could be relaxed to allow this, with its mapping algorithms adding a pre-processing step to re-attach roles and adjust constraints internally. In spite of identifying association classes with their underlying association, UML displays them separately, making the connection by a dashed line. In contrast, ORM intuitively envelops the association with an object-type frame (see Figure 1).

**CONSTRAINTS**

In Figure 1, the UML diagram includes *multiplicity constraints* on the association roles. The “1..*” indicates that each paper is written by one or more persons. In ORM this is captured as a *mandatory role* constraint, represented graphically by a black dot. The ORM tool within Microsoft Visual Studio.NET Enterprise Architect allows this constraint to be entered graphically, or by answering a multiplicity question, or by induction from a sample population, and can automatically verbalize the constraint. If the inverse predicate “is written by” has been entered, the constraint is verbalized as “each Paper is written by at least one Person.”

In UML, the “*” on the right-hand role indicates that each person wrote zero or more papers. In ORM, the lack of a mandatory role constraint on the left role indicates it is optional (a person might write no papers), and the arrow-tipped line spanning the predicate is a *uniqueness constraint* indicating the association is many:many (when the fact table is

---

*Figure 1: Writing is depicted as an objectified association in UML and ORM*
populated, each whole row is unique). A uniqueness constraint on a single role means that entries in that column of the associated fact table must be unique. Figure 2 shows equivalent constraint notations for binary associations, read from left to right. The third case (m:n optional) is the weakest constraint pattern. Though not shown here, 1:n cases are the reverse of the n:1 cases, and 1:1 cases combine the n:1 and 1:n cases.

An internal constraint applies to roles in a single association. For an n-ary association, each internal uniqueness constraint must span at least n - 1 roles. Unlike many ER notations, UML and ORM can express all possible internal uniqueness constraints. For example, Figure 3 is a UML diagram that includes a ternary association (Usage) in which both Room-Time and Time-Activity pairs are unique. This schema also includes a textual constraint written informally as a note attached by dashed lines to the three associations involved in the constraint. A textual constraint is needed here since UML provides no graphical way of capturing the constraint. UML does allow the constraint to be captured formally in OCL (Object Constraint Language), but the syntax of OCL is too mathematical for it to be used to validate rules with subject matter experts who typically have little technical background.

An ORM depiction of the same situation is shown in Figure 4, along with sample populations. The dotted arrow with a circled “⊆” symbol is a join-subset constraint (see later) that formally captures the textual constraint shown in the UML version. ORM visually distinguishes value types from entity types by using dashed-ellipses (e.g., FacilityName). Note how useful populations are for checking the constraints. For example, the data clearly illustrate the uniqueness constraints. If Time-Activity is not really unique, this can be shown by adding a counterexample. If UML object diagrams were used to show these populations, the constraint patterns would be far harder to see, since each instance of each class and association would be shown separately (checking even the simple 1:1 constraint between facility codes and names requires inspecting three facility objects). Although ORM-like fact tables could be used with UML associations, relating association-roles to the relevant fact columns is often awkward.

Multiplicity constraints in UML may specify any range (e.g., 1, 3..7), but each is applied to a single role. For n-ary associations, the multiplicity indicates how many instances may be associated with each instantiation of the other n-1 classes. As a consequence, UML’s concept of multiplicity does not generalize fully for n-aries, since it cannot express a

Figure 2: Some equivalent representations in UML and ORM
mandatory or minimum, positive frequency constraint that applies to fewer than n-1 roles (Halpin, 2001b). ORM allows the same ranges, but partitions the multiplicity concept into the orthogonal notions of mandatory role and frequency constraints. This useful separation localizes global impact to just the mandatory role constraint (e.g., every population instance of an object type \( A \) must play every mandatory role of \( A \)). Because of its non-local impact, modelers should omit this constraint unless it is really needed (as in Figure 1).
ORM frequency constraints apply only to populations of the constrained roles (e.g., if an instance plays that role, it does so the specified number of times) and hence have only local impact. Frequency constraints in ORM are depicted as number ranges next to the relevant roles. For example, to add the constraint that papers must be reviewed by at least two people, we add the mark “³ 2” beside the first role of Paper is reviewed by Person. Uniqueness constraints are just frequency constraints with a frequency of 1, but are given a special notation because of their importance. Because ORM’s mandatory and frequency constraints are applied separately and directly to the role populations, these constraints generalize properly to n-ary associations.

Attribute multiplicity constraints in UML are placed in square brackets after the attribute name (e.g., Figure 1). If no such constraint is specified, the attribute is assumed to be single-valued and mandatory. Multi-valued attributes are arguably an implementation concern. Mandatory role constraints in ORM may apply to a disjunction of roles, e.g., each academic is either tenured or contracted till some date (inclusive-or). UML cannot express disjunctive mandatory role constraints graphically. Perhaps influenced by oids, UML omits a standard notation for attribute uniqueness constraints (candidate keys). It suggests that boldface might be used for this (or other purposes) as a tool extension. Another alternative is to annotate unique attributes with comments (e.g., {P} for primary reference, {U1}, etc.).

Frequency and uniqueness constraints in ORM may apply to a sequence of any number of roles from any number of predicates. This goes far beyond the graphical expressibility of UML. For example, consider the m:n fact type: Account(nr) is used by Client(nr) and the n:1 fact type Account(nr) has AccountType(code), and add the uniqueness constraint that for any given account type, each client has at most one account (Halpin, 2001a, p. 687). ORM constraints that span different predicates are called external constraints. Few of these can be graphically expressed in UML. For example, subset and equality constraints in ORM may be expressed between two compatible role-sequences, where each sequence is formed by projection from possibly many connected predicates. Figure 5 includes two simple examples: students have second names only if they have first names, and may pass tests in a course only if they enrolled in that course.

UML can diagram only basic subset constraints between binary associations, e.g., in Figure 6, a person who chairs a committee must be a member of it. UML cannot diagram subset constraints between parts of associations (as in Figure 5), and this inability to project on the relevant roles invites modeling errors (e.g., Blaha & Premerlani, 1968, 68).

However, as Figure 7 illustrates, it is possible to capture ORM subset constraints in UML by adding a textual constraint or sometimes by applying a model transformation (e.g., remodel a ternary using an association class).

Figure 5: Subset constraints in ORM
The dotted arrow with the circled “⊆” in Figure 4 expressed the following join-subset constraint: if a Room at a Time is used for an Activity that requires a Facility then that Room provides that Facility. If we record the title and sex of each employee, we should also include a populated relationship type indicating which titles determine which sex (e.g., “Mrs,” “Miss,” “Ms,” and “Lady” apply only to the female sex). In ORM, this is easily visualized as a join-subset constraint (see Figure 8) and verbalized (if Person has a Title that determines Sex, then Person is of Sex.). If we instead model title and sex as attributes, this rule cannot be diagrammed. In ORM, a value constraint restricts the population of a value type, and is indicated in braces. In UML, such constraints may be declared as enumerations or as textual constraints (e.g., see the Sexcode constraint in Figure 8).

ORM allows exclusion constraints over a set of compatible role-sequences, by connecting “⊗” by dotted lines to the relevant role-sequences. For example, given the associations Person wrote Paper and Person reviewed Paper, consider the two exclusion constraints: no person wrote and reviewed; no person wrote and reviewed the same paper. ORM distinguishes these cases by noting the precise arguments of the constraint. If a set of roles is both exclusive and disjunctively mandatory, ORM orthogonally combines an exclusion and disjunctive-mandatory constraint. This “exclusive or” case is captured in UML by connecting “{xor}” to the relevant associations by dashed lines to indicate exactly which one is played. UML has no graphic notation for simple exclusion between roles, role-sequences, attributes, or between attributes and associations. ORM includes equality constraints (circled “=” as an abbreviation for subset constraints in both directions.

UML uses qualified associations in many cases where ORM uses an external uniqueness constraint for co-referencing. Figure 9 is based on an example from the UML specification (OMG, 2001), along with the ORM counterpart. Qualified associations are shown as
Figure 8: ORM makes it easy to capture the constraint between title and sex

(a) Employee

| empNr (P) | title | sex: Sexcode |

«enumeration» Sexcode

m  f

(b) Employee (empNr)

Title determines

Sex (code)

{M,'F'}

Figure 9: Qualified association in UML, and co-referenced object type in ORM

Bank

| accountNr |

0..1

Client

ORM

Account

| is in |

| is used by |

| has |

| is in |

| is used by |

| has |

Bank (name)

AccountNr

Client (nr)

named, smaller rectangles attached to a class. ORM uses a circled “u” to denote an external uniqueness constraint (the bank name and account number uniquely define the account). The UML notation is less clear, and less adaptable. For example, if we now want to record something about the account (e.g., its balance) we need to introduce an Account class, and the connection to accountNr is unclear. The problem can be solved in UML by using an association class instead, though this is not always natural.

Both UML and ORM support subtyping, including multiple inheritance. Both show subtypes outside, connected by arrows to their supertype(s), and both allow declaration of constraints between subtypes such as exclusion and totality. UML provides only weak support for defining subtypes: a discriminator label may be attached to subtype arrows to indicate the basis for the classification (e.g., a “sex” discriminator might specialize Man and Woman from Person). This does not guarantee that instances populating these subtypes have the correct values for a sex attribute that might apply to Person. Moreover, more complicated subtype definitions are sometimes required. Finally, subtype constraints such as exclusion and totality are typically implied by subtype definitions in conjunction with existing constraints on the supertypes; these implications are captured in ORM but are ignored in
UML, leading to the possibility of inconsistent UML models. For further discussion on these issues see Halpin (2001a) and Halpin and Proper (1995a).

ORM includes other graphic constraints with no counterpart in UML. For example, *ring constraints* such as irreflexivity, asymmetry, intransitivity and acyclicity, may be specified over a pair of roles played by the same object type (e.g., Person is parent of Person is acyclic and deontically intransitive). Such constraints can be specified as comments in UML. UML treats aggregation as a special kind of whole/part association, attaching a small diamond to the role at the “whole” end of the association. In ORM, this is shown as an m:n association. UML treats composition as a stronger form of aggregation in which each part belongs to at most one whole (in ORM, the “contains” predicate becomes 1:n). Whole and Part are not necessarily disjoint types, so ring constraints may apply (e.g., Part contains Part). UML aggregation also has behavioral semantics concerned with implementation at the code level (e.g., copy and access semantics), but these are not conceptual issues and have no counterpart in ORM.

UML allows *collection types* to be specified as annotations. For example, to record the order in which authors are listed for any given paper, the UML diagram in Figure 1 may have its author role annotated by “{ordered}.” This denotes an ordered set (sequence with unique members). ORM has two approaches to handle this. One way keeps base predicates elementary, annotating them with the appropriate constructor as an implementation instruction. In this example, we use the ternary fact type Person wrote Paper in Position, place uniqueness constraints over Person-Paper and Paper-Position, and annotate the predicate with “{seq}” to indicate mapping the positional information as a unique sequence. Sets, sequences and bags may be treated similarly. This method is recommended, partly because elementarity allows individual instantiation and simplifies the semantics. The other way allows complex object types in the base model by applying constructors directly to them, e.g., ter Hofstede and van der Weide (1994).

Both ORM and UML include *object cardinality constraints* for limiting the cardinality of a type’s population. For example, the number of senators may be capped at 50 in ORM by writing “#£50” beside the object type Senator, and in UML by writing “50” at the top right-hand corner of the Senator class. In UML, attributes may be assigned default values, and restrictions placed on their changeability (e.g., “frozen”). Although not supported in ORM, such features could easily be added to ORM as role properties. More sophisticated proposals to extend ORM to handle default information have also been made (Halpin & Vermier, 1997).

ORM includes various sub-conceptual notations that allow a pure conceptual model to be annotated with implementation detail (e.g., indexes, subtype mapping choices, constructors). UML includes a vast set of such annotations for class diagrams, providing intricate detail for implementation in object-oriented code (e.g., navigation directions across associations, attribute visibility [public, protected, private], etc.). These are irrelevant to conceptual modeling and are hence ignored in this chapter.

**SCHEMA TRANSFORMATION AND TEXTUAL LANGUAGES**

Although *schema transformation and equivalence* has been investigated in UML (e.g., Blaha & Premerlani, 1998), the formalization of these notions has been taken much further in ORM (e.g., Halpin & Proper, 1995b; Halpin, 2001a). One application of this theory is
conceptual schema optimization, where a conceptual model is preprocessed to an equivalent schema that maps to a more efficient implementation. For example, consider the ORM schema in Figure 10. The circled dot denotes an inclusive-or constraint. If input to Rmap, an ORM-to-Relational mapping algorithm (Halpin, 2001a), this schema maps to a relational schema containing seven normalized table schemes.

Applying a default optimization algorithm (Halpin, 2001a), the conceptual schema in Figure 10 may be transformed into the equivalent schema shown in Figure 11.

When passed to Rmap, the conceptual schema in Figure 11 maps to a relational schema with just three normalized table schemes. For most situations, this 3-table schema is much more efficient than the 7-table schema. The richness of the ORM graphic constraint notation

**Figure 10: This ORM schema maps to a relational schema with 7 table schemes**

**Figure 11: An “optimized” ORM schema semantically equivalent to Figure 10**
enables such transformations to be easily visualized. In contrast, a modeler who wishes to perform such transformations in UML is often forced to resort to a textual language to specify how constraints are reshaped by the transformation. This makes it much harder to transform schemas or to see optimization opportunities. For example, the frequency constraint of “2” in Figure 10 is reshaped into exclusion and equality constraints when the nomination predicate is specialized into the two nomination predicates in Figure 11. Although the frequency constraint of “2” in Figure 10 can be reformulated in UML using a “[0,2]” multiplicity constraint on the referee role, the corresponding exclusion and equality constraints in Figure 11 cannot be diagrammed in UML because these kinds of constraints are not built into UML.

Even though the additional graphical constraints in ORM often arise in practical applications, our experience suggests that UML modelers typically omit these constraints unless they are very experienced. UML and ORM both permit users to add other constraints and derivation rules in a textual language of their choice. UML suggests OCL for this purpose (Warmer & Kleppe, 1999), but does not mandate its use. Although OCL is a formal and hence unambiguous language, its mathematical syntax renders it unsuitable for validating rules with non-technical domain experts. ORM’s conceptual query language, ConQuer, is both formal and readily understandable to non-technical users, and its attribute-free nature makes it much more semantically stable than an attribute-based language such as OCL (Bloesch & Halpin, 1996, 1997; Halpin & Bloesch, 1999). Although textual languages are needed for completeness, it is much easier for a modeler to think of a rule in the first place if it is part of his/her graphical rule language. Although the UML notation can be extended to cater for the extra ORM constraints, some of these require either radical surgery or awkward changes to the UML metamodel (Halpin, 2001b).

CONCLUSION

This chapter identified the following principles for evaluating modeling languages and applied them in evaluating UML and ORM for conceptual data modeling: expressibility; clarity; semantic stability; semantic relevance; validation mechanisms; abstraction mechanisms; and formal foundations. Although ORM’s richer constraint notation makes it more expressive graphically, both methods extend expressibility through the use of textual languages. ORM scores higher on clarity, because its structures may be directly verbalized as sentences, it is based on fewer, more orthogonal constructs, and it reveals semantic connections across domains. Being attribute-free, ORM is more stable for both modeling and queries. ORM is easier to validate (through verbalization and multiple instantiation). Both methods are amenable to similar abstraction mechanisms, and have adequate formal foundations.

UML class diagrams are often more compact, and can be adorned with a vast array of implementation detail for engineering to and from object-oriented programming code. Moreover, UML includes mechanisms for modeling behavior, and its acceptance as an OMG standard is helping it gain wide support in industry, especially for the design of object-oriented software. Thus, both methods have their own advantages. For data modeling purposes, it seems worthwhile to provide tool support that would allow users to gain the advantages of performing conceptual modeling in ORM, while still allowing them to work with UML. Visual Studio Enterprise Architect already supports transformations between ORM, ER, Relational and Object-Relational models, as well as fully supporting UML.
Unlike UML, ORM is based on a small set of easily mastered, orthogonal concepts. UML modelers willing to learn ORM can get the best of both approaches by using ORM as a front-end to their information analysis and then mapping their ORM models to UML, where ORM constraints with no UML counterpart can be captured in notes or formal textual constraints. This option will become more attractive once commercial tools provide automatic transformation between ORM and UML.

REFERENCES


Chapter XVII

Formal Approaches to Systems Analysis Using UML: An Overview

Jonathan Whittle
NASA Ames Research Center, USA

Formal methods, whereby a system is described and/or analyzed using precise mathematical techniques, is a well-established and yet, under-used approach for developing software systems. One of the reasons for this is that project deadlines often impose an unsatisfactory development strategy in which code is produced on an ad hoc basis without proper thought about the requirements and design of the piece of software in mind. The result is a large, often poorly documented and un-modular monolith of code that does not lend itself to formal analysis. Because of their complexity, formal methods work best when code is well structured, e.g., when they are applied at the modeling level. UML is a modeling language that is easily learned by system developers and, more importantly, an industry standard, which supports communication between the various project stakeholders. The increased popularity of UML provides a real opportunity for formal methods to be used on a daily basis within the software lifecycle. Unfortunately, the lack of precision of UML means that many formal techniques cannot be applied directly. If formal methods are to be given the place they deserve within UML, a more precise description of UML must be developed. This chapter surveys recent attempts to provide such a description, as well as techniques for analyzing UML models formally.

The Unified Modeling Language (UML) (Object Management Group, 1999; Booch, Jacobson, & Rumbaugh, 1998) provides a collection of standard notations for modeling almost any kind of computer artifact. UML supports a highly iterative, distributed software development process, in which each stage of the software lifecycle (e.g., requirements capture/analysis, initial and detailed design) can be specified using a combination of particular UML notations. The fact that UML is an industry standard promotes communication and understanding between different project stakeholders. When used within a commercial tool (e.g., Rhapsody [I-Logix Inc, 2001], Rational Rose [Rational Software Corporation, 2001]) that supports stub code generation from models, UML can alleviate many
of the traditional problems with organizing a complex software development project. Although a powerful and flexible approach, there currently exist a number of gaps in the support provided by UML and commercial tools. First and foremost, the consistency checks provided by current tools are limited to very simple syntactic checks, such as consistency of naming across models. A greatly improved process would be obtained if tools were augmented with deeper semantic analyses of UML models. Unfortunately, although many of these techniques already exist, having been developed under the banner of Formal Methods, they cannot be applied directly to UML. UML is, in fact, grossly imprecise. There is as yet no standard formal semantics for any part of UML, and this makes the development of semantic tool support an onerous task.

This chapter gives an overview of current attempts to provide an additional degree of formality to UML and also of attempts to apply existing Formal Methods analyses to UML models. Space prevents the presentation of too much detail, so the description is at a more introductory level. Our starting point is the UML definition document itself (Object Management Group, 2000), which actually includes a section on UML semantics. Unfortunately, this semantics is by no means formal but provides merely a collection of rules or English text describing a subset of the necessary semantics.

UML is mainly a diagrammatic modeling language, consisting of various graphical notations for modeling all aspects of an object-oriented system and its design. In addition, UML contains a textual language, Object Constraint Language (OCL) that can be used to specify additional constraints on a UML model. A UML class diagram is a notation for modeling the static structure of a system. It describes the classes in a system and the relationships between them.

Figure 1 shows an example of a class diagram for part of an Automated Teller Machine (ATM) example. In object-oriented fashion, the main class (here “ATM toplevel”) is broken down into sub-classes. The aggregation relation ( ) shows when one class is part of another one. If the diamond shape is filled, the relation is a composition, which means that the part of relationship is strict, in the sense that the part cannot exist without the whole. The generalization relation ( ) shows when one class is an instance of another. Classes can also be related by simple associations, which are drawn as lines between classes. For further details, see Object Management Group (2000). Statecharts, introduced originally by David

Figure 1: A UML class diagram
Harel (1987) are finite state machines extended with hierarchy and orthogonality (parallelism), allowing a complex system to be expressed in a more compact and elegant way.

Figure 2 shows a simple example of a statechart. Nodes can either be simple nodes, or composite nodes that themselves contain other statecharts. The initial node in a statechart is marked by a bullet. Transitions between states have labels of the form $e[c]/a$. If event $e$ occurs and guard $c$ holds, then the transition may be selected to fire which results in action $a$ being taken and a state change occurring. This behavior is extended in a natural way to handle composite nodes. In Figure 2, if the system is in state $B$ when the transition $e[c]/a$ fires, then the system changes to $A1$. If the system is in any of the states $A1$, $A2$, $A3$ when transition $f[d]/b$ fires, then it will change to state $C$. By using composite nodes, hierarchy (or depth) can be introduced. This not only decreases the number of individual nodes substantially, but also enhances readability and maintainability of a statechart. A UML sequence diagram shows the interaction between objects of a system over time. The sequence diagram in Figure 3 is an example of an interaction between the objects “User,” “ATM,” “Consortium,” and “Bank.” The vertical lines represent the life-line (or time-line) for the given object, defining the object’s life during the interaction. Messages (like “Insert card”) are exchanged between the objects.

To motivate the need for a more formal semantics for UML, consider the case of sequence diagrams. Sequence diagrams, derived in part from their close neighbor, Message Sequence Charts (MSCs) (ITU-T, 1996), are a way of visualizing interactions (in the form of message sending and receipt) between different objects in the system. They can be used either in the early stages of development as a way of expressing requirements, or in later stages as a way of visualizing traces (e.g., error traces) of a design or actual code. It is in their use as requirements specifications that their inherent ambiguities are most worrying. Misunderstandings at this stage of development can easily lead to costly errors in code that might not be discovered until final testing.

The simplicity of sequence diagrams makes them suitable for expressing requirements, as they can be easily understood by customers, requirements engineers and software developers alike. Unfortunately, the lack of semantic content in sequence diagrams makes them ambiguous and therefore difficult to interpret. For example, is the intended semantics of Figure 1 that the interaction may take place within the system, or that it must take place? UML provides no standard interpretation. Expressions in square brackets denote conditions,

**Figure 2: Example of a UML statechart**
but it is not clear from the Standard whether a condition should only apply to the next message or to all subsequent messages—in the figure, does the [card inserted] condition apply only to the “Cancel” message or to all following messages? It is these kinds of ambiguities that could lead to costly software errors. In practice, each stakeholder would probably impose his or her own interpretation (possibly ambiguous in itself) of the sequence diagrams. When these are passed to a design modeler, the designer may introduce yet another interpretation. The result is a loss of traceability between software phases and a high degree of unintended behaviors in the final system.

Further problems with interpretation occur when integrating multiple diagrams. In the case of integrating a collection of sequence diagrams, it is not specified, for example, whether two sequence diagrams can be interleaved or must form completely separate interactions. This problem is experienced more generally when combining diagrams of different types (e.g., sequence diagrams and state diagrams).

**UML SEMANTICS**

**The Standard Semantics**

Version 1.3 of the UML Specification (Object Management Group, 2000) contains a section on UML Semantics, which is intended as a reference point for vendors developing tools for UML. The semantics is only semi-formal, however, and ultimately provides merely a set of guidelines for tool developers to follow in order to seek UML compliance. The abstract syntax of each modeling notation (class diagrams, use-case diagrams, sequence diagrams, state diagrams, activity diagrams, etc.) is given as a model in a subset of UML consisting of a UML diagram with textual annotations. The static semantics of each notation is given as a set of well-formedness rules expressed in the Object Constraint Language (OCL) (Warmer & Kleppe, 1999) and English. The use of OCL gives a misleading air of formality to this part of the semantics. OCL is a side-effect free, declarative language for expressing constraints, in the form of invariant conditions that must always hold. Constraints
can be placed on classes and types in a model, used as pre- and post-conditions on methods, and be used to specify guards. However, as we shall see, OCL does not have complete formal rigor, and hence its use to express well-formedness rules is a little unfortunate. The dynamic semantics in the UML Specification is given as English text.

The official semantics is based on the four-layer architecture of UML, consisting of user objects, model, metamodel and meta-metamodel layers. The semantics is primarily concerned with the metamodel layer. Table 1 reproduces the four-layer architecture from the UML Specification.

Each layer is further divided into packages (Foundation, Core and Behavioral Elements). The Core package defines the semantics of basic concepts (e.g., classes, inheritance) and the Behavioral Elements package deals with other notations (e.g., sequence diagrams, state diagrams). Each feature of UML is introduced with an English language description followed by its well-formedness rules as OCL constraints and its dynamic semantics as English text. To illustrate the inadequacy of the semantics, consider the specification for sequence diagrams. Sequence diagrams are a particular type of collaboration (collaboration diagrams are another type). Collaborations are stated to consist of a number of interactions, where an interaction specifies the communication between instances performing a specific task. Interactions in turn consist of messages sent between instances, where each message has, among other things, an activator (a message that invokes the behavior causing the dispatching of the message) and a set of predecessors, or messages that must have been completed before the current message can be executed. The use of predecessors could be utilized for resolving ambiguities—the predecessors give us a partial ordering on the messages, akin to that which appears in a sequence diagram. The well-formedness rules in fact give additional details about predecessors, such as “A message cannot be the predecessor of itself,” expressed in OCL as:

\[
\text{not self.allPredecessors-> includes (self)}
\]

and “The predecessors must have the same activator as the Message”:

\[
\text{self.allPredecessors-> forAll (p | p.activator = self.activator)}
\]

However, the semantics still says nothing about how to combine multiple sequence diagrams, or about the existential or universal nature of a sequence diagram. As such, the Standard gives a framework in which to express semantic constraints, but is incomplete.

**Formalizing the Semantics**

There is a large body of research on formalizing object-oriented systems and graphical notations associated with them. Most of this work has concentrated on the static aspects of the system (e.g., class diagrams). A number of different formalisms exist for the basic concepts of class diagrams, e.g., Bourdeau and Cheng (1995); Overgaard (1998); Wieringa and Broersen (1998). A full explanation of these is beyond the scope of this paper. Less work has been done on modeling the semantics of the dynamic aspects of systems (i.e., sequence diagrams, state diagrams, etc.) but some research is appearing (Jackson, 1999; Overgaard, 1999; Araujo, 1998).
The Precise UML Group

A group of researchers, affiliated with the UML Standardization effort, have recognized the impreciseness of the official UML semantics and have formed the Precise UML Group (pUML) (2000) to investigate ways of developing UML as a formal modeling language. The approach of the pUML is to define a core semantics for essential concepts of the language and then to define the rest of the semantics, in a denotational manner, by mapping into this core semantics. For example, the meaning of a sequence diagram might be understandable in terms of a subset of the behavioral semantics of the core. This approach is consistent with the standard semantics of the metamodel, and indeed, the intention is to build on the current informal semantics, rather than integrating many varied and disparate semantics. While this is a noble effort, it is inevitably a huge undertaking, and so thus far, only small portions of UML have been worked on (e.g., see Evans, France, & Lano, 1999, for a discussion of semantics for generalization and packages).

To give a small flavor of the work of the pUML, consider part of the task of making generalization precise, taken from Evans, France, and Lano (1999) and Bruel and France (1998). In the UML document, the more generic term for a class is a classifier. The metamodel fragment for relating a classifier to its instances is given in Figure 2. This is given along with English language and OCL annotations. The pUML addition to the semantics essentially adds further OCL constraints corresponding to the English text.

Page 2-36 of Object Management Group (2000) states that a set of generalizations is disjoint in the sense that an instance of the parent classifier may be an instance of no more than one of the children. This is formalized by pUML with the following OCL expression:

\[
\text{context } c : \text{ Class inv } \\
c.\text{specialization.child} \rightarrow \\
\text{forall}(i : \text{ Classifier } | \text{forall } j : \text{ Classifier } | \\
i <> j \text{ implies } i.\text{instance} \rightarrow \\
\text{intersection(j.instance) } \rightarrow \text{isEmpty})
\]

which says that for any two children, i and j, of a class, the set of instances of i and j are disjoint.
Another important idea to come out of the pUML group is the notion of a UML *preface* (Cook et al., 1999). It has been proposed that the complexity of UML and its widespread use make it impossible to settle on a universal semantics for the language. The use of UML in a particular domain may require a different semantics than in another domain. For example, there are many different possible interpretations for the orthogonality construct in a UML statechart, depending on the level of parallelism or synchronicity required, and each interpretation may be suited to a particular domain. It is for this reason that the Object Management Group (OMG) actively supports extensions to UML through the use of stereotypes and tagged values. Moreover, the OMG is soliciting proposals for UML *profiles* or specializations of UML (e.g., by including additional well-formedness rules) tailored to specific domains (e.g., real-time systems, Enterprise Distributed Object Computing). A UML *preface* is a very similar concept to a profile except that the importance of specifying the semantics is made paramount for a preface. The idea is that each UML model should come with a preface that defines how that model’s elements should be interpreted. In this way, a model could be used to mean different things in different contexts, depending on which preface is used as a basis. Since prefaces will likely be large documents, a standard library of prefaces could be provided from which additional prefaces could be built up if so desired.

**Other Relevant Semantic Efforts**

The pUML work has so far concentrated on basic concepts like objects and inheritance which, unfortunately, do not deal with how sequence diagrams, state diagrams, etc., are meant to be interpreted. Formal semantics already exist for variants of some of the UML notations. State diagrams are a typical example. State diagrams are derived from Harel’s statecharts (Harel, 1987) and a large number of semantics have been developed for various versions of statecharts (see von der Beek, 1994, for a comparison of many of these). None of these can be directly applied to UML state diagrams because the semantics of UML state diagrams is based on an object-oriented model. Sequence diagrams are another good example. They are derived from (among other things) the Message Sequent Chart (MSC) notation (ITU-T, 1996a) for which a formal standard semantics based on process algebras already exists (ITU-T, 1996b). As Damm and Harel (1999) point out, however, this semantics still leaves open a number of fundamental questions. In an attempt to overcome this, Harel has produced a formal semantics for his own version of MSCs (Damm & Harel, 1999). He defines Live Sequence Charts (LSCs) as a conservative extension of MSCs, the main addition being the ability to specify a *temperature* (hot or cold) on both the vertical timelines in the chart and on the messages. Hot timelines indicate interactions that *must* take place, whereas cold timelines indicate possible interactions. Similarly, hot messages must be sent (or be received) whereas cold ones *might* be sent/received. The motivation behind this is that developers actually use MSCs differently depending on the stage of development. Early in the software lifecycle, the designer is more interested in possible behaviors whereas, later on, the emphasis is usually on specifying more precisely exactly what behaviors will occur and when. Another notation
similar to UML sequence diagrams is Timed Sequence Diagrams, a formal semantics of which, based on trace theory, is presented in Facchi (1995).

It remains an open question if it is possible to integrate existing semantics such as these with the metamodel semantics pursued by the pUML group.

**Object Constraint Language**

The Object Constraint Language (OCL) is a side-effect free, declarative, constraint language included in UML both to allow users to express additional constraints that could not be expressed in the existing notations, and to facilitate specification in a formal, yet comprehensible way. It was designed to bring the advantages of formality and yet be readable enough for use by software developers. The specification of OCL itself, however, is largely given by an English language description that is often ambiguous or inconsistent. As an example of this, consider the following OCL expression (taken from Richters and Gogolla, 1998):

```plaintext
context rs : RentalStation
inv
rs.employee->iterate(p: Person;
    names : String = "" | names.concat(p.lastname))
```

This query is intended to build a list of names of employees of a car rental station. Object Management Group (2000) prescribes no statement about the order of evaluation if the structure being iterated over is a set or a bag, hence evaluations may yield different results caused by different iteration sequences.

Richters and Gogolla (1998) present a set-based semantics of OCL (not currently accepted as a UML standard) and overcome the problem above by defining a deterministic evaluation semantics. For iteration over sequences, Richters and Gogolla take the elements in the order of the sequence (as in the UML Standard). For sets or bags, iteration must satisfy a precondition that states that the operation that combines expressions at each stage of the iteration (in this case, concat) is associative and commutative. If this precondition fails, then the result is dependent on the order of execution, and so the set or bag should be converted to a sequence first.

Vaziri and Jackson (1999) point out a number of shortcomings of OCL. It has been shown (Mandel & Cengarle, 1999) that OCL is, in terms of its expressiveness, not equivalent to the relational calculus (an example is given of an operation that cannot be encoded in the relational calculus but can be expressed in OCL).

**TOOL SUPPORT**

Despite the current lack of a precise semantics, there have been a number of attempts to provide sophisticated tool support for UML. In this paper, we concentrate on those approaches that are already well established in the Formal Methods community.

**The Different Guises of Model Checking**

The term *model checking* has been used in a variety of different contexts with different meanings. This ranges from its use as meaning the checking of simple syntactic constraints (e.g., in the Rhapsody tool, which offers this capability) to its use in the Formal Methods community (McMillan, 1993; Holzmann, 1997) to describe a technique for exhaustively
searching through all possible execution paths of a system, usually with the intention of detecting a system state that does not satisfy an invariant property, expressed in temporal logic.

Model checking, in the latter sense, is now a well-established technique for validating hardware systems, and as such, has been incorporated into commercial CASE tools. However, the use of model checking in the software world is more problematic. Software systems generally tend to be more complex than hardware, containing more intricate data structures that may involve hierarchy, making them structured differently from hardware systems. In addition, software is often specified with infinite state descriptions, whereas model checking can only be applied to finite state systems (as is the case for hardware).

There are two main types of model checking. In both cases, the idea is to enumerate each possible state that a system can be in at any time, and then to check for each of these states that a temporal logic property holds. If the property does not hold, a counterexample can be generated and displayed to the user (this is done using MSCs in the Spin model checker (Holzmann, 1997)). Explicit model checking represents each state explicitly and exploits symmetries for increased performance. Symbolic model checking, on the other hand, uses an implicit representation in which a data structure is used to represent the transition relation as a Boolean formula. In this way, sets of states can be examined together—those that satisfy particular Boolean formulas. The key to the success of the method is the efficient representation of states as Boolean formulas, and the efficient representation of these Boolean formulas as Binary Decision Diagrams (BDDs).

Despite recent successes in model-checking software, (e.g., Havelund, Lowry, & Penix, 1997), all approaches fall foul of the so-called state space explosion. The sheer complexity of software systems leads to a state space that is too large (e.g., > 2^{1000} states) to be explored exhaustively. This means that only very small systems can be analyzed automatically. There exist techniques for dealing with this problem, most notably the use of abstractions (Lowry & Subramaniam, 1998) that reduce the size of the model by replacing the model by a simpler but equivalent one. The drawback here is that coming up with useful abstractions is a highly creative, and hence non-automatable task. In addition, the equivalence of the abstracted and original system may need to be proven formally, which generally requires an intimate knowledge of theorem-proving techniques. Another promising technique is to modify the model checking algorithms, developed for hardware, to take into account the structure of the software system. This idea has the potential to alleviate the state space explosion problem, but research in this area is very much in its infancy. Structure can be exploited either by compositional model checking (de Roever, Langmaack, & Pnueli, 1998) in which a complex system is broken down into modules that are checked individually, or by hierarchical model checking (Alur & Yannakakis, 1998) in which the hierarchy in a system is used to direct the search.

Despite these drawbacks, it still seems that incorporating existing model-checking techniques into UML would be worthwhile. Indeed, the highly structured nature of most UML models (yielded by the object-oriented style and also hierarchy in state diagrams) may mean that model checking is likely to be more successful when applied to UML than to other languages.

So far, there have been surprisingly few attempts to apply model-checking to UML and all of these have concentrated on checking state diagrams. UML state diagrams are a variant of Harel’s statecharts (Harel, 1987) and have always been an attractive target for model-checking software because they are more abstract than code. Indeed, there has been a reasonably large body of work on model-checking statecharts (Day, 1993; Mikk, Lakhnech,
Siegel, & Holzmann, 1998), including case studies on large, real-world systems such as the TCAS II (Traffic Alert and Collision Avoidance System II) (Chan, Anderson, Beame, & Notkin, 1998), a complex airborne collision avoidance system used on most commercial aircraft in the United States. One issue that has always plagued researchers attempting to analyze statecharts is the choice of a semantics. Harel did produce an official semantics for statecharts, as incorporated in iLogix’s CASE tool STATEMATE (I-Logix Inc., 1996), but this is not a compositional semantics and so does not support compositional checking of large specifications.

The semantics of UML state diagrams differ significantly from Harel’s statecharts. This stems from the fact that STATEMATE statecharts are function-oriented, whereas UML state diagrams are object-oriented. As a result, both Paltor and Lilius (1999) and Harel and Grey (1997) give UML state diagrams a semantics that includes an event queue for each object and a run-to-completion algorithm that dispatches and executes events on an object’s event queue until the top state generates a completion event and the state machine exits. This is in contrast to the STATEMATE semantics in which events reach their destinations instantly.

vUML applies model checking based on the semantics given in Paltor and Lilius (1999). The key idea of the vUML tool is that UML state diagrams and an associated collaboration diagram are translated into the PROMELA language, which is the input language for the widely recognized SPIN model checker (Holzmann, 1997). A collaboration diagram (see Figure 3) is equivalent to a sequence diagram, but emphasizes the relationship between objects in an interaction rather than timing constraints between messages. The collaboration diagram is required to specify the instances of classes present in a particular execution. vUML is also capable of displaying counterexamples from the SPIN model checker as UML sequence diagrams. Note that SPIN itself generates counterexample traces as MSCs, but vUML then gives a sequence diagram in terms of the original UML objects and messages, not the internal PROMELA syntax used in the model checking. As yet, vUML has only been tested on a small number of examples, including the well-known Dining Philosophers problem and the verification of a Production Cell. Although the vUML semantics is not compositional, the authors claim that large models can be broken down into parts and model checked by using a feature that non-deterministically generates events not specified in the current model.

JACK (Gnesi, Latella, & Massink, 1999) is a tool suite for various formal methods techniques that includes a model checker for UML state diagrams based on branching time temporal logic. The main difference between JACK and vUML is that hierarchy within state diagrams is treated as a first-class citizen in JACK. This means that when translating into the model checker’s input language, JACK preserves the hierarchical structure of the state diagrams, whereas vUML “flattens out” the diagrams. In principle, this preservation of structure is a necessary step if hierarchical model-checking algorithms are going to be used for UML in the future. The structure preservation is done by first translating UML state diagrams into hierarchical automata (HA) (Mikk, Lakhnech, & Siegel, 1997), which are essentially statecharts but without inter-level transitions, i.e., transitions with source and target at different levels in the hierarchy. Inter-level transitions are akin to GOTO statements in imperative programming and as such, make formal analysis near impossible because they destroy modularity. The trick in translating to HAs is that each inter-level transition is “lifted” to the same level by annotating it with additional information regarding which sub-state of its target state it will enter upon firing. The statechart model checker MOCES (Mikk, Lakhnech, Siegel, & Holzmann, 1998) uses the same technique for model checking function-oriented statecharts.
Model Finding

Another technique, which is often also referred to as model checking but which is more properly termed model finding, is the enumeration of all possible assignments until a model is found (i.e., an assignment in which the given formula is true). A UML specification can be translated into a formula, and model finding can then be used either for simulation, in which models of the specification are presented to the user, or for detecting errors, in which case the model finder is applied to the negation of the formula and if a model is found, it is a counterexample. In this way, simulation is useful in the early stages of development—suggested models may be surprising and suggest necessary modifications. Model checking is useful in the later stages and can be used to find model counterexamples. Note that model finding does not carry out a temporal analysis as is the case for model checking in the previous section, nor does it compute reachable states. However, model finding can be an extremely useful technique for ironing out bugs in software specifications.

Perhaps the most successful application of model finding so far is the Alloy Constraint Analyzer system that is based not on UML but on an alternative object-modeling language called Alloy (Jackson, 1999). Alloy is not as extensive as UML but essentially consists of a language for expressing object diagrams very similar to class diagrams, and a relational language similar to OCL, but designed to overcome some of the drawbacks of OCL mentioned earlier. In particular, Alloy is intended to be similar in spirit to OCL and the relational specification language Z (Davies & Woodcock, 1996) but more amenable to automatic analysis.

The Alloy Constraint Analyzer is a model finder for Alloy based on an earlier incarnation Nitpick (Jackson, 1996), but using different underlying technology. Whereas Nitpick looked for all possible assignments and attempted to reduce the number of possibilities by looking for symmetries, the Constraint Analyzer translates an Alloy model into a huge Boolean formula. This Boolean formula can then be checked for satisfaction using standard SAT solvers. This translation is only possible because the user specifies a scope over which model finding is carried out. This scope states the maximum number of instances of each particular class that are allowed and reduces any Alloy model to a finite one. Relations can then be translated into a matrix of Boolean formulas. For each relational variable, a matrix of Boolean variables is created. Each term is then translated by composing the translations of its sub-

Figure 5: A collaboration diagram for the Dining Philosophers
terms. As an example, if terms $p$ and $q$ are translated into matrices $[p]$ and $[q]$ respectively, then $[p]_j$ is a Boolean formula that is interpreted as being true when $p$ maps the $i$th element of its domain type to the $j$th element of its range type. Similarly for $[q]_j$. The translation of the term $p \cap q$ is given by

$$[p \cap q]_j = [p]_j \land [q]_j$$

Note that all matrices are finite because the scope imposes finite limits on the number of instances.

The Alloy Constraint Analyzer turns out to be reasonably successful at ironing out simple errors in the early stages of development. The author of the tool states that a handful of relations with a scope of size 3 can be handled easily. Any larger scope and the tool begins to fade. This limit of size 3 scope appears at first sight to be extremely limiting. However, the developers of the Constraint Analyzer claim that most problems can be discovered using a very small scope, and that most errors present in specifications with a large number of instances are also present if the number of instances is restricted. The Alloy Constraint Analyzer is still very much a research prototype and so further experimentation is needed to see if indeed this claim holds. The Constraint Analyzer is in many senses similar to the FINDER model finding tool (Slaney, 1995), although it has been developed specifically with object models in mind.

### OTHER APPROACHES

Model-checking and model-finding techniques have found greater acceptance both in general and within UML than have approaches based on theorem proving. Theorem provers are very general purpose machines for carrying out mathematical analysis. In principle, they can take a UML description as input and automatically or interactively be used to prove properties about the UML model. In practice, however, theorem proving is an extremely difficult task to automate and although there has been some success (Richardson, Smaill, & Green, 1998), most theorem proving is highly interactive, making it unsuitable for the average software engineer. Theorem proving can be successful in a very restricted domain. The Amphion system (Lowry, Philpot, Pressburger, & Underwood, 1994) uses theorem proving in a restricted setting (that of space trajectories) to automatically synthesize Fortran functions. These functions are extracted from a proof that a specification holds and can be synthesized using very little or no knowledge about theorem proving. It may be possible to identify restricted subtasks in analyzing UML models that theorem proving could be applied to in a similarly fruitful way. One possibility is to synthesize UML state diagrams from a collection of sequence diagrams annotated with OCL constraints. Certain OCL expressions correspond to a particular implementation as a statechart, for example, an OCL expression stating that the order of dispatch of a set of messages is irrelevant would be implemented as a statechart differently than an OCL expression enforcing a particular execution order of messages. Theorem proving could be used to refine a collection of OCL expressions into those that correspond to a particular statechart implementation. In this way, theorem proving would yield an intelligent synthesis of statecharts from sequence diagrams. First steps in this direction have already been taken (Whittle & Schumann, 2000).

### Rewriting

A technique that is in many ways similar to theorem proving is rewriting, in which a rewriting engine applies rewrite rules to rewrite a term into an equivalent one. Very efficient
rewriting engines now exist, e.g., the Maude system (Clavel et al., 1999), and there are signs that such engines are being applied to UML development. Work underway at the University of Murcia, Spain, (e.g., Aleman & Alvarez, 2000) attempts to formalize UML using the algebraic specification language of the Maude system (which is based on the OBJ3 specification language). The specification language of Maude is rewriting logic, a logic based around the use of rewrite theories. A rewrite theory \((T, R)\) is a pair with \(T\) a membership equational theory and \(R\) a collection of labeled and possibly conditional rewrite rules. The Murcia group uses rewriting logic to express the abstract syntax of UML (as given in the UML specification) as well as the well-formedness rules of the definition of the static semantics. Since rewriting logic can be executed in Maude, a by-product is that a UML model can be checked for conformance to the well-formedness rules by writing these rules as rewrites and then applying them exhaustively. The idea is to provide rewrite rules that rewrite a badly formed UML model into an expression containing an exception. By considering each kind of exception as a violation of a particular well-formedness rule, feedback can be given to the user regarding which rule has been violated.

Transformations on UML Diagrams

The framework described in the previous paragraph can also be used to express transformations on UML models. In this context, transformations can be seen as formal rewrite rules that when applied to a UML model provide a different viewpoint, translate to another notation, optimize a model, etc. The Murcia group has carried out initial work on making information hidden in a UML model explicit, by using transformations to reveal that information. For any given UML model, there may be many different ways of expressing the information. For example, consider a class diagram in which class \(A\) is a subclass of \(B\) which is in turn a subclass of \(C\). Although it may not be given explicitly, this diagram shows that \(A\) is also a subclass of \(C\). In certain situations, it may be useful to extract such hidden information using transformations. Aleman and Alvarez (2000) describe how this approach can be used to derive new associations on class diagrams. In a class diagram, the ends of an association can be optionally annotated with a multiplicity that is a range of integers. Multiplicities describe how many instances of a particular class can be associated with a single instance of the class at the opposite end of the association. When deriving new associations, new multiplicities must also be derived. The algorithm in Aleman and Alvarez (2000) shows how these multiplicities can be inferred in practice.

Some work on providing deductive transformations has been done by the pUML group members. The motivation there has been to support transformational development—the refinement of abstract models into more concrete models, using design steps that are known to be correct with respect to the semantics. This development methodology has been investigated for a long time within the context of developing correct implementations of safety-critical systems. The approach, presented in Lano (1998), is based on a semantics given in terms of theories in a Real-time Action Logic (RAL). An RAL theory has the form:

\[
\text{theory name} \\
\text{types local type symbols} \\
\text{attributes time-varying data representing instance or class variables} \\
\text{actions actions that may affect the data such as operations, statechart transitions and methods} \\
\text{axioms logical properties and constraints between the theory elements}
\]
A UML class $C$ can then be represented as a theory:

\[
\text{theory } \Gamma_c \\
\text{types } C \\
\text{attributes } C : \exists(C) \\
\quad \text{self} : C \rightarrow C \\
\quad \text{att}_i : C \rightarrow T_i \\
\quad \ldots \\
\text{actions } \text{create}_c(c : C) \{C\} \\
\quad \text{kill}_c(c : C) \{C\} \\
\quad \text{op}_i(c : C, c : X_i) : Y_i \\
\quad \ldots \\
\text{axioms } \\
\quad c : C \cdot \text{self}(c) = c \land [\text{create}_c(c)](c \in C) \land [\text{kill}_c(c)](c \notin C)
\]

$\exists$ is the “set of finite sets of,” $X$ is the set of existing instances of $X$. For an action symbol, $\alpha$, and predicate $P$, $[\alpha]P$ is a predicate meaning “every execution of $\alpha$ establishes $P$ on termination.”

Other parts of the UML (e.g., associations) can be formalized similarly. The introduction of the RAL representation of UML means that transformations can be defined on UML and proven to be correct. Lano (1998) identifies three kinds of transformation:

- Enhancement transformations that extend a model with new model elements.
- Reductive transformations that reduce a UML model in the full language to a model in a sub-language of UML.
- Refinement transformations that support rewriting models in ways which lead from analysis to design and implementation.

As an example, Figure 4 shows a refinement based on composition of compositions. The refinement adds an extra composition, in the case that $A$, $B$ and $C$ have no common objects. The approach to assuring correctness is that each UML component should satisfy certain properties; e.g., composition should be:

- one-many or one-one (a whole can have many parts, but a part cannot belong to different wholes at the same time);
- deletion propagating - deleting the whole deletes its parts;
- transitive;
- irreflexive.

For a particular transformation, such as that given in Figure 4, it can be proven using the RAL theories that the transformation preserves these properties.

This notion of correctness for transformations is very similar to the one used for assuring correctness of refactorings of C++ programs in Opdyke (1992). The transformations make only very minor changes to the model so that the proofs of correctness are usually trivial. In addition, the notion of correctness seems to be based on guaranteeing a set of predefined constraints, and hence cannot guarantee full behavioral correctness. UML does at least provide a framework for expressing such refactoring transformations. The properties to be checked can often be extracted from the UML specification. This is not the case for C++ refactorings, and, as a result, the refactorings in Opdyke are often overly constrained (so that their proofs remain trivial) to a point where they cannot be used in many real situations.
An interesting twist on this topic is presented in Evans (1998), in which the author proposes that properties of UML models (specifically class diagrams) be proven using diagrammatic transformations. The idea is that to check an invariant property of a model, the software developer expresses the property as another class diagram. Diagrammatic transformation rules are then used to transform the class diagram model into the class diagram property. This proves the property. All transformations are proved correct in a similar way to that in the previous paragraph, but using a set-based semantics expressed in the Z notation. The advantage of the approach is that no knowledge of complex logical, formal languages is required, as all theorem proving is done diagrammatically. On the other hand, the class of properties that can be proven is limited because the properties must be expressible as a class diagram. Moreover, if the class diagram contains additional OCL constraints, the soundness of the transformation rules may no longer hold.

A final system worth mentioning is UMLAUT (Ho, Jezequel, Le Guennec, & Pennaneac’h, 1999), a framework for implementing transformations on UML models, where transformations are based on pre-defined atomic transformations.

**CONCLUSION**

This chapter has presented an overview of some of the attempts to formalize the semantics of UML and to apply techniques from Formal Methods to analyzing UML models. Surprisingly, despite the plethora of analysis techniques from other formalisms such as statecharts (Harel, 1987) and Petri Nets (The World of Petri Nets, 2000), there has been relatively little transfer of these ideas to UML. The author believes that this is due to the largely informal nature of the UML semantics. As a result, more effort should be directed towards making the semantics precise. In addition, it is still worthwhile to investigate the application of formal techniques, even if the semantics is not fully decided. By taking a pragmatic approach, tools could support the well-understood part of the semantics and be designed in such a way that variations in the rest of the semantics could be quickly integrated. UML provides a great opportunity for practitioners in Formal Methods. In addition to offering a huge potential user base, the UML can be seen as a test bed for different techniques. For
instance, UML could provide a collection of standard examples to be used as case studies for the various techniques. The author believes that more effort should be invested in the application of Formal Methods to UML.

REFERENCES


Formal Approaches to Systems Analysis Using UML: An Overview


INTRODUCTION

In recent years, the need for a temporal dimension in traditional spatial information systems and for high-level models useful for the conceptual design of the resulting spatiotemporal systems has become clear. Although having in common a need to manage spatial data and their changes over time, various spatiotemporal applications may manage different types of spatiotemporal data and may be based on very different models of space, time, and change. For example, the term spatiotemporal data is used to refer both to temporal changes in spatial extents, such as redrawing the boundaries of a voting precinct or land deed, and to changes in the value of thematic (i.e., alphanumeric) data across time or space, such as variation in soil acidity measurements depending on the measurement location and date. A spatiotemporal application may be concerned with either or both types of data. This, in turn, is likely to influence the underlying model of space employed, e.g., the two types of spatiotemporal data generally correspond to an object- versus a field-based spatial model. For either type of spatiotemporal data, change may occur in discrete steps, e.g., changes in land deed boundaries, or in a continuous process, e.g., changes in the position of a moving object such as a car. Another type of spatiotemporal data is composite data whose components vary depending on time or location. An example is the minimum combination of equipment and wards required in a certain category of hospital (e.g., general, maternity, psychiatric), where the relevant regulations determining the applicable base standards vary by locality and time period.
A conceptual data-modeling language for such applications should provide a clear, simple, and consistent notation to capture alternative semantics for time, space, and change processes. These include point- and interval-based time semantics; object- and field-based spatial models; and instantaneous, discrete, and continuous views of change processes. Multiple dimensions for time (e.g., valid, transaction) and space should also be supported.

Although there has been considerable work in conceptual data models for time and space separately, interest in providing an integrated spatiotemporal model is much more recent. Spatiotemporal data models are surveyed by Abraham and Roddick (1999), including lower-level logical models (Claramunt, 1995; Langran, 1993; Pequet & Duan, 1995). Those models that deal with the integration of spatial, temporal, and thematic data at the conceptual level are the most relevant to this work and are reviewed here.

Several conceptual frameworks have been designed to integrate spatial, temporal, and thematic data based on Object-Oriented (OO) or Entity-Relationship (ER) data models that include a high-level query language capable of specifying spatiotemporal entity types. The data definition component of these query languages thus has some potential for use in modeling spatiotemporal applications.

Becker, Voigtmann, and Hinrichs (1996) and Faria, Medeiros, and Nascimento (1998) propose OO models based on extensions of ObjectStore and O2 respectively. Becker et al. consider both object- and field-based spatial models, defining a hierarchy of elementary spatial classes with both geometric and parameterized thematic attributes. Temporal properties are incorporated by adding instant and interval timestamp keywords to the query language. In Faria et al. spatial and temporal properties are added to an object class definition by associating it with pre-defined temporal and spatial object classes. This solution is not suitable for representing temporal or spatial variation at the attribute level, as the timestamp and spatial locations are defined only at the object component level. In addition, both Becker et al. and Faria et al. offer text-based query languages; the non-graphical query languages of these models reduce their suitability as conceptual modeling languages.

In a clear indication that the need for a spatiotemporal graphical modeling language has been well recognized, there have been several concurrent efforts to develop such a language recently reported in the literature. The MADS model (Parent, Spaccapietra, and Zimanyi, 1999) extends an object-based model with pre-defined hierarchies of spatial and temporal abstract data types and special complex data types to describe all of an attribute’s properties, i.e., name, cardinality, domain, and temporal or spatial dimensions. The use of a non-standard, hybrid ER/OO model and the definition of new composite data structures to incorporate spatiotemporal properties, rather than exploiting existing features of the ER or OO models, increase the complexity of this model. The SpatioTemporal ER model (Tryfona & Jensen, 1999) adds temporal and spatial icons to entities, attributes, and relationships in the ER model to support timestamped spatial objects and fields (i.e., spatiotemporal variation in thematic attributes). Composite data whose components vary over space and relationships associated with their own spatial extents are not considered: instead, spatial relationships are used to represent explicit geometric or topological relationships between associated spatial objects, which could otherwise be derived on demand. Therefore, temporal relationships describe model structure (i.e., timestamps), whereas spatial relationships describe model integrity (i.e., constraints). Peceptory (Brodeur, Bedard, & Proulx, 2000) is a spatiotemporal model and CASE tool aligned with both geographic and object-oriented standards, based on adding spatial and temporal stereotypes to objects in UML. The focus is on spatial objects; spatial
associations and fields are not supported. None of the models described above provide explicit support for modeling a group of thematic properties measured at the same times and locations, consider interpolation, or support alternative time models (i.e., periodic versus aperiodic recording of data values).

In this chapter, we describe an extension of UML proposed by Price, Tryfona, and Jensen (2000a) intended to address the goals outlined earlier, i.e., to support a range of spatiotemporal models and data types using a clear, simple, and consistent notation. Extending the OMG standard for OO modeling was selected as the best approach given its high level of acceptance, tool support, understandability, and extensibility. The result is the Extended Spatiotemporal UML (XSTUML). The language is extended further to support modeling of complex spatial objects and their topological constraints by Price, Tryfona, and Jensen (2000b) and Price, Tryfona, and Jensen (2001) respectively.

Spatial (and spatiotemporal) applications typically require modeling of complex asymmetric relationships between spatial objects where one object—the whole—is regarded as representative in some way of a group of other objects—the parts. We refer to this related group of spatial objects as spatial Part-Whole (PW) relationships. Price et al. (2000b) propose a classification framework and modeling constructs for spatial PW relationships based on both spatial and non-spatial characteristics, integrating and extending previous work examining spatial (Parent et al., 1999; Tryfona, Pfoser, & Hadzilacos, 1997) and non-spatial (Henderson-Sellers & Barbier, 1999; Motschnig-Pitrik & Kaasboll, 1999; Odell, 1994; Saksena et al., 1998; Winston, Chaffin, & Herrmann, 1987) characteristics of complex objects separately.

Of the three spatial characteristics—orientation, metrics, and topology—used to describe spatial PW relationships, topology serves as a particularly important descriptor in conceptual application modeling both because of its qualitative, thus more intuitive, nature (e.g., recognizing a specific topology such as contains is easier than recognizing a specific distance metric) and because—unlike orientation and metrics—it is preserved through many of the common distortions that can occur in representations of real-world objects. Since topological classifications proposed to date (Clementini, Di Felice, and Califano, 1995; Egenhofer, Clementini, & Di Felice, 1994; Hadzilacos & Tryfona, 1992; Tryfona & Egenhofer, 1997) do not consider the range of spatial object types or n-ary (set-based) topological relationships required to support modeling of PW relationships in spatial applications, Price et al. (2001) propose a topological classification and modeling constructs designed to address this requirement.

The rest of the chapter is organized as follows. The next section illustrates the problems with using UML to model spatiotemporal data from a Geographic Information Systems context. This is followed by a section that describes the syntax and semantics of the fundamental new constructs introduced—the spatial, temporal, and thematic symbols—in XSTUML. The subsequent section discusses three other symbols: the attribute group symbol, the existence-dependent symbol, and the specification box (used to specify the details of the spatiotemporal semantics). A section then shows how the example from the first section below would be modeled using XSTUML and compares the XSTUML schema to the equivalent UML schema. The penultimate section describes support for modeling spatial PW relationships and their topology, and it illustrates this using XSTUML. Conclusions are presented last.
USING UML FOR SPATIOTEMPORAL DATA

In this section, we evaluate the core constructs and extension mechanisms defined in UML (Booch, Rumbaugh, & Jacobson, 1999; Rumbaugh, Jacobson, & Booch, 1999; OMG, 1999) in terms of their suitability for modeling spatiotemporal data and defining a UML extension to facilitate such modeling respectively. The UML usage and notation used is based on Rumbaugh et al. (1999), except that we use informal textual descriptions for complex attribute domains and constraints for the sake of readability. We use Backus-Naur Form (BNF) for specific explanations of syntax or terminology.

The following regional health application will be used to illustrate the use of UML to model spatiotemporal data. Assume an application measuring health statistics of different provinces, in terms of average lifespan, as related to the location (i.e., a point in 2D space), number of beds, accessibility (i.e., a half-hour travel zone around the hospital), and surrounding population densities of a province’s hospitals. A hospital is classified by category, where a given category is required to have a minimum number of beds in specific kinds of wards. However, category definitions may differ between regions due to local regulations.

For properties dependent on time and/or location, we want to record information about when (using time intervals unless otherwise specified) and/or where a given value is valid (i.e., valid time) or current (i.e., transaction time). For example, a province’s population densities and average lifespans can vary and are recorded yearly at the same time instants (values are averaged between yearly measurements) and for the same regions. The number of beds, the half-hour travel zone, a hospital’s category, and the regional definition of hospital categories may change over time as well. We want to record existence and transaction time for hospitals, valid time and transaction time for a hospital’s category, and valid time for all of the other time dependent properties. The time unit for the half-hour travel zone is not yet specified, demonstrating incremental design specification. Time elements are used to model hospital existence time, since hospitals may sometimes be closed and later re-opened based on changes in local population density. Note that the number of beds, half-hour travel zone, and hospital category are only defined when the hospital is open.

Representation of spatiotemporal concepts using the core constructs of UML is not straightforward, as is illustrated using the regional health example in Figure 1. Figure 1 uses the following BNF definitions:

```
spatial-extent := { point | line | region | volume } #
timestamp := { instant | interval | element } #
```

Attributes with spatial and/or temporal properties (e.g., the half-hour travel zone or number of hospital beds) can be modeled (e.g., `halfHourZone` or `numBeds` attributes, respectively) using composite attribute domains consisting of a set of tuples, where each tuple consists of a thematic value, spatial extent, and/or timestamp(s). Alternatively, an attribute with spatial and/or temporal properties (e.g., population density or average lifespan) could be promoted to a separate but associated class with the same information added to the new class. Although not required by the semantics of the example application, we must also create an artificial `identifier` attribute for this class because its instances must be uniquely identified (see Rumbaugh et al., 1999, pp. 304, 307). Of more concern, this approach will lead to redundancy whenever the same attribute value is repeated for different object instances, times, and/or spatial extents. This is especially significant for spatial data because of their size.
A more correct approach, in general, would be to promote the association to an association class (e.g., Has) with spatial data in the associated class (e.g., Measurement-Region) and thematic and/or timestamp data (e.g., populationDensity, averageLifespan, and valid-time) in the association class. This still does not solve the problem of the artificial identifier or the extra complexity introduced for adding classes. However, this approach is preferred when (1) the same spatial extent is associated with different object instances or timestamps or (2) several attributes are associated with the same timestamps or spatial extents. Classes and associations with temporal and/or spatial properties (e.g., Hospital and hospital Is-of category, respectively) can be treated similarly by adding timestamp and/or spatial attributes, after promoting the association to an association class in the latter case.

Constraints are used to indicate the time units for timestamps, the time model, the dimensions of spatial extents, and the existence-dependencies described for the application example. Notes are used to show interpolation semantics. Association, rather than generalization, is used to represent hospital category since its definition varies regionally and does not affect the properties defined for the hospital class.

Figure 1: Regional health application in UML
Figure 1 shows that a new association class must be created for each association with spatial or temporal properties. As can be seen, this leads to the creation of a host of artificial constructs that significantly complicate the schema diagram. Furthermore, there is no single, easily visible notation to represent spatiotemporal properties. This violates the requirement that the notation be simple, clear, and consistent. A better approach is to extend the fundamental characteristics of the existing UML elements to directly support spatiotemporal requirements. This would involve changes to the structure of instantiated UML elements (i.e., object, association, and attribute instances) to provide for associated time periods or spatial extents. Extension mechanisms defined within UML to extend core element semantics (stereotypes, tagged values, constraints) do not support the structural changes in element instances required to model spatiotemporal semantics. Therefore, although these extension mechanisms may be used as a guide, a strict adherence to their UML definitions in defining a spatiotemporal UML extension is not desirable. Alternative methods for extending a data model and UML are discussed in detail in Price et al. (2000a). In the next section, we describe a UML extension, XSTUML, that is based on defining a small set of orthogonal constructs and consistent rules for combining them.

**XSTUML**

The proposed extension to UML is based on the addition of five new symbols, illustrated in Figure 2, and a specification box describing the detailed semantics of the spatiotemporal data represented using the five symbols. The basic approach is to extend UML by adding a minimal set of constructs for spatial, temporal, and thematic data, represented respectively by *spatial*, *temporal*, and *thematic* symbols. These constructs can then be applied at different levels of the UML class diagram and in different combinations to add spatiotemporal semantics to a UML model element. In addition, the *group* symbol is used to group attributes with common spatiotemporal properties or inter-attribute constraints and the *existence-dependent* symbol is used to describe attributes and associations dependent on object existence.

As discussed previously, although these new symbols can be roughly described as stereotypes, they do not adhere strictly to the UML definition. For improved readability, we use the alternative graphical notation for stereotypes described by Rumbaugh et al. (1999, pp. 451). These symbols can be annotated with a unique label used to reference the associated specification box. The first four symbols can optionally be used without the abbreviations shown in the figure (i.e., S, T, Th, and G, respectively). The specific alphanumeric domain can be optionally indicated, e.g., Th: int.

The *group* symbol, *existence-dependent* symbol, and *specification box* are discussed in the next section. The *spatial*, *temporal*, and *thematic* symbols are described in this section.

---

*Figure 2: XSTUML symbols*
which first provides a general overview of the meaning and use of these three symbols. The section subsequently explains the associated semantics of these symbols at the attribute (and attribute group), object class, and association levels.

**Spatial, Temporal, and Thematic Constructs**

These constructs can be used to model spatial extents, object existence or transaction time, and the three different types of spatiotemporal data previously discussed (i.e., temporal changes in spatial extents; changes in the values of thematic data across time or space; and composite data whose components vary depending on time or location). To understand the use and semantics of the spatial, temporal, and thematic constructs, we first discuss the interpretation of each individual symbol separately.

The *spatial* symbol represents a spatial extent, which consists of an arbitrary set of points, lines, regions, or volumes. The spatial extent may be associated with thematic or composite data, or may be used to define an attribute domain. The *temporal* symbol represents a temporal extent, or timestamp, which may be associated with thematic, spatial, or composite data. Timestamps may represent existence time for objects, valid time for associations and attributes, and transaction time for objects, associations, and attributes. The *thematic* symbol represents thematic data.

The *thematic* symbol can only be used at the attribute level and only in conjunction with one of the other two symbols to describe an attribute with temporal or spatial properties. A thematic attribute domain with no spatial or temporal properties uses standard UML notation, i.e., `<attribute-name>: <domain>`. When there are such properties, either this notation can be used or the specific thematic domain can be indicated inside the thematic symbol. Figure 3 illustrates the four possible cases for a thematic attribute: attributes with a thematic domain and (a) no spatial or temporal properties, (b) temporal properties, (c) spatial properties, or (d) spatiotemporal properties. Adjectives are used to describe the attribute domain (e.g., thematic attribute) and adverbs with the word dependent to describe additional attribute properties for composite attribute domains (e.g., temporally dependent thematic attribute). Therefore, the four possible cases for thematic attributes are called (a) thematic, (b) temporally dependent thematic, (c) spatially dependent thematic, or (d) spatiotemporally dependent thematic attributes, respectively.

The semantics of XSTUML depend on three factors: (a) the symbol used, (b) the model element described by the symbol (i.e., object, association, or attribute), and (c) whether the symbol is combined with other symbols. The general rules for combining symbols can be summarized as follows:

- **Nesting one symbol inside another** mathematically represents a function from the domain represented by the inner symbol to the domain represented by the outer symbol.

**Figure 3: Thematic attribute examples**

(a) populationDensity: int  
(b) populationDensity: Th: int  
(c) populationDensity: Th: int  
(d) populationDensity: Th: int
Therefore, different orders of nesting symbols correspond to different functional expressions and represent different perspectives of the data. For example, Figure 3(b) represents a function from the time to the integer domain for a given object or association instance. If we reverse the order of the symbol nesting, this would represent the inverse function from the integer to the time domain. However, from the conceptual design and schema perspective, both represent the same semantic modeling category and would result in the same conceptual and logical schema, i.e., a temporally dependent, thematic attribute. Users select the order of nesting that best matches their perspective.

Note also that in Figure 3(b), only one integer value is associated with each timestamp; however, several different timestamps may be associated with the same integer value. In Figure 3(d), several integer values will be associated with each timestamp, one for each spatial location.

- Placing one symbol next to another symbol mathematically represents two separate functions, one for each symbol. The order in which the two symbols are written is not significant.

We now give the rule for which symbolic combinations are legal at each model level, the semantic modeling constructs defined at each level, and a mapping between the two. The textual definition, symbolic representation, and mathematical definition is given for each semantic modeling construct for the symbol nesting used in this chapter’s figures. However, as discussed previously, any other nesting order for the same semantic modeling construct is allowed, described in detail by Price et al. (2000a).

Note that any reference to a timestamp, timestamps, a time point, or time validity in the definitions for a given symbol nesting could be for any time dimension, i.e., transaction and/or either valid (for attributes and associations) or existence (for objects) time dimensions. We first summarize the primitives used in this section to denote various time, space, and model elements.

\[
\begin{align*}
&T ::= \text{domain of time instants} \\
&T^2 ::= \text{arbitrary set of time instants, i.e., a timestamp or set of timestamps} \\
&S ::= \text{domain of points in space} \\
&S^2 ::= \text{arbitrary set of points in space, i.e., a spatial extent or set of spatial extents} \\
<oid> ::= \text{domain of object-identifiers} \\
<aid> ::= \text{domain of association-instance identifiers, essentially } \{ <oid> \}^n \\
<id> ::= \text{domain of object and association identifiers, essentially } \{ <oid> | <aid> \} \\
&D ::= \text{thematic, i.e., alphanumeric, domain (e.g., integer, string)} \\
<d> ::= \text{thematic attribute symbol} \\
<t> ::= \text{temporal symbol} \\
<s> ::= \text{spatial symbol} \\
<s&d> ::= \text{any nested combination of a spatial and a thematic symbol} \\
<td> ::= \text{any nested combination of a temporal and a thematic symbol} \\
<s&t>d ::= \text{any nested combination of a spatial, a temporal, and a thematic symbol} \\
<ED> ::= \text{existence-dependent symbol}
\end{align*}
\]

The Attribute (and Attribute Group) Level

At the attribute level, we can model *temporal changes in spatial extents*, where the spatial extent represents a property of an object (i.e., spatial attribute), and *changes in the*
value of thematic data across time and/or space (i.e., spatially and/or temporally dependent thematic attributes).

Legal combinations of symbols at the attribute level are any nested combination of a spatial symbol, a temporal symbol, and/or a thematic symbol. The only exception is that the temporal symbol cannot be used alone. An attribute with a temporal domain is treated as thematic data since temporal data types are pre-defined for popular standard query languages such as SQL. The attribute domain can optionally be followed by an existence-dependent symbol (discussed in the next section). The rule for notation at this level can be defined using BNF notation and the primitives defined previously: \( \text{attribName}: [ <D> | <s&d> | <t&d> | <s&t&d> | <s> | <s&t> ] [<ED>] \)

Six different attribute domains are possible, corresponding to the semantic categories of attributes (i.e., modeling constructs) shown below. Note that each one of the definitions below applies to the identified object or association instance; therefore, we do not state this explicitly in the definitions.

- **Thematic Attribute**: This is an attribute with thematic values.

  \[ f: <id> \xrightarrow{<D>} \]

  Returns the thematic attribute value.

- **Spatially Dependent Thematic Attribute**: This is a set of thematic attribute values, each associated with a spatial extent representing the location where that attribute value is valid. This implies that the attribute values may change over space and their changed values may be retained.

  \[ f: <id> \xrightarrow{(S \xrightarrow{<D>})} \]

  Returns a set of spatial points, each with its associated thematic attribute value (valid for that spatial point).

- **Temporally dependent Thematic Attribute**: This is a set of thematic attribute values, each associated with one or more timestamps, representing the attribute value’s valid and/or transaction time. This implies that the attribute values may change over time and their changed values may be retained.

  \[ f: <id> \xrightarrow{(T \xrightarrow{<D>})} \]

  Returns a set of time points, each with its associated thematic attribute value (i.e., valid for that time point).

- **Spatiotemporally Dependent Thematic Attribute**: This is a combination of spatially and temporally dependent thematic attributes as defined above, i.e., a set of thematic attribute values, each associated with a spatial extent and one or more timestamps.

  \[ f: <id> \xrightarrow{(T \xrightarrow{(S \xrightarrow{<D>})})} \]

  Returns a set of time points, each with its associated set of spatial points, and, for each spatial point, its associated thematic attribute value (i.e., valid for that time and spatial point).

- **Spatial Attribute**: This is an attribute with a spatial domain, i.e., the attribute value is a spatial extent.

  \[ f: <id> \xrightarrow{<2s>} \]

  Returns the spatial attribute value.
• Temporally Dependent Spatial Attribute: A spatial attribute is associated with one or more timestamps, representing the spatial extent’s valid and/or transaction time.

\[
f: \langle \text{id} \rangle \rightarrow (\langle \text{T} \rangle \rightarrow \langle 2^\circ \rangle)
\]

Returns a set of time points, each with its associated spatial attribute value (i.e., spatial extent).

The use of these symbols at the attribute level is illustrated in Figure 4. The difference between (a) thematic attributes, (b) temporally dependent thematic attributes, (c) spatiotemporally dependent thematic attributes, (d) spatial attributes, and (e) temporally dependent spatial attributes is illustrated by (a) name (for Hospital and Province), (b) numBeds, (c) populationDensity, (d) location, and (e) halfHourZone, respectively.

A thematic attribute domain is indicated as a string after the attribute or—if that attribute also has temporal or spatial properties—by the use of a thematic symbol. If no domain is explicitly specified for an attribute, then the use of the spatial symbol indicates that the attribute has a spatial domain. Thus, the Hospital location and halfHourZone attributes represent spatial data. The nested temporal symbol used for halfHourZone indicates that the spatial extent associated with this attribute may change over time and thus should be timestamped. Therefore, an attribute marked by a spatiotemporal symbol (and no thematic domain) represents a spatial extent that changes over time. In this case, as transport networks change, the geometry of the half-hour travel zone must be updated.

In contrast, an attribute that has a thematic domain and a spatial and/or temporal symbol represents a spatially and/or temporally dependent thematic attribute. This is indicated graphically by using the thematic symbol; thus this symbol is used to differentiate two different types of spatiotemporal data: temporal changes in spatial extents and changes in the value of thematic data across time and space. Therefore, the fact that numBeds has an integer domain associated with a temporal symbol indicates that the integer value of numBeds may change over time and should be timestamped. Analogously, the integer value of populationDensity may change over time or space and thus each value is associated with a timestamp and spatial extent.

Figure 4: Using XSTUML at the attribute level
The Object-Class Level

At the object-class level, we can model *temporal changes in spatial extents*, where the spatial extent is associated with an object instance. We can also model the time an object exists in the real world (i.e., existence time) or is part of the current database state (i.e., transaction time).

An object class can be marked by a temporal symbol, a spatial symbol, or any nested combination of these. In addition, this is the only level where the symbols can be paired; i.e., a temporal symbol can be paired with either a spatial symbol or a nested combination of the two symbols. The separate temporal symbol represents the existence or transaction time of the object. The spatial symbol represents the spatial extent associated with that object. If the spatial symbol is combined with a nested temporal symbol, then the spatial extent is timestamped to show the valid or transaction time of the spatial extent. Since the object can exist or be current even when not actually associated with a spatial extent, separate timestamps are required for the object instance and for the object instance’s spatial extent. The rule for object level notation can be given in BNF as follows: `className [ <s> | <s&t> ] [ <t> ].`

The five instantiations of this rule, i.e., categories, (`<s>; `<s&t>; `<t>; `<s><t>; and `<s&t><t>`) are given below.

- **Spatial Object (Class):** An object is associated with a spatial extent. This is equivalent to an object having a single spatial attribute except that there is no separate identifier for the spatial extent.

  \[ f: \text{<oid>} \rightarrow \text{<2S>} \]

  Returns the spatial extent of the identified object.

- **Temporally Dependent Spatial Object (Class):** The spatial extent associated with a spatial object is also associated with one or more timestamps, representing the spatial extent’s valid and/or transaction time.

  \[ f: \text{<oid>} \rightarrow (\text{<T>} \rightarrow \text{<2S>}) \]

  Returns a set of timepoints, each associated with the spatial extent of the identified object at that timepoint.

- **Temporal Object (Class):** An object is associated with one or more timestamps, representing the object’s existence and/or transaction time.

  \[ f: \text{<oid>} \rightarrow \text{<2T>} \]

  Returns the timestamp of the identified object.

- **Spatiotemporal Object (Class):** This is a combination of a spatial and temporal object as defined above, i.e., each object instance is associated with a spatial extent and one or more timestamps representing the object’s existence and/or transaction time.

  \[ \bigcirc \bigcirc f: \text{<oid>} \rightarrow \text{<2T>} \text{ and } f: \text{<oid>} \rightarrow \text{<2S>} \]

  Returns the timestamp and the spatial extent of the identified object.

- **Temporally Dependent Spatiotemporal Object (Class):** This is a combination of a temporally dependent spatial object and a temporal object as defined above, i.e., an
object is associated with a spatial extent, one or more timestamps representing the spatial extent’s valid and/or transaction time, and one or more timestamps representing the object’s existence and/or transaction time.

\[ \triangledown \otimes f: \text{oid} \rightarrow <2T> \text{ and } f: \text{oid} \rightarrow (<T> \rightarrow <2S>) \]

Returns the timestamp of the identified object and a set of timepoints, each with its associated spatial extent (i.e., valid at that timepoint), for the identified object.

The use of symbols at the object-class level is illustrated in Figure 5. In Figure 5(a), the temporal symbol at the Hospital object level represents a temporal object class with existence and transaction time (see the next section). In Figure 5(b), we give an example of a temporally dependent spatial object. This example assumes that there is no need to represent hospital location separately from the half-hour travel zone. Instead, a hospital object is treated as a spatial object with a single associated spatial extent, showing the half-hour travel zone around that hospital. The temporal symbol indicates that the spatial extent should be timestamped, since the half-hour travel zone can change over time. Finally, Figure 5(c) combines (a) and (b), illustrating a temporally dependent spatiotemporal object. The object is \textit{spatiotemporal} because it has a timestamp and a spatial extent; and it is \textit{temporally dependent} because the spatial extent also has a timestamp.

**The Association Level**

At the association level, we can model \textit{temporal changes in spatial extents}, where the spatial extent is associated with a relationship between object instances (i.e., spatiotemporal association), and \textit{composite data whose components vary depending on time or location} (i.e., spatiotemporal aggregation or composition). The following discussion applies to any type of association, including aggregation and composition.

At the association level, any nested combination of a spatial and/or a temporal symbol represents a legal combination describing spatiotemporal properties of the association. Except for the omission of the \textit{thematic} symbol, the association level is similar to the attribute level. The association spatiotemporal properties can optionally be followed by an \textit{existence-dependent} symbol (discussed in the next section). The rule for the association-level notation can be given in BNF as follows:

\[ \text{assoc-line} \ [ s | t | s\&t ] [ \text{ED} ] \]

\textit{Figure 5: Using XSTUML at the object-class level}

```plaintext
(a) Hospital
  name: string
  numBeds: Th: int
  location: S
  halfHourZone: S
  operations

(b) Hospital
  name: string
  numBeds: Th: int
  operations

(c) Hospital
  name: string
  numBeds: Th: int
  operations
```
Reading the BNF rule from left to right, three different categories of associations are possible, as defined below.

- **Spatially Dependent Association:** An association instance is associated with a spatial extent representing the location where the association instance is valid. This implies that the association instances may change over space and their changed instances may be retained.

  \[ f: \langle \text{aid} \rangle \rightarrow \langle \text{2S} \rangle \]

  Returns the spatial extent of the identified association.

- **Temporally Dependent Association:** An association instance is associated with one or more timestamps, representing the association’s valid and/or transaction time. This implies that association instances may change over time and the changed instances may be retained.

  \[ \bigtriangledown f: \langle \text{aid} \rangle \rightarrow \langle \text{2T} \rangle \]

  Returns the timestamp of the identified association.

- **Spatiotemporally Dependent Association:** This is a combination of spatially and temporally dependent associations as defined above, i.e., an association is associated with a spatial extent and one or more timestamps.

  \[ \bigtriangledown f: \langle \text{aid} \rangle \rightarrow (\langle \text{T} \rangle \rightarrow \langle \text{2S} \rangle ) \]

  Returns a set of time points, each with the associated spatial extent for the identified association at that time point.

  The use of these symbols at the association level is shown in Figure 6. Marking the Is-of association with a temporal symbol signifies that the category of a hospital may change over time, as local health needs change and wards are opened or closed. Therefore, association instances should be timestamped.

  A spatially dependent association is one where an association instance is associated with a spatial extent to show where that instance is valid. For example, the same category of hospital may require different categories of wards in different areas depending on local regulations. Therefore, the Contains aggregation association must be spatially dependent. In fact, since the local categories may also change over time, the Contains aggregation association is actually spatiotemporally dependent. In this case, both of the associated object classes are purely conceptual. An association between two physical object classes can also be spatiotemporally dependent; e.g., a consultation of a ward doctor with a specialist is scheduled for a specific location and period of time in the hospital.

  Since constraints between timestamps or spatial extents of objects participating in a temporally and/or spatially dependent association are application dependent (see Price et al. (2000a) for a detailed discussion), they are not incorporated as implicit defaults in XSTUML but should instead be specified explicitly. This can be done either on an ad hoc basis as required using UML constraints or by defining explicit modeling constructs for commonly used constraint patterns. The latter approach is illustrated by the introduction of the existence-dependent construct (see the next section) to support the semantics of temporal dependency and by the introduction of constructs for spatial PW relationships and their topological constraints (see the penultimate section) to support the semantics of spatial dependency.
The previous section described the different types of timestamps that can be associated with an attribute, association, or object class; but where do we specify which types are required for a given application? Detailed spatiotemporal semantics are specified in a specification box, which can be associated with any of the icons or combinations using a unique naming convention (used in this chapter) or label. The specification box includes information on the time units and the time and space dimensions, models, and interpolation. Users can specify regular (recorded at regular intervals) or irregular time models and object- or field-based space models. Interpolation functions can be specified to derive values between recorded spatial locations or timestamps for spatially and/or temporally dependent thematic attributes. The time dimensions and units (i.e., instant, interval, element) used are defined by Jensen and Dyreson (1998). Specification boxes can be inherited from parent classes as with any other class property. The specification box syntax is illustrated in Figure 7.

Time dimensions include existence time (for objects), valid time (for attributes and associations), and transaction time (for objects, attributes, or associations), as defined by Jensen and Dyreson (1998). However, object existence time is more precisely defined as the time during which existence-dependent attributes and associations can be defined (i.e., have legal values) for that object. In other words, existence-dependent attributes and associations are those that are defined only when the related object(s) exist. This implies that attributes and associations that are not existence-dependent (e.g., an employee’s social security number) may be defined even when the related object(s) no longer exist. Other attributes, e.g., work phone number, are defined only while the related object(s) exist (e.g., the employee works at the company) and are therefore existence-dependent.
Note that existence time is not necessarily equivalent to biological lifespan. The exact meaning will be dependent on the application; therefore, individual applications define which attributes and associations are existence-dependent. Object identifiers are never existence-dependent, as they can be used to refer to historical objects. Any other attribute or association can be defined as being existence-dependent.

If existence time is associated with a given object, the existence-dependent attributes and associations for that object class must be explicitly marked as such by adding the superscript $ED$ to the attribute or association name. Conversely, existence-dependent attributes and associations can only be defined for objects having existence time specified. In the case of an existence-dependent association, existence time must be defined for at least one of the participating objects. If an existence-dependent attribute or association is temporally dependent, then its valid-time must be included within the existence time of the corresponding object instance; or—for associations—the intersection of the existence times for those participating object instances that have existence time defined.

Note that the time model and interpolation specification apply only to valid time, whereas the time unit specification is used both for valid or existence time and transaction time. Therefore, the dimension must be specified for time unit whenever a model element is associated with both valid or existence time and transaction time. In addition, time interpolation is normally used for temporally dependent thematic attributes. Time interpolation of spatial attributes (i.e., spatial extents) must be discrete (i.e., no interpolation) or user defined.

Space dimensions include the dimensions of the spatial extent(s) being specified, followed by the dimensions of the underlying search space. The object-based spatial model is used for a spatial attribute, i.e., the attribute instance for a single object instance consists of a single spatial extent. The field-based spatial model is used for a spatially dependent, thematic attribute; where a single object instance has a set of thematic values, each associated with a different spatial extent. Space interpolation applies only to spatially dependent thematic attributes using the field-based spatial model.

The specification box can also be used to specify spatiotemporal constraints, including constraints within an attribute group. The group symbol is used to group attributes sharing the same timestamps or spatial extents, that then only need to be specified once for the group.
Thus, the group symbol graphically illustrates associated sets of attributes and avoids the possibility of redundantly specifying the same spatial extents and timestamps. Note that a group’s attributes never share thematic values, even if the thematic symbol is used in the group specification. If the group’s attributes have different thematic domains, then these can be indicated next to each attribute using standard UML text notation. Additional constraints for the construct associated with the specification box can be associated after the group constraints.

Following UML convention, another compartment is added to the object class to accommodate the specification boxes for that class, i.e., the specification compartment. The specification compartment can be used to specify spatiotemporal semantics for the object, the attributes of the object class, and any associations in which the object class participates. Alternatively, a specification compartment can be added to an association class to specify spatiotemporal semantics for that association and its attributes. A detailed discussion of the specification compartment and box has been provided by Price, Srinivasan, and Ramamohanarao (1999).

USING XSTUML: THE REGIONAL HEALTH CARE EXAMPLE

Figure 8 shows the full regional health application, described earlier, as it would be represented using the proposed extension and illustrates the use of the specification box, group symbol, and existence-dependent symbol.

For example, Hospital location is specified as a single point in 2D space. Hospital halfHourZone and Contains are specified as a region in 2D space. In contrast, the Province populationDensity and averageLifespan group is associated with a 2D field in 2D space. This means that, for a single object instance, the two attributes in the group are associated with a set of regions and have a separate attribute value for each region for a given point in time. Since these two attributes share common timestamps and spatial extents, they are grouped. Since both attributes are integers, we can specify the thematic domain in the group symbol. If the attributes had different thematic domains, then we would specify them for each attribute rather than for the group.

The group is then associated with a single symbol and specification box. Here we specify that any attribute in the group uses average interpolation in time and no interpolation in space, has a valid time dimension using instant as the time unit, and is measured yearly (i.e., a new set of values is recorded for the attribute each year). This means that the population density and average lifespan between recorded time instants is assumed to be the average of the values at the two nearest time instants and undefined outside of recorded spatial regions. No inter-attribute constraints are defined for the group, as shown by the keyword independent.

The temporal symbol at the Hospital object level is used to indicate existence time and transaction time. Existence time is used to model the periods when the hospital is open, i.e., when the existence-dependent attributes numBeds and halfHourZone and the existence-dependent association Is-of are defined. Since these model elements are temporally dependent, the valid timestamps of all their instances must be included within the Hospital existence time. Attribute numBeds is specified as irregular because this attribute is not recorded periodically; whenever it changes, the new value is recorded.
The specification box for an association (e.g., \textit{Is-of}) can be placed in the specification compartment of either of its participating object classes (e.g., \textit{Hospital} or \textit{Hospital-category}). Note that since \textit{Hospital-category} is not temporal and therefore does not have existence time defined, the only constraint on the valid-time timestamps of the \textit{Is-of} association comes from the \textit{Hospital class} existence time.
Comparing the schemas of the regional health application from Figure 8 and Figure 1, it is clear that the schema that uses XSTUML is much simpler than the corresponding UML schema. The use of UML in Figure 1 results in the creation of a host of artificial constructs to represent spatiotemporal semantics, obscuring the schema design. Far fewer object classes and attributes are required in Figure 8, since graphical symbols and specification boxes were used instead of extra associations or attributes (e.g., for time dimensions or identification) to provide a compact, distinct, and consistent representation of spatiotemporal properties.

By incorporating spatiotemporal semantics in the modeling language itself, XSTUML reduces the complexity of the resulting schemas. The level of detail is reduced without sacrificing understandability. This allows the application developer to concentrate on the characteristics of the specific application domain of interest. The modular specification of spatiotemporal properties also facilitates schema reuse and extension.

For example, if we want to reuse the schema from Figure 8 for the same application but without historical records, we can simply delete all of the temporal symbols and specifications. Similarly, if hospital definitions do not vary regionally, one need only remove the spatial symbol from the Contains icon and specification box. In contrast, the modifications required to reuse the schema from Figure 1 are not nearly so obvious or modular. Each schema element would have to be examined to determine which model elements would need to be modified or deleted.

The specification box aids readability by providing a clear and consistent framework for the detailed specification of spatiotemporal semantics. These semantics are represented in UML using constraints and notes that are unlikely to be standardized among users, making the diagram more difficult to read. The specification box can serve as a guideline for application developers, highlighting generally relevant semantics to be considered when modeling spatiotemporal data. This facilitates effective communication and consistent design documentation.

**MODELING COMPLEX SPATIAL OBJECTS AND THEIR TOPOLOGICAL CONSTRAINTS**

One of the characteristics of spatial (and spatiotemporal) applications is the need to record and manage complex relationships between spatial objects, objects with associated spatial extents. Furthermore, individual spatial objects may be simple (connected) or composite (consisting of disjoint spatial components) and be regular or irregular (having holes, crossings, punctures, or other discontinuities). Examples are a supranational organization formed from member countries, the division of an administrative region into voting districts, or the structures erected on a building site during construction, where a single country or voting district may consist of several disjoint sub-units and/or have discontinuities such as holes. Such complex relationships typically involve an asymmetric relationship between spatial objects, where one object—the whole—is regarded as representative in some way of a group of other objects—the parts. We refer to this related group of spatial objects as spatial part-whole (PW) relationships, described in detail by Price et al. (2000b).

We classify spatial PW relationships based on whether the spatial extent of the whole object is derived from or constraining those of its parts, termed spatial derivation and spatial constraint and illustrated respectively by a supranational organization and a building site. Spatial PW relationships are defined in terms of their primary mathematical, dependence-
related, structural, and spatial characteristics. The relationship between the different categories of spatial PW relationships and their defining, primary properties are shown in Figure 9.

To describe topological constraints in complex objects during application analysis and design, a classification scheme and modeling constructs were proposed by Price et al. (2001) for binary and n-ary (set-based) topological relationships. The first level of classification for binary relationships is based only on whether the intersection and difference of two non-empty spatial extents $A$ and $B$ are empty or non-empty. Essentially, non-intersecting spatial extents (the intersection is the empty set) are disjoint and intersecting spatial extents (the intersection is non-empty) have the topological relationship equal (the difference in both directions is the empty set), contain (the forward difference $A-B$ is non-empty and the reverse difference $B-A$ is the empty-set), inside (the forward difference is the empty-set and the reverse difference is non-empty), or connected (the difference in both directions is non-empty). So, for example, the equal and contain relationships can be used to model the voting district and building site examples, respectively. The two non-symmetrical relationships, contain and inside (i.e., contained-by), can be combined through disjunction into one symmetric nested relationship where either the forward difference or the reverse difference—but not both—is the empty set. The connected and disjoint categories have a further level of classification defined when more refinement of the model is required for a specific application.

Connected objects can be further classified based on whether they have a boundary, interior, or mixed overlap, i.e., whether their intersection includes only boundary, only interior, or both boundary and interior points. Since boundary and interior points often represent semantic differences in applications, it is useful to be able to specify whether the intersection involves object boundaries, interiors, or both. For example, in the case of voting districts for a given administrative region, interior points are used to represent administrative jurisdiction and boundary points are used to represent a change in jurisdiction. A crucial aspect of the connected sub-categories is that, in contrast to other proposed topological

---

**Figure 9: Primary and derived characteristics for spatial part-whole relationships**
classification schemes, the only assumption is that every point in a spatial extent must be either a boundary or interior point but not both. Further definition is left to the user as appropriate to specific application requirements.

A disjoint relationship between two spatial entities can be further distinguished based on whether their spatial extents inter-penetrate, i.e., whether the minimum bounding figures of the two objects intersect. As with the definition of boundary and interior used for the sub-categories of connected, the decision as to exactly how to determine the minimum bounding figure is left to the user. Separate is a disjoint relationship where the minimum bounding figures of the two spatial extents do not intersect and interpenetrating is a disjoint relationship where they do intersect.

The eight categories—separate, interpenetrating, equal, contain, inside, boundary-overlap, mixed-overlap, and interior-overlap—represent a complete and mutually exclusive set of binary topological relationships as shown by Price et al. (2001). The more general categories—nested, connected, disjoint, and intersecting—can be derived from these eight relationships. The binary classification scheme can then be used to illustrate the spatial constraint relationship sub-types shown in Figure 9. Spatial cover (SCE) is exemplified by a guaranteed phone service coverage area that must be completely contained by or equal to the part-union of the phone service cells. A building site and the structures on that building site represent an example of spatial inclusion (SIE), since the part-union of the structures must be inside or equal to the building site area. The stricter constraint of spatial interior (SI) applies to house furnishings (referring here to appliances and furniture), since the furnishings must be inside but cannot completely equal the area of the house in order to ensure walking room. Finally, the part-union of taxi dispatch zones (the area over which a given taxi driver ranges) must be exactly equal to the metropolitan area covered by the taxi company, i.e., spatial equal (SE). This ensures complete coverage of the metropolitan area without risking cases where the company insurance policy may not be applicable.

The binary topological classification described in the last section is sufficient to describe topological constraints between a whole and its part-union. However, complex spatial objects are also characterized by topological constraints between the parts. For example, the voting districts created for an administrative region cannot have overlapping interiors, as this would lead to the possibility of a single constituent being able to vote in more than one district. Similarly, member countries of a supranational organization cannot share interior points. N-ary (set-based) topological relationships are required to describe such topological constraints between the parts. Given some binary topological relationship $R$ defined for two spatial objects, we extend this to $n$ spatial objects by defining modeling constructs based on whether $R$ holds for every pair (all), at least one pair (some), or no pair (none) of the spatial objects. One additional modeling construct is defined based on a special case of some, where any two spatial extents in the set can be related directly or indirectly by the given topological expression (linked). We first describe the notation used for the spatial extents and their topological relationships, and then give the formal definitions for the n-ary modeling constructs.

Let $O = \{ o_1, ..., o_i, ..., o_j, ..., o_n \}$ a finite set of $n$ spatial extents, where $n \geq 2$ and $i \neq j$.

Let $R$ a topological expression, e.g., one (or a disjunction or conjunction) of the binary topological relationships described earlier.

Let $S \subseteq O$ (a non-empty sub-set of $O$) $= \{ s_1, ..., s_k, ..., s_p \}$ a set of $p$ spatial extents, where $p > 1$ and $p < n-2$.

The following modeling constructs are defined for describing n-ary topological rela-
tionships between two or more spatial extents. (By definition, the constructs evaluate to TRUE for zero or one spatial extent(s).)

\[
\text{all}(R, O) \equiv A o_i, o_j \in O, (o_i R o_j)
\]

\[
\text{some}(R, O) \equiv \exists o_i, o_j \in O, (o_i R o_j)
\]

\[
\text{none}(R, O) \equiv \neg \exists o_i, o_j \in O, (o_i R o_j)
\]

\[
\text{linked}(R, O) \equiv A o_i, o_j \in O, ((o_i R o_j) \lor (\exists S, ( (o_i R s_1) \land \ldots \land (s_{k-1} R s_k) \land \ldots \land (s_p R o_j) ) ) )
\]

We can then describe the non-overlapping constraint on voting districts or countries, using the modeling construct \(\text{all}(\text{boundary-overlap or disjoint})\) to specify that interior points cannot be shared between parts in a spatial PW relationship. In a similar manner, the \(\text{linked}(\text{mixed-overlap})\) constraint can be used to specify that phone service cells must overlap to ensure continuous phone coverage.

These modeling constructs allow specification of general topological relationships between the spatial extents—whether simple or composite—of \(n\) spatial objects. However, there may be some cases where we want to treat a set of composite spatial extents as a set of their individual components and to model topological constraints between these individual components; for example, to ensure a continuous national road network when individual roads may have disconnected components (e.g., a long-distance road with segments that belong to a freeway interspersed with local roads), we need to evaluate topological relationships between road segments rather than roads in the national road network. To do this, we need to define an additional modeling construct that decomposes a set of spatial extents into the set of all their individual components. That is, given a set \(O\) of \(m\) composite spatial extents \(o_1, \ldots, o_i, \ldots, o_m\) with \(n_1, \ldots, n_i, \ldots, n_m\) components respectively and where \(c_{ik}\) is the \(k\)th component of the \(i\)th composite spatial extent \(o_i\), we have the following:

\[
decompose(O) \equiv \{ \ldots, c_{ik}, \ldots \} \text{ where } i=m \text{ and } k=n_i
\]

We can then use any of the previously defined constructs for \(n\)-ary topological relationships, replacing \(O\) with \(\text{decompose}(O)\). So in the case of the national road network, the \(\text{decompose}\) operator is used to refer to individual road segments. The \(\text{connected}\) binary topological operator discussed earlier is used to compare pairs of road segments. The \(\text{linked}\) relation is then used to specify that it must be possible to find a finite sequence of connected pairs linking any two road segments. Assuming that we have the set of roads \(r_1, \ldots, r_n\) in the road network, this constraint would be formally specified as: \(\text{linked}(\text{connected}, \text{decompose}(\{r_1, \ldots, r_n\}))\)

Figure 10 illustrates how support for modeling spatial PW relationships and their topology is integrated into XSTUML, using the example of a supranational organization with headquarters and member countries (each having an associated 2D spatial extent). The example assumes that all timestamps are real-world (i.e., valid) time intervals and that population density for countries is recorded yearly. We incorporate spatial PW relationships in XSTUML by introducing a new type of association, represented by a circle at one end and an abbreviation inside the circle to indicate the type of spatial PW relationship. All of the standard UML and XSTUML notations and rules apply. For example, XSTUML temporal associations can be used to define time-dependent spatial PW relationships. As with any other XSTUML symbol, an identification label can be included in the circle and used to refer to a specification box giving further details of the spatial PW relationship’s semantics. For instance, the specification box is used to specify additional topological constraints on the parts in the spatial PW relationship. The constraint that member countries not overlap can be specified by using the modeling constructs for \(n\)-ary topological constraints (e.g.,
all\{boundary-overlap or disjoint\} . This specification box is located in the specification compartment of the whole object type. Since more than one spatial extent can be associated with an object in XSTUML (e.g., a Supranational Organization can have a location and headquarters), the spatial PW relationship always refers to the spatial extent directly associated at the object level with each whole and part object. Therefore, every object type in a spatial PW relationship must have a spatial extent modeled at the object level.

A down arrow is used to indicate any attribute of a part object, represented in Figure 10 by the generic attribute someAttribute, whose value propagates from the whole object attribute of the same name and domain. For example, a European Union passport is also used as the passport for each member country. Note that this allows the individual specification of propagation in terms of the particular attribute and part object type. Similarly, any attribute of the whole object that is derived from the values of part objects (e.g., average population density is derived from member countries’ population densities) is indicated graphically with a labeled up arrow. The derivation formula is specified in the corresponding specification box in the whole object’s specification compartment. Alternatively, we can use an operation that references the attributes of the part objects instead of a derived attribute.

The SM relationship is used to show derivation of the supranational organization’s spatial extent. Note that the spatial extent of the whole object in a spatial derivation relationship is always derived from that of its part objects. This is already implied by the use of an SP or SM spatial PW relationship; therefore, there is no need for any additional notation at the object level. It therefore follows that if the SP or SM relationship and parts’ spatial extents are timestamped, then timestamps for the whole’s spatial extent can be derived (i.e., from those of its members’ spatial extents for the countries participating in the SP or SM

Figure 10: Supranational organization with spatial part-whole relationships
relationship at any given time). For example, if we know when certain countries were members of the EU and when their spatial extents changed during that time, then we can determine the complete history of the change in the EU’s spatial extent over time. The same applies to derived attributes of the whole that are timestamped.

In this example, the spatial PW relationship corresponds to a separately and clearly defined modeling construct with an associated graphical notation. The use of explicit constructs for spatial PW relationships provides a standard graphical notation to represent their semantics. The use of the SM symbol in Figure 10 is associated with a well-defined set of constraints such as spatial derivation of the whole object’s spatial extent (e.g., the supranational organization’s location) and type asymmetry (e.g., a supranational organization cannot be a member of a country). Propagation or derivation of specific attributes can be clearly and consistently modeled on an individual basis as needed for different applications (e.g., `someAttribute` and `avgPopDensity` attributes, respectively). Without the use of constructs for spatial PW relationships, users must individually specify such information using UML constraints or notes; however, these will not be standardized between users or applications and, without an easily recognized graphical representation, will not be immediately obvious from the schematic diagram.

**CONCLUSIONS**

In summary, this chapter describes a UML extension, XSTUML, that supports applications requiring a range of spatiotemporal models and types. A clean technique is introduced for modeling *composite data whose components vary depending on time or location, temporal changes in spatial extents, and changes in the value of thematic data across time and space*. Alternative models of time and change processes are also supported, as well as valid, transaction, and existence time dimensions. By introducing a small base set of modeling constructs that can be combined and applied at different levels of the UML model (including attribute groups), language clarity and simplicity is maintained without sacrificing expressive power or flexibility.

The introduction of a thematic symbol and formal rules for combining spatial, temporal, and thematic symbols provides a consistent and well-defined notation for representing spatiotemporal semantics. Temporal and spatial associations are treated in a parallel manner, i.e., to describe model structure. Attribute groups are introduced to explicitly model related groups of thematic attributes with common spatial and/or temporal properties.

In addition, existence time is precisely defined based on application-defined dependencies of individual object properties and introduced modeling constructs to reflect these semantics. This allows users to differentiate between those properties that are still defined when the object does not exist (e.g., employee social security number) and other properties that are not (e.g., work phone number).

Finally, explicit modeling constructs are defined to support modeling of spatial PW relationships and their topological constraints and their use in XSTUML is illustrated.
REFERENCES


Chapter XIX

The Role of Use Cases in the UML: A Review and Research Agenda

Brian Dobing
University of Lethbridge, Canada

Jeffrey Parsons
Memorial University of Newfoundland, Canada

A use case is a description of a sequence of actions constituting a complete task or transaction in an application. Use cases were first proposed by Jacobson (1987) and have since been incorporated as one of the key modeling constructs in the UML (Booch, Jacobson, & Rumbaugh, 1999) and the Unified Software Development Process (Jacobson, Booch, & Rumbaugh, 1999). This chapter traces the development of use cases, and identifies a number of problems with both their application and theoretical underpinnings. From an application perspective, the use-case concept is marked by a high degree of variety in the level of abstraction versus implementation detail advocated by various authors. In addition, use cases are promoted as a primary mechanism for identifying objects in an application, even though they focus on processes rather than objects. Moreover, there is an apparent inconsistency between the so-called naturalness of object models and the commonly held view that use cases should be the primary means of communicating and verifying requirements with users. From a theoretical standpoint, the introduction of implementation issues in use cases can be seen as prematurely anchoring the analysis to particular implementation decisions. In addition, the fragmentation of objects across use cases creates conceptual difficulties in developing a comprehensive class diagram from a set of use cases. Moreover, the role of categorization in human thinking suggests that class diagrams may serve directly as a good mechanism for communicating and verifying application requirements with users. We conclude by outlining a framework for further empirical research to resolve issues raised in our analysis.
INTRODUCTION

The Unified Modeling Language, or the UML (Booch, Jacobson, & Rumbaugh, 1999), has rapidly emerged as a standard language and notation for object-oriented modeling in systems development, while the accompanying Unified Software Development Process (Jacobson, Booch, & Rumbaugh, 1999) has recently been developed to provide methodological support for the application of the UML in software development. The adoption of the UML brings focus to object-oriented developers faced with the task of choosing among dozens of proposed approaches to object-oriented analysis and design. In light of this activity, driven primarily by practitioners, it is important from an academic perspective to independently evaluate the capabilities and limitations of the UML and the Unified Process. Such evaluations can contribute to the development of theoretical underpinnings of the UML, to an improvement in its modeling power and usability, and to its appropriate application in systems development projects.

This chapter focuses on two components of the UML: use cases and class diagrams. In particular, we consider whether use cases add value as a component of an object-oriented modeling language by looking at two of their key roles, gathering requirements and developing class diagrams. We examine the variability in the amount of detail use cases should contain, according to various proponents, and introduce a theoretical rationale for including fewer task details than many proponents advocate. We discuss the lack of “object”-orientation in use cases, and present a theoretical argument that use cases may, in fact, not be necessary or valuable in the UML. Finally, we develop a framework for empirical research to evaluate the value of use cases and their relationship to class diagrams in the UML.

USE CASE FUNDAMENTALS

The term “use case” was introduced by Jacobson (1987) to refer to “a complete course of events in the system, seen from a user’s perspective” (Jacobson, Christerson, Jonsson, & Overgaard, 1992, p. 157). The concept resembles others being introduced around the same time. Rumbaugh, Blaha, Premerlani, Eddy, and Lorensen (1991), Wirfs-Brock, Wilkerson, and Wiener (1990), and Rubin and Goldberg (1992) use the terms “scenario” or “script” in a similar way. But, despite concerns about the awkwardness of the name, the use case has become an important part of most object-oriented analysis and design methodologies. Use cases were incorporated into the UML in late 1995, after Ivar Jacobson joined forces with Grady Booch and James Rumbaugh.

The use case differs from typical structured requirements analysis tools that preceded it in two important ways. First, the use case is largely text-based. Structured analysis emphasized the importance of graphical tools, such as Work Flow and Data Flow Diagrams. The rationale for preferring diagrams to text was the oft-cited “a picture is worth a thousand words.” In addition, before structured methodologies became available, analysts often generated extensive and unstructured text descriptions of existing and proposed systems that were very difficult to use. The UML has not abandoned diagrams; there are currently eight in the UML Version 1.3 (OMG, 1999), and the class, activity, sequence, statechart and use-case diagrams all play important roles during analysis. But use cases are written in the customer’s language, so that “users and customers no longer have to learn complex notation” (Jacobson et al., 1999, p. 38).
Second, use cases focus on transactions from the user’s perspective. In Data Flow Diagrams (DFDs), transaction sequences were often not explicitly articulated. All the steps needed, for example, to sell goods to a customer would be there, but the connections between taking orders, checking inventory levels, determining payment types and authorizations, printing receipts, and other activities were not always clear. In addition, DFDs include steps performed internally by the system while use cases leave those steps for the design stage. The focus on complete transactions shares some important similarities with the concept of a “process” in Business Process Reengineering, “a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer” (Hammer & Champy, 1993, p. 35). Both emphasize complete transactions viewed from a customer or user perspective, although the terms “user” and “customer” imply different levels of analysis. Jacobson, Ericsson, and Jacobson (1994) deal extensively with using use cases to support reengineering, suggesting the similarity is not coincidental.

Use cases have been all but universally embraced in object-oriented systems analysis and development books written since Jacobson et al. (1992). There are a few exceptions, but their alternatives still share some common features. For example, Coad (1995) refers to “scenarios” that seem more detailed or lower level than use cases (e.g., a sale calculating its total [p. 61]). Nevertheless, Norman (1996, p. 165) suggests that Jacobson’s use cases and Coad’s scenarios are “similar concepts.” Kilov and Ross (1994, pp. 9-10) use the notion of a “contract” that states “what has to be true before and what will be true after the operation.” Contracts focus more on pre- and post-conditions rather than the steps in between, but again there are similarities.

**USE CASE INTERNAL STRUCTURE**

**Analysis Versus Design Focus**

Despite the strong endorsement of the general use-case concept, there are many variations on Jacobson’s original theme. Not all use cases are created equal. First, there is a difference in content. Use cases, at least during the analysis phase, are generally viewed as a conceptual tool. The use case should emphasize “what” and not “how” (Jacobson et al., 1994, p. 146). This suggests use cases shouldn’t mention technology (e.g., Evans, 1999).

A review of use case examples shows that determining when the “what” ends and the “how” begins is not always easy. Brown (2002) interprets “what” to mean what the system will do rather than the internal implementation. Thus, his use cases include references to screen designs; so do those of Satzinger and Orvik (1996, p. 126). Others (e.g., Korson, 2000) argue that the initial requirements should be interface neutral. Harmon and Watson (1998, p. 121) go further in their example and refer to the salesperson’s laptop. And even Jacobson et al. (1992, p. 162) refer to a display “panel,” “receipt button,” and “printer” in one of their examples. Some use cases also include more detail on business rules. For example, the IBM Object-Oriented Technology Center (1997, p. 489) video store example includes the condition that customers who are not members pay a deposit of $60.00. In contrast, Kulak and Guiney (2000, p. 23, emphasis at source) state that use cases “should show the what exclusively” and their examples seem to follow this philosophy.

However, as Larman (2002, p. 75) notes, use cases are not tied to object-oriented methodologies and thus are technology-independent in that sense. The same cannot be said...
for Data Flow Diagrams, which were designed to produce a basic module structure for a COBOL program. Object-oriented systems can be built without use cases and, conversely, use cases could be used in non-OO projects.

Second, there are several variations that have been proposed for use case formats. Some, such as whether use case titles should begin with gerunds (e.g., “Adding a Customer”) or action verbs (e.g., “Add a Customer”), are not serious. More interesting is the format of the text itself. While the first use cases in Jacobson et al. (1992) were written as a paragraph of text, most others have adopted numbered steps. More recently, Jacobson et al. (1994, p. 109) have done so as well. This may not appear to be a serious issue, but sequenced and numbered steps are an invitation to write about “how.” While the underlying technology need not be mentioned, use cases have become very process-oriented. In most cases, they go much further than simply documenting requirements to providing a suggested solution.

Third, the granularity of use cases also varies from coarse (few use cases) to fine (many). Most take a minimalist approach. Jacobson et al. (1994, p. 105) suggest that use cases should offer “measurable value to an individual actor.” MacMaster (1997) argues that use cases be used only for main system functions. But White (1994, p. 7) states that “the collected use cases specify the complete functionality of the system.” While Dewitz (1996) uses 11 use cases in her video store example, the IBM Object-Oriented Technology Center (1997) has 24. Kulak and Guiney (2000, p. 37) suggest that “most systems would have perhaps 20 to 50 use cases and some small systems even fewer.” But, as they later point out (p. 88), “there are no metrics established to determine correct granularity.” In contrast, Armour and Miller (2001, p. 244) claim that large systems may have hundreds of use cases.

When the number of use cases is larger, they may be organized in some hierarchical fashion. For example, Cockburn (2001, p. 3) uses both “summary level” and “user-goal level” use cases. Armour and Miller (2001) group use cases into packages and suggest several techniques to help do this. Functional decomposition of use cases with “uses” relationships, which Fowler (1998) claims is “the antithesis of object-oriented development,” also occurs in practice.

Fourth, the level of detail within each use case also varies. For example, both Kulak and Guiney (2000, p. 125) and Armour and Miller (2001, p. 125) recommend limiting the length of the flow of events to two pages of text, but the latter also note that some practitioners prefer a few longer use cases to many short ones. Constantine and Lockwood (2000) distinguish between “essential” use cases, containing few if any references to technology and user interface implementation, and “concrete” use cases that specify the actual interactions. Clearly, use cases could move from essential to concrete as the development process proceeds. But not everyone agrees that concrete use cases should ever be used (e.g., Evans, 1999). There are alternative mechanisms that can be used to document screen design choices and similar decisions.

Jacobson et al. (1999) advocate an iterative development approach in which both the number of uses cases and their level of detail increase as the life cycle progresses. They suggest that only the most critical use cases (less than 10%) be detailed in the first (inception) phase. As analysis progresses and requirements become firmer, additional use cases can be added and each can be expanded to include considerably more detail. The analyst could move toward concrete use cases or simply expand the detail within essential use cases. Some authors have become quite specific in describing the different levels. For example, Kulak and Guiney (2000) have identified four levels. However, knowing where to start, how far to go at each phase, and when to stop are clearly critical issues not easily resolved.
To further complicate the issue, some of those who favor fewer or less detailed use cases supplement them with “scenarios.” Booch (1994, p. 158) defines scenarios as examples of what can happen within a use case. “Add a customer” is a use case. Adding a specified customer with a particular name, address, etc., is a scenario. Some references use scenarios to provide further detail on exception handling and other special cases (e.g., customers with missing, improbable, or unusual data [Lockheed Martin, 1996, Bennett, Skelton, & Lunn, 2001]). However, the UML defines a scenario as “a specific sequence of actions that illustrates behaviors” (OMG, 1999, p. 803) and uses “alternative flows” (acceptable, but less common, ways of successfully completing the use case) and “exceptional flows” (for error handling) within the use case to capture this information (p. 224). These paths can also have scenarios. While scenarios are commonly recommended for testing the completed system, they can also be used to test the use cases themselves (Kulak & Guiney, 2000; Armour & Miller, 2001). How many scenarios, alternate paths and exception paths should be developed, and what their role should be in developing class and object diagrams, is not clear. A minimalist approach to use cases combined with extensive scenarios and paths may still result in a large and very detailed set of specifications.

While the general consensus seems to be in favor of a smaller set with relatively brief descriptions, “use case modeling concepts may be applied very informally and loosely or very rigorously and formally” (Armour & Miller, 2001, p. 70). The difficulty is determining when each is appropriate. Stories about organizations mired in hundreds or even thousands (Korson, 2000) of use cases suggest that some limits need to be applied. Users will not read, or at least not properly understand, long and complex use cases. But a smaller set may be insufficient to develop the class diagram. Thus, the two key roles of use cases—gather requirements and support development of the class diagram—may conflict somewhat, and the ideal set of use cases for each role are perhaps quite different.

Fifth, and perhaps most important, the role of use cases varies among methodologies. Earlier work on the UML focused on the language itself, and was largely agnostic on issues of methodology. But the Unified Process (Jacobson et al., 1999, p. 34) makes clear what was always implicit—use cases “drive the whole development process.” In particular, they provide “major input when finding and specifying the classes, subsystems and interfaces.” In addition, they drive the development of subsequent behavioral models, including activity, sequence, and statechart diagrams (Masciaszek, 2001, pp. 133-150).

Rosenberg and Scott (1999), however, suggest that “domain modeling” precede use case development. Their domain model is a “glossary of terms” based on “available relevant material” (p. 16), and is intended to evolve into the objects, attributes, operations and associations. From this, a skeletal class diagram is constructed. They warn, “Don’t try to write use cases until you know what the users will actually be doing” (p. 45). Armour and Miller (2001, pp. 48, 84) start with a Vision Document and Business Case, then create a “system glossary” prior to use case development. Their glossary builds on an existing “business glossary” if one exists. They next identify the system boundary and actors, then define goals for the system that are essentially use case names (e.g., “Apply for Loan”) (p. 20). Use cases and object models are built as “parallel activities” (p. 98). Blaha and Premerlani (1998, p. 49) state that, “Once you have a sound object model, you should specify use cases” and warn that early use cases must be regarded as “tentative and subject to revision” (p. 150). Schmuller (1999) starts with client (“the person who has the problem to be solved” [p. 5]) interviews to generate the class diagrams, then moves to user interviews to create the user case models. We
have also seen organizations begin with class diagrams for pragmatic reasons, such as when the data structure is already largely defined by existing systems.

Schneider and Winters (2001) also precede use of the UML by preparing a problem description and risk analysis, while Booch et al. (1999) include a Vision Statement (p. 344) containing a feature list and some risk identification. Why this document is not considered part of the UML is unclear, although it may be because projects have different starting points that are often outside the control of the analyst (Booch et al., 1999, p. 113). Nevertheless, some clear understanding of the problem seems essential before use case writing begins.

In summary, review of the literature shows extensive differences in how use cases are defined and used. These differences certainly exceed the basically cosmetic variations in Data Flow Diagram and Entity-Relationship Diagram formats found in standard structured analysis books. The existence of different use-case formats and roles is not surprising, given the UML’s relatively short history. Moreover, the UML brings together many analysis and design constructs because of its roots. While this is a notable achievement, the end product is loosely defined, complex (perhaps overly so), lacks a strong theoretical foundation, and thus is very difficult to test in a definitive way.

**Determining Appropriate Use Case Focus**

The use-case variations are real. Despite a general consensus that use cases are intended for conceptual modeling of system requirements, many versions of use cases incorporate significant design and implementation details (even at the level of the user interface, although Jacobson et al. [1999] now recommend separate interface prototypes). One potential way to resolve this apparent inconsistency is to adopt a contingency perspective. Different approaches may be useful under different circumstances, with the best approach in a specific situation depending on the analysts, the task, the users, and other situational variables.

However, we believe a stronger basis can be adopted to predict a most appropriate form for use cases that is applicable across a wide range of circumstances. The key to this proposal is implied by the general idea outlined earlier that use cases are requirements analysis and modeling tools that should describe what a system does (or should do), rather than how the system works (or should work).

Within this context, detailed use cases that specify low-level actor interactions with a system (e.g., down to the point of screen designs) essentially embed certain design choices. Introducing such considerations during analysis may prematurely guide the developers to specific implementation decisions. This is particularly a concern when the development process is intended to support the reengineering of existing processes, an endeavor for which Jacobson et al. (1994) strongly advocate the application of use case-driven methodology.

The potential impact on systems development of use cases that embed design decisions can be understood in the context of a well-known phenomenon in psychology—anchoring and adjustment (Tversky and Kahneman, 1974). Experiments have shown that, when people are given a problem and an initial estimate of its solution, and then are asked to find a final solution to a problem, they tend to anchor to the initial estimate (Plous, 1993). That is, they tend to provide solutions close to the initial estimate (anchor), even when those estimates are severely flawed. Anchoring is a useful heuristic that helps humans simplify problem solving in a complex situation. Unfortunately, people tend to rely on anchoring too much, resulting in an adjustment bias, in which people fail to make adequate modifications to an initial solution.
The concepts of anchoring and adjustment, although originally proposed in the context of activities such as subjective probability estimation, have a natural application to use cases. To the extent that use cases include design or implementation details that reflect current ways of doing things, reengineering or process innovation are likely to be inhibited. Consequently, we postulate that the level of innovation that can be achieved through use case-driven process modeling is inversely related to the level of design or implementation detail embodied in the use cases.

FROM USE CASES TO A CLASS MODEL

Finding Objects in Use Cases

In addition to modeling systems requirements from a user perspective, use cases and use-case diagrams specify the behavior of the objects in a system. Some developers use them to identify the object classes required in the implementation, and the behavior of objects. In this way, use cases feed the development of subsequent models in the UML—particularly the class diagram, but also sequence, activity and statechart diagrams and other UML artifacts (Maciaszek, 2001). However, it should be noted that these diagrams are recommended when “it is almost impossible to keep a textual use-case description consistent” (Booch et al., 1999, p. 159). Thus, they are designed to complement use cases rather than to provide a pictorial view of the same information.

In this context, it is useful to examine prescriptions in the UML literature for proceeding to the development of a class diagram from use cases. Booch et al. (1999) advocate applying “use case-based analysis to help find these abstractions” (p. 55), and describe this as an “excellent” way to identify classes. This view has subsequently been echoed in the Unified Process. According to Jacobson et al. (1999, p. 34), “use cases provide major input when finding and specifying classes.” They further go on to assert “classes are harvested from the use case descriptions as the developers read them looking for classes that are suitable for realizing the use cases.” More recently, Jacobson wrote, in a forward to Armour and Miller (2001, p. xiii) that “objects naturally fall out of use cases.” However, they do not offer specific prescriptions for finding classes of objects in use cases.

Jacobson et al. (1994) provide a more detailed description of the role of use cases in finding classes of domain objects:

When you have a first proposal for the most obvious entity objects, you continue to work with the use-cases. You identify objects by traversing one use-case description at a time to ensure that there is an object responsible for each part of the use case’s course of events. … When you work through the use case’s course of events in this way, it is probable that you will identify further object entities. (pp. 184-185)

“Noun/verb analysis” is also applied to use cases (e.g., Holland & Lieberherr, 1996; Armour & Miller, 2001; Brown, 2002). Nouns, particularly things, persons or roles, events, places and interactions, are possible classes. But Jacobson et al. (1994, p. 105) state: “When we say that we identify and describe a use case, we mean that we identify and describe the class.” This suggests that whoever is writing use cases should have a reasonable understanding of what classes are and which ones are likely to emerge during analysis. Interestingly, using nouns to identify classes of objects or entities for an application predates
the UML by a large period, and has been advocated for data modeling for many years. In contrast, some others have suggested the class diagram (or at least an initial attempt) ought to precede the creation of use cases. Pooley and Stevens (1999), for example, offer a detailed description of methods for identifying classes. They describe a process of identifying nouns in a systems requirement document as a mechanism for identifying candidate classes for an application (p. 58). These nouns may come from use case descriptions or other requirements documents, although Pooley and Stevens are silent on the source and nature of these documents. Rosenberg and Scott (1999, p. 16-17) search for nouns and verbs in “available relevant material,” which includes the “problem statement, lower-level requirements, and expert knowledge,” along with other sources such as marketing literature. They also identify classes before writing use cases. Booch (1994) similarly advocates the use of noun analysis to identify classes.

Indeed, Pooley and Stevens (1999) indicate a potential problem with use cases as a component of the UML:

Use case modeling should be used with caution, however, since … [t]here is a danger of building a system which is not object-oriented. Focusing on use cases may encourage developers to lose sight of the architecture of the system and of the static object structure. (p. 101)

Moreover, they go on to state: “We do not believe that examination of the use cases is on its own a good way to find objects and classes” (p. 102, emphasis at source).

Meyer (1997, p. 738) also states that “use cases are not a good tool for finding classes.” One reason is that use cases emphasize procedural sequences and this is at best irrelevant to class modeling and could even be dangerous to the process. Armour and Miller (2001, pp. 45-46) also acknowledge that use cases “represent functional stripes across the objects” and that less experienced developers risk “ending up with a functional model.” Other concerns are that users will either tend to develop use cases around what is happening now, thus failing to consider reengineering of the process, or will simply revert to functional design. However, Meyer believes that use cases can be effectively employed as a validation tool and implementation guide. The final system must be capable of handling the scenarios identified by users, although perhaps not in the same way as they originally envisioned.

Another approach to modeling classes is the use of CRC cards (Beck & Cunningham, 1989; Pooley & Stevens, 1999). While not specifically part of the UML, they can be used to model the required functionality responsibilities and association collaborations of classes once the classes that are needed have been identified.

In summary, the process for moving forward from the use-case model to identify classes is neither universally accepted, even among use case adherents, nor does it appear to be clearly defined or articulated. Proposed techniques, such as noun identification, are rooted in older techniques from data modeling. The lack of integration between use cases and class diagrams raises questions about the value of use cases in an object-oriented modeling approach.

**Objects versus Processes**

A use case is inherently task focused. It describes a sequence of activities, from start to finish, involved in completing a well-defined task or transaction. As in any task, many participants may be involved in the successful completion of a use case. These participants are candidates for objects that will be important to the system. A task or process focus, however, involves participants only to the extent that they contribute to the task. Hence, a
use case involves objects only peripherally and only as needed for the task being modeled. Therefore, a complete use-case model may not offer a cohesive picture of the structural and behavioral characteristics of the objects in the domain. Instead, these characteristics may be spread over several use cases.

The fragmentation across use cases of information needed to construct class definitions conceptually violates the principle of encapsulation, widely recognized as one of the cornerstones of object orientation. As a result, it can create a significant amount of work for analysts and developers in “defragmentation,” or reconstructing classes from a potentially large number of narrowly focused views that might be embedded in many different use cases. Although we are not aware of empirical research, or even anecdotal reports, on the extent of this problem, a case can be made that the task can be daunting. The problem is analogous to the issue of view integration in database design (Navathe, Elmasri, and Larson, 1986). There, the issue is one of developing a global conceptual schema from a set of diverse user views of the kinds of entities about which data need to be kept. Since different users have different needs, they generally have a different perspective on which entities are important, and how they are defined in terms of attributes and relationships. Problems to be resolved include identifying synonyms (entities, attributes, and/or relationships with the same meaning that have different names in different views) and homonyms (entities, attributes, and/or relationships with different meanings that have the same name in different views). This problem will increase with system size and the number of use case developers (Armour & Miller, 2001). Similar problems are possible when identifying object classes, their attributes, and their operations from a series of use cases. Given that different use cases are likely to be relevant to different users of a system, it is reasonable to expect that resolving synonyms and homonyms will impede the comprehensive and consistent identification of objects from use cases. Consequently, we propose that identifying a comprehensive and consistent class diagram from use cases alone will be very difficult, if not practically impossible.

USE CASES AS A COMMUNICATION MECHANISM

Isolating Users from the Class Diagram

In view of the apparent lack of “object” focus in use cases and the potential problems that can arise in deriving a class diagram from a use-case model, it is natural to question the rationale for including use cases in the UML. This is particularly interesting since use cases are a relatively recent addition to the UML. Much of the rationale for adopting use-case modeling in the UML focuses on their simplicity and the fact that they are “comparatively easy to understand intuitively, even without knowing the notation. This is an important strength, since the use-case model can sensibly be discussed with a customer who need not be familiar with the UML” (Pooley & Stevens, 1999, p. 93). This view suggests that other UML models, in particular the class diagram, are too technical for end users to understand or be capable of verifying.

Communication with the system’s intended users is clearly an important, if not always explicitly articulated, goal of use-cases. A use case model provides an inventory of the kinds of interactions that can occur between users and a system, providing “a forum for your domain experts, end users, and developers to communicate to one another” (Booch et al.,
Use cases are thus oriented towards interaction with end users for the purpose of verifying the developers’ understanding of how a system works or will work.

This understanding is essential for effective system development, and also helps create a “shared understanding” among team members that is a critical part of the trust building process (Ring & Van de Ven, 1989). Text may be easier to understand than diagrams, at least to an untrained user. Thus, use cases could contribute both to the accuracy of the requirements specification and also to its apparent openness. The analyst does not appear to be hiding behind diagrams that only IS professionals can understand.

In discussing the value of use cases in reengineering business processes, Jacobson et al. (1994) similarly explain the role of the use case in communicating with users or those responsible for a business process: “Use cases are best described using simple language to facilitate understanding. … The rightful owner, that is, the defined business process owner for the use case, will thereafter validate each use case’s compliance with the established corporate objectives” (p. 178).

Here, use cases are clearly established as a tool for communicating and verifying with users the developers’ understanding of how tasks are performed. In contrast, they clearly see the verification of class or object models as the purview of developers:

The reviewers are normally people in the reengineering team. It is unusual to communicate the object models to the employees in general, which means that the only people who are really involved and competent to review these models are in the reengineering team. (p. 190)

Taken together, these statements suggest that use cases are an appropriate mechanism to “shield” users from the underlying technical UML models that are the basis for systems design and implementation.

The need to exclude users from direct exposure to the class diagram in particular highlights an interesting contradiction in the UML. One of the main arguments offered for developing object-oriented approaches to systems analysis and design is that objects provide a “natural” way of thinking about a problem domain. In this regard, Booch (1996, p. 39) notes that “in a quality object-oriented software system, you will find many classes that speak the language of the domain expert” and “(e)very class in an object-oriented system should map to some tangible or conceptual abstraction in the domain of the end user or the implementer.” Jacobson et al. (1992) make the case more directly:

People regard their environment in terms of objects. Therefore it is simple to think in the same way when designing a model. A model which is designed using an object-oriented technology is often easy to understand, as it can be directly related to reality. Thus, with such a design method, only a small semantic gap will exist between reality and the model. (p. 42, emphasis at source)

The previous discussion shows that, despite this avowal of the naturalness and ease of understanding of UML models, the developers of the language explicitly introduce use cases as the primary mechanism for communicating with users to verify understanding of system functionality.

**Use Cases versus Class Diagrams for Communication**

The contradiction highlighted above can be dealt with in at least two ways. First, there is a significant literature in cognitive psychology to support the contention that people think about the world in terms of things that are classified in particular categories (e.g., Medin & Smith, 1984). Lakoff (1987) views such category structures as vital for human survival,
arguing that “(w)ithout the ability to categorize, we could not function at all” (p. 1). Parsons and Wand (1997) apply categorization research to analyze those aspects of object orientation that are meaningful from a systems analysis perspective, and conclude that classification is a vital element for object-oriented analysis.

From a cognitive perspective, one would expect that users should be able to handle class models as a mechanism for communicating with developers in verifying the conceptual structure of the domain being modeled. Of course, issues such as the difficulty of learning the notation associated with a particular class-modeling technique can negatively influence communication. Nevertheless, the fundamental idea that a domain can be described in terms of the kinds of objects in it, the attributes of the objects, the behavior the objects can exhibit, and the associations among kinds of objects, is highly consistent with research on the nature of categories that people use to structure their knowledge about things in the world. Consequently, we hypothesize that end users will be able to interact directly with class models in verifying the structure of a domain.

Cognitive psychology also provides a second basis for understanding the contradiction inherent in advocating use cases as the primary mechanism for communicating and verifying system requirements with users. Advocates of use cases point to the ease with which they can be understood, as they describe a process from start to finish. Not surprisingly, a significant body of research in cognitive science deals with how people think procedurally. For example, Schank and Abelson’s (1977) work on scripts deals with the sequencing of ordinary and exceptional events involved in a goal-oriented activity. Scripts provide a mechanism by which people can understand the temporal relationship in a list of events, including inferences about events that are not explicitly stated in a description (Bower, Black, and Turner, 1979).

Since people can think in either process-oriented or object-oriented modes, we postulate that both process-oriented and object-oriented models can be understood by users and are appropriate for verifying different aspects of application requirements. This suggests that advocating use cases for work with users, while isolating users from the class models that are the primary basis for the design of an object-oriented architecture, is not necessary. Moreover, the peripheral and diffuse role of objects in use cases is a potential source of difficulty in developing class models from use cases and verifying whether they are a good model of the domain’s category structure as understood by users. It may be more appropriate to use class models directly as a mechanism for communicating and verifying the structure of the application domain with users.

**CALL FOR RESEARCH**

The analysis presented above is purely theoretical. As far as we are aware, advocates of use cases do not offer empirical evidence that they are a “good” mechanism for communicating with users. “Goodness” of use cases could be ascertained by developing a standard of effective communication against which use cases can be evaluated. Alternatively, “goodness” could be established in a relative sense by comparing use cases to other mechanisms for communicating the same information with users. At present, the value of use cases has not been established empirically in either of these senses, although there is ongoing research interest in this direction (Siau & Lee, 2001).

Similarly, although we have presented an argument that use cases may be inadequate for developing class diagrams, such inadequacy has not been demonstrated empirically. In addition, although research on classification points to the naturalness of category structures
in organizing information about things in the world, there are few empirical studies addressing
the ability of users to understand class models. The few studies that have addressed this
were conducted prior to the development of the particular class-modeling technique that is
part of the UML, and do not provide conclusive evidence. Vessey and Conger (1994) found
that novice analysts were better able to specify requirements using process- and data-
oriented methodologies than using object-oriented methodologies. Agarwal, Sinha and
Tanniru (1996) found that the nature of the task (process-oriented or object-oriented)
influenced the relative performance of students using an object-oriented model. In contrast,
Wang (1996) found that student subjects using an object-oriented technique produced better
solutions in less time than subjects using structured analysis.

In addition, we have identified the growing tendency for use cases to include design or
implementation decisions that could be a possible impediment to effective process design
in systems development. Despite the attention paid by some to the role of use cases in process
reengineering, there is reason to believe that popular use case structures may anchor
developers to particular solution approaches and thereby narrow the scope of possible
solutions considered. However, there is no empirical evidence that such adjustment biases
occur in practice.

In view of the movement toward the UML as a standard modeling language in practice,
the paucity of empirical research on the effectiveness of various modeling techniques and
prescriptions in the UML is troubling. We have offered a theoretical framework for studying
three issues: premature inclusion of design decisions, the adequacy of use cases for
extracting class models, and the justification for choosing use cases as the primary
mechanism for developer interaction with users. From these perspectives, we think it is
important to conduct a range of empirical studies to evaluate the various modeling
components of the UML.

First, research is needed to examine whether including design and implementation
details in use cases leads to anchoring and adjustment problems with respect to effective
process redesign. This question can be addressed directly through lab experiments in which
developers design a system starting from either abstract use cases or use cases in which
design or implementation decisions are stated. In each group, the “innovativeness” of the
resulting designs relative to existing processes can be measured. To measure the external
validity of such results, correlational field studies of object-oriented development using the
UML can also be undertaken to measure the relationship between the structure of use cases
used and the extent to which implementations achieve effective redesign.

Second, research is needed to test the assertion that, since use cases do not focus on
objects, it will be difficult to extract a class diagram from a set of use cases. Although it may
be possible to test this in a controlled laboratory experiment, it would be difficult to avoid
biases in the development of use cases that might influence the ability to extract class
diagrams. Consequently, an appropriate method for examining the degree to which use cases
support the development of class diagrams (and, more generally, how class diagrams are
developed and verified) would be surveys and/or case studies of the progression from use
case models to class diagrams in projects that use the UML. Among the variables to measure
are: the extent to which use cases are the exclusive mechanism for communication and
verification of requirements with users; the extent to which use cases drive the development
of the class diagram; problems encountered in using use cases to develop the class diagram;
perceptions about the causes of such problems; and approaches that are used to deal with
these problems.
The Role of Use Cases in the UML: A Review and Research Agenda

Third, research is needed to examine whether users are capable of directly reading and understanding class diagrams, as well as other UML models. In addition, there is a need to study whether use cases add value (e.g., in ease of understanding or ability to capture additional information relative to other models in the UML). For this type of study, laboratory experiments offer the ability to enforce necessary control to permit useful comparisons across groups. Several issues need to be resolved in conducting this kind of study. For example, use cases include process or task information, while class diagrams do not. Hence, comparisons between use cases and class diagrams must be restricted to object/attribute/relationship identification, or class diagrams must be used in conjunction with other UML models to conduct comprehensive comparisons with use cases.

Table 1 summarizes a research framework for studying the need for, and effectiveness of, use cases in the UML.

### Table 1: A framework for empirical research on use cases

<table>
<thead>
<tr>
<th>Research Question</th>
<th>Primary Independent Variable</th>
<th>Primary Dependent Variable</th>
<th>Methodology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Do design/implementation details in use cases impede</td>
<td>Use case structure</td>
<td>Process innovation</td>
<td>Experiment; Case study</td>
</tr>
<tr>
<td>process redesign efforts?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can class diagrams be effectively extracted from use</td>
<td>Use cases</td>
<td>Class diagram completeness</td>
<td>Case study; Developer surveys</td>
</tr>
<tr>
<td>cases?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Do use cases facilitate communication between developers</td>
<td>Communication medium (use</td>
<td>User understanding</td>
<td>Experiments; User surveys</td>
</tr>
<tr>
<td>and users?</td>
<td>cases or class diagrams)</td>
<td>Domain coverage</td>
<td></td>
</tr>
</tbody>
</table>

CONCLUSIONS

The UML is a modeling language for object-oriented development that grew out of the combination of three distinct approaches developed in the early 1990s. Much of the conceptual foundation of the language comes out of issues in object-oriented programming (Booch, 1994), and there is little evidence about the extent to which it is appropriate as a language for modeling an application domain or system requirements. In short, we feel there is a strong need for academic research to evaluate the usefulness of the UML and determine its limitations for modeling requirements. Here, we have offered a framework for evaluating the roles of, and relationships between, use cases and class diagrams in the UML. Similar research is needed to understand the capabilities and limitations of the other models in the language.

ACKNOWLEDGMENT

This research was supported in part by a research grant from the Social Sciences and Humanities Research Council of Canada to Jeffrey Parsons.

REFERENCES


**About the Authors**

**Keng Siau** is a J.D. Edwards Professor and an Associate Professor of Management Information Systems (MIS) at the University of Nebraska, Lincoln (UNL). He is also the Editor-in-Chief of the *Journal of Database Management*. He received his Ph.D. degree from the University of British Columbia (UBC) where he majored in Management Information Systems and minored in Cognitive Psychology. He has published more than 35 refereed journal articles, and these articles have appeared in journals such as *MISQ, CACM, IEEE Computer, Information Systems, ACM's Data Base, Journal of Database Management, Journal of Information Technology, International Journal of Human-Computer Studies, Transactions on Information and Systems*, and others. In addition, he has published over 55 refereed conference papers in proceedings such as ICIS, ECIS, HICSS, CAiSE, IRMA, AMcis, and HCI. He has edited three books, three journal special issues, and six proceedings—all of them related to systems analysis and design. Dr. Siau’s primary research interests are Object-Oriented Systems Development Using Unified Modeling Language (UML), Enterprise E-Business, Enterprise Resource Planning (ERP), and Web-based Systems Development. He has served as the Organizing and Program Chairs for the International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD) (1996–2001). More information about him can be obtained at URL: [http://www.ait.unl.edu/siau/](http://www.ait.unl.edu/siau/).

***

**Jin-Kyun Ahn** is a M.S. student at the Department of Management Information Systems, KAIST. He received a B.A. degree in Industrial Design from KAIST in 1993, and completed his M.S. degree from KAIST, Daejon in 1996. Mr. Ahn’s research includes data mining and decision support systems.

**Youssef Amghar** received a Ph.D. (Computer Science) in 1989. Since 1990, he has been Assistant Professor in Information Systems at the National Institute of Applied Sciences in France. His research expertise lies in the areas of active databases, integrity constraints, and various aspects of Information System Design Methodologies. Another domain of his research activity concerns documentation aspects (XML, document modeling).

**Akhilesh Bajaj** is Assistant Professor of Information Systems Management at the H. John Heinz III School of Public Policy and Management, at Carnegie Mellon University. He received a B.Tech. in Chemical Engineering from the Indian Institute of Technology, Bombay, an M.B.A. from Cornell University, and a Ph.D. in MIS (minor in Computer Science) from the University of Arizona. Dr. Bajaj’s research deals with the construction and testing of tools and methodologies that facilitate the construction of large organizational systems, as well as studying the decision models of the actual consumers of these information systems. He has published articles in several academic journals and conferences. He is on the editorial board.
of the *Journal of Database Management*. He teaches graduate courses on basic and advanced database systems, as well as enterprise-wide systems. Akhilesh can be contacted at akhilesh@andrew.cmu.edu.

**Mira Balaban** is Senior Lecturer in the Dept. of Information Systems Engineering at Ben-Gurion University, Israel. She holds a B.Sc. in Mathematics and Statistics (1970) from Tel-Aviv University, and a M.Sc. and Ph.D. in Computer Science from Weizmann Institute of Science (1975 and 1982). She also graduated in music performance (1969) from the Rubin Academy of Music in Tel-Aviv. Her research areas are artificial intelligence, conceptual modeling, databases, programming languages, and computer music. In artificial intelligence, she works on knowledge representation and temporal reasoning, temporal constraints, and description logics. In conceptual modeling and databases, she works on semantics of databases, integrity enforcement and extensions of conceptual models, and in computer music, she develops knowledge representation tools for music concepts and processes. Dr. Balaban has published extensively in these areas, and is co-editor of a paper collection on computer music. She organized several international workshops on AI and music, and served as a member of the program committee of several conferences on computer music and on artificial intelligence.

**Qing Cao** is an Assistant Professor of MIS at the University of Missouri-Kansas City. He has co-authored several journal articles. His current research interests include software engineering and data modeling, and their applications to e-commerce. He served as a consultant for various organizations in the areas of management information systems and operations management.

**Amita Goyal Chin** is an Associate Professor in the Information Systems Department at Virginia Commonwealth University. She received her B.S. in Computer Science and M.S. and Ph.D. in Information Systems, all from The University of Maryland at College Park. Her research focuses primarily on database technology and its applications in information systems. She has been investigating new approaches in data models for text representation and document databases, hoping this area will contribute to information delivery vehicles beyond the World Wide Web. Additionally, she is also pursuing new answers to managing data quality in database systems. Dr. Chin has combined her information technology experience with her classroom skills to experiment with new teaching technologies and methodologies.

**Jérôme Darmont** received his M.Sc. and Ph.D. degrees in Computer Science from Blaise Pascal University, Clermont-Ferrand, France in 1994 and 1999, respectively. He has been an Assistant Professor and then an Associate Professor at the University of Lyon 2, France, since September 1999. His current research interests include database performance evaluation and optimization, DBMS and data warehouses auto-administration, data mining, and medical databases.

**Brian Dobing** is an Assistant Professor, Faculty of Management, University of Lethbridge, Lethbridge, AB, Canada. He received his B.Sc. in Mathematics from the University of Victoria, his M.B.A. and M.Sc. in Computational Science from the University of Saskatchewan, and his Ph.D. from the University of Minnesota. His research focuses on issues in user-analyst relationships and object-oriented analysis.
**Dov Dori** is tenured faculty at Technion, Israel Institute of Technology, and Visiting Faculty at MIT. He has been developing and teaching Object-Process Methodology during the past eight years at Technion and MIT. Dr. Dori was an invited speaker at Ford, Kodak, FAA, and Xerox, as well as many universities and international conferences worldwide. He is a Fellow of International Association for Pattern Recognition and a Senior Member of IEEE. He wrote over 130 articles and book chapters. He holds a B.Sc. in Industrial Engineering, a M.Sc. in Operation Research, and a Ph.D. in Computer Science. He is founder of Sight Code, Inc. (www.sightcode.com), which is developing Systemantica® to support OPM.

**Andre Flory** is a Full Professor of Databases and Information Systems in the Department of Computer Sciences at the National Institute of Applied Sciences (INSA) of Lyon (France) since 1989. He is currently the Director of the Computer Sciences Resources Center of the Institute. From 1970 to 1979 he was Associate Professor at the Claude Bernard University (Lyon) and from 1979 to 1998 Professor in the MIS department of Jean Moulin University (Lyon). He has served as a visiting professor at the MIS Department of the University of Minnesota in 1984. He has worked on database design for many years, and his current research interest is medical information systems. He is author of more than 100 papers published in conference proceedings, books and scientific journals. He has served on the program committees of many international conferences and has participated in several research projects sponsored by the French government. He also served as an expert at the European communities.

**Terry Halpin**, B.Sc., Dip.Ed., B.A., M.Lit.Stud., Ph.D., is Program Manager in Database Modeling, Enterprise Framework and Tools Unit, at Microsoft Corporation in Redmond, CA. Formerly an academic in Computer Science, he headed database research at Asymetrix Corporation, InfoModelers Inc., and Visio Corporation, now part of Microsoft. His research focuses on conceptual modeling and conceptual query technology for information systems, using a business rules approach. He has presented at dozens of international conferences in both industry and academia. His doctoral thesis formalized Object-Role Modeling (ORM/NIAM), and his publications include over ninety technical papers and four books, including *Information Modeling and Relational Databases*, published by Morgan Kaufmann.

**Soon-Young Huh** is an Assistant Professor of Management Information Systems at the Korea Advanced Institute of Science and Technology (KAIST). He received a B.S. degree in Electronics Engineering from Seoul National University in 1981, an M.S. degree from the KAIST, Seoul in 1983, and a Ph.D. from the University of California, Los Angeles in 1992. Dr. Huh has published articles in such journals as *Decision Sciences*, *International Journal of Intelligent Systems in Accounting, Finance and Management*, *Omega*, and the *Journal of Systems and Software*. He has also presented his research at Conferences such as Decision Science Institute Conference and INFORMS. Dr. Huh’s 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.

**Christian S. Jensen** is a Professor of Computer Science at Aalborg University, Denmark, and an Honorary Professor at Cardiff University, UK. His research concerns data management technology and spans issues of semantics, modeling, and performance. He has authored or coauthored more than 100 scientific papers in the database area. He is on the editorial
boards of ACM TODS and IEEE TKDE. He was the general chair of the 1995 International Workshop on Temporal Databases and a vice program committee chair for the 1998 IEEE ICDE Conference. He was co-program committee chair for the Workshop on Spatio-Temporal Database Management, held with VLDB’99, and for the 2001 International Symposium on Spatial and Temporal Databases, and he is the program committee chair for the 2002 EDBT Conference. He also serves on the boards of directors for a small number of companies.

**Trevor H. Jones** qualified as a management accountant through the Institute of Management Accountants (CIMA), London, England in 1979. He then moved to the United States where he attended graduate school at the Universities of South Florida and Maryland Eastern Shore, earning an M.Sc. in Applied Computer Science, May 1991. He went on to attend Drexel University, Philadelphia, earning his Ph.D. (Information Studies) in May 1995. Dr. Jones is currently Dept. Chair and Associate Professor of Information Technology at the Schools of Business Administration, Duquesne University, Pittsburgh, PA. His current research interests include conceptual modeling for database design, technology application in e-commerce, and, in a different domain, the use and effectiveness of technology in education.

**Judith Kabeli** is a Doctoral student at the Department of Information Systems Engineering at Ben-Gurion University, Israel. She earned her B.Sc. and M.Sc. degrees in Mathematics and Computer Science (1994) from Ben-Gurion University. Prior to her Ph.D. studies, Judith held professional and managerial position at the Israel Aircraft Industries. Kabeli’s research interests including data modeling, information systems analysis and design, and intelligent web-based user interfaces. Her doctoral dissertation is concerned with the development and testing of FOOM methodology.

**Peter Kawalek** gained his Ph.D. from the Department of Computer Science, University of Manchester, UK. He worked as a Research Fellow for that department, engaging with industry on many research and consultancy projects. He subsequently moved to Warwick Business School, University of Warwick, and then to Manchester Business School. He has published two books as well as several journal and conference papers. Recent projects have focused on organizational IT capability, software flexibility and process reengineering. Working with David Wastell, he has been involved in the development of major e-government initiatives.

**Mehdi Khosrow-Pour** has taught undergraduate and graduate information system courses at the Pennsylvania State University for 20 years where he was the chair of the information Systems Department for 14 years. He has also lectured at the Florida International University, American University, University of Lyon (France), University of West Indies (Jamaica), Kuwait University, University Carlos III - De Madrid, and Tehran University (Iran). He is currently the Executive Director of the Information Resources Management Association (IRMA) and Senior Editor for Idea Group, Inc. He is also the Editor-In-Charge of the *Information Resources Management Journal* (IRMJ), the *Annals of Cases on Information Technology* (ACIT), the *Information Management* (IM), and consulting editor of the *Information Technology Newsletter* (ITN). He also serves on the editorial review board of seven other international academic information systems journals. He is the former Editor-In-Charge of the *Journal of End User Computing* and the *Journal of Database Management*. Dr. Khosrow-Pour is the author/editor of more than 20+ books on various topics of
information technology utilization and management in organizations, and more than 50+ articles published in various conference proceedings and journals such as *Journal of Information Systems Management*, *Journal of Applied Business Research*, *Journal of Systems Management*, *Journal of Education Technology Systems*, *Computing Review*, and *Review of Accounting Information Systems*. He is a frequent speaker at many international meetings and organizations such as: the Association of Federal IRM, Contract Management Association, Financial Women Association, National Association of States IRM Executives, IBM, and the Pennsylvania Auditor General Department.

**Ramayya Krishnan** is the W. W. Cooper and Ruth F. Cooper Professor of Information Systems at Carnegie Mellon University. He holds appointments at the Heinz School of Public Policy and Management. Krishnan’s research interests are in e-business technology, management and policy issues. His work on these topics has been funded by DARPA, Army Research Office, and the National Science Foundation. Krishnan serves on the editorial board of several Information Systems journals. He recently co-edited a special issue of *Interfaces* (available online at http://pubsonline.informs.org/feature/) and is currently co-editing special issues of *Management Science on E-business*.

**Frederique Laforest** received the title of Engineer in Computer Science from the National Institute of Applied Sciences of Lyon (INSA Lyon - France) in 1993. She received a Ph.D. degree in Computer Science at INSA Lyon in 1997 with a thesis entitled “Generic models and user interfaces—application to medical information systems.” From 1996 to 1998, she was Assistant Professor at the Department of Informatics of the Technological Institute of Lyon University. Since 1998, she is Associate Professor at the Telecommunications, Services and Uses Department of INSA-Lyon. Her teaching fields are Algorithms and Programming, Databases, Operating Systems and Information Systems. Since 1993, she is working in the Information Systems Laboratory (LISI). Her major research topics concern information systems, documents and databases.

**Heesook Lee** is an Associate Professor of MIS at Korea Advanced Institute of Science and Technology (KAIST). He received his doctorate in Management Information Systems from the University of Arizona. He taught MIS for the University of Nebraska at Omaha during 1992-1994. His research interests include information strategy, digital business, and knowledge management. He has published in *Journal of Database Management*, *Information and Management*, *Omega, Information Systems*, *Journal of Systems and Software*, *Information Processing and Management*, and *Annals of Operations Research*.

**Madjid Meziane** received his Ph.D. degree in Computer Science from the National Institute of Applied Sciences in France in 1998. His research interests include active database and design methodologies. His is currently junior manager at PEUGOT (France) and works on a project concerning software maintenance and application re-documentation.

**Kae-Hyun Moon** is a Ph.D. student at the Department of Management Information Systems, KAIST. He received a B.A. in Management Science from KAIST in 1994, and M.S. from KAIST, Seoul, in 1996. Mr. Moon’s research includes intelligent query processing and decision-support systems.
Jeffrey Parsons is Professor of Information Systems in the Faculty of Business Administration at Memorial University of Newfoundland. He received a Ph.D. from The University of British Columbia. His research interests include systems analysis and design, database management, and electronic commerce. His research has been published in journals such as Management Science, Communications of the ACM, and ACM Transactions on Database Systems.

Parag Pendharkar is Assistant Professor of Information Systems at Penn State Harrisburg. His research interests are artificial intelligence and expert systems. His work has appeared in Annals of Operations Research, Communications of ACM, Decision Sciences, and several others. He is a member of INFORMS, APUBEF, and DSI.

Rosanne Price is a lecturer at RMIT University, Australia. She is currently completing her Ph.D. at Monash University, where she previously worked as an assistant lecturer. She received her M.Sc. in Computer Science from National University of Singapore, B.A. in Biology from Vassar College, was a visiting student in the Computer Science Department of M.I.T., and a research intern at Bell Laboratories in Murray Hill. After work in industry in medical computing, she joined the staff of the Institute of Systems Science (National University of Singapore) researching multimedia database and information retrieval systems and later worked as a lecturer for Boston University and University of Maryland’s Masters in Computer Science program in Germany. Her main interests are conceptual data modeling, multimedia/object-oriented databases, and information retrieval.

James A. Rodger is an Associate Professor of Management Information Systems at Indiana University of Pennsylvania. He received his Doctorate in MIS, from Southern Illinois University at Carbondale, in 1997. Dr. Rodger teaches Network Administration, System Architecture, Microcomputer Applications and Intro to MIS, at IUP. He has worked as an Installation Coordinator for Sunquest Information Systems, and presently does consulting work on Telemedicine Connectivity for the Department of Defense and Management Technology Services, Inc. Dr. Rodger has published several journal articles related to these subjects. His most recent article, “Telemedicine and the Department of Defense,” was published in Communications of the ACM.

Michel Schneider is Full Professor in Computer Science at Blaise Pascal University, Clermont-Ferrand, France. He has been contributing for many years to the development of this discipline. He is presently in charge of the Computer Science Doctoral School. His current research interests include clustering in object-oriented databases, benchmarking, document retrieval, and web-based systems.

Peretz Shoval is Professor of Information Systems and head of the Department of Information Systems Engineering at Ben-Gurion University, Israel. He earned his B.Sc. in Economics (1970) and M.Sc. in Information Systems (1975) from Tel-Aviv University, and Ph.D. in Information Systems (1981) from the University of Pittsburgh, where he specialized in expert systems for information retrieval. In 1984, he joined Ben-Gurion University, where he started the Information Systems Program in the Department of Industrial Engineering & Management. Prior to moving to academia, he held professional and managerial positions in computer companies and in the IDF. Prof. Shoval’s research interests include information systems analysis and design methods, data modeling and database design, and information retrieval and filtering. He has published numerous papers in journals and presented his
research in many conferences. Shoval has developed methodologies and tools for systems analysis and design, and for conceptual and logical database design.

**Il-Yeol Song** is a Professor in the College of Information Science and Technology at Drexel University, Philadelphia, PA. He received his M.S. and Ph.D. degrees in Computer Science from Louisiana State University in 1984 and 1988, respectively. His current research areas include database modeling and design, design and performance optimization of data warehousing, OLAP, object-oriented analysis and design, and object-oriented modeling. He has published over 80 refereed articles in various journals, conferences, and books. He has been the guest editor for several journals, including *Journal of Computer and Software Engineering* (1995) and *Journal of Computer Science and Information Management* (1999). He served as the program co-chair for CIKM99, DOLAP98 and DOLAP99.

**Woojong Suh** is a research fellow of e-Business Research Center at POSCO Research Institute (POSRI) in Korea. He received his doctorate in Management Information Systems from Korea Advanced Institute of Science and Technology (KAIST). He received his bachelor’s degree and master’s degree in Applied Statistics from Yonsei University, Korea. His research interests include hypermedia information systems, e-business strategy, workflow analysis, and knowledge management.

**Nectaria Tryfona** is an Associate Professor in Department of Computer Science, Aalborg University, Denmark. Her research interests are in the area of database modeling, geographic information systems design, spatiotemporal information systems, and spatial data mining. She is the author of various articles in books, journals, and conference proceedings. She received her Dipl. Eng. and Ph.D. from the Department of Computer Engineering and Informatics, University of Patras, Greece, in 1991 and 1994, respectively. She was with the National Center of Geographic Information and Analysis at the University of Maine as Post-Doctoral Research Associate from 1995 to 1997. She continues to serve as a program committee member in ACM GIS, STDBM, ISD, IDEAS, and as external reviewer for ICDE, CaiSE, COSIT, SSDBM, GeoInformatica Journal and International Journal of GIS.

**David Wastell** gained his Ph.D. from the Applied Psychology Unit at Cambridge University, UK. He worked for a number of years lecturing in Information Systems at the Department of Computer Science, University of Manchester. He moved from there to his current position as Chair of the Information Society in the Information Systems Institute, University of Salford. He publishes regularly in major journals and conferences. He has undertaken many research and consultancy projects. Two prominent recent projects are the development of e-government initiatives at Salford City Council, and the Multi-Agency Data Exchange (MADE) for Community Safety in Lancashire.

**Jonathan Whittle** received his B.A. degree in Mathematics from the University of Oxford in 1995. He obtained a Masters degree in Artificial Intelligence in 1996 and a Ph.D. in Formal Methods in 1999, from the University of Edinburgh, Scotland. He now works in the Automated Software Engineering Group at NASA Ames Research Center, Moffett Field, CA. His research interests lie in making lightweight Formal Methods practical and usable. His Ph.D. concentrated on developing an editor for functional programs that incrementally carries out sophisticated forms of static analysis. He has also written papers on applying lightweight analysis to UML models.
Index

A
a priori 35
abstract syntax 308
abstract value 213
abstraction barrier 8
abstraction level 213
active behavior 147
active database 134
Actor-Network Theory (ANT) 97
ADISSA methodology 59
aeromedical evacuation suitcase 274
aggregate metrics 297
aggregation 37
algorithm 167
Alloy Constraint Analyzer 333
anchoring and adjustment 371
application 229
application-oriented meta-information systems 250
approximate join 221
approximate operator 221
approximate selection 218
arity 3, 311
association 37, 310
association level 352
attribute modification 8
attribute updates 44
automation 113

B
base case 174
behavior specification 144
behavioral stratification 149
binary 4

Binary Decision Diagrams (BDDs) 331
binary relationship types 7
business adaptation 119

C
C4ISR networks 268
canvas view 237
cardinality constraints 4, 23, 147
clarity of a language 308
classical information system 193
classification scheme 233
cluster 243
clustering effect evaluation 53
CMU-WEB schema 237
code column 198
cognitive overhead 234, 242
cognitive psychology 376
coherence 234
cohesion 243
complex transaction 75
complexity management problem 88
conceptual clustering approach 212
conceptual data modeling 17
conceptual join 222
conceptual schema optimization 318
conceptual selection 220
crone syntax 308
connect method 12
consistency 242
Constrained Ternary Relationship (CTD) Rule 24
container objects 40
content-dependent metadata 251
control type 253
coupling 243
credit card transactions 92
Cybernetics 120
cyclomatic complexity measure 295

D
data allocation 157
data flow diagrams 136, 369
database administrator 157
database evolutions 44
database instance 6
database level operations 8
database system extension 152
DDL (data definition language) 2
declarative knowledge 110
decomposition 24
Defense Information Systems Network
(DISN) 268
degree of support for interrelated events
229
deletion constraint 29
derivation rules 147
design questions 234
destination node 255
determining appropriate 371
difficulty of navigation 235
disconnect method 13
Distributed Database Management System
(DDBMS) 156
DML (data manipulation language) 2
document analyzer algorithm 199
document loaning 140
document restitution 140
document type definition (DTD) 190
documents- and rules-based user interface
for data 193
dom 6
domain abstraction 214
domain abstraction hierarchy 213
domain-general 113
domain-specific 113
drill-down search 260
DRO (Detection & Reclustering of Objects)
34
DSTC (Dynamic, Statistical, Tunable
Clustering) 34
dynamic anchor 253
dynamic-sized objects (DO) 40

E
EDMS (Electronic Document Management
System) 250
EER schema 2
efficiency of use 235
elementary methods 67
encapsulation 374
encapsulation problem 87
encoding 109
Engineering Integrated Product Team (E-
IPT) 280
enhanced ER (EER) 1
entities 309
entity relationship (ER) 1, 17
entity type 3
Entity-Relationship-Event-Rule (ERER) 137
event 229
event-condition-action (ECA) 135
exclusion constraints 316
exhaustive search 168
existence-dependent symbol 346
Explicit Binary Permission (EBP) Rule 23
expressibility of a language 308
extended entity-relationship model (EER)
17
extended ER model 137

F
field medical surveillance system 268
FOOM 58
formal foundation 309
formal methods 323
format-dependent metadata 251
free text documents 190
full REALLOCATE 168
fundamental structural relations 92
fuzzy database approach 212

G
global behavior 139
global coherence 234, 241
graphical model 234, 241
Gray’s Criteria 46
group symbol 346
H

hierarchical automata 332
holistic change vision 271
human information-processing 109
hybrid domain systems 250
Hyperdocument Meta-information System (HyDoMiS) 248, 254
hyperlink outside application (HLOA) 237
hyperlink within application (HLWA) 237
hypermedia documents 234, 249
hypermedia information systems (HISs) 261
hypermural model benchmark 37
hypermural model database 37
hypermural model operations 37

I

I & I events 232
Implicit Binary Override (IBO) Rule 24
independent metrics 296
information chunk (IC) 237
information chunk depiction (ICD) 237
information modeling 106
information modeling method 107
inheritance 37
insertion constraint 27
integrity methods 2
internal database application 152

J

JINC system 276
justitia benchmark 40
justitia database 40
justitia operations 41

K

key 3
keyword search 260
knowledge abstraction hierarchy (KAH) 211

L

language design 309
Law of Requisite Variety 121
learnability of interface 235
Libretto 268
lightweight data entry tool (LDET) 275
lines-of-code (LOC) 294
local area network (LAN) 274
local behavior 139
local coherence 234, 241
location of processing 229
log-dependent metadata 251
look-across constraint 4

M

management-oriented meta-information systems 250
mandatory key 4
mapping component 201
mapping world base 201
medical equipment sets (MES) 272
medical record domain 191
medical treatment facilities (MTFs) 269
medical workstation (MeWS) 268, 274
memorability of interface 235
menus class 69
message diagram 76
meta-information system 249-50
meta-model 196
metadata 249
metadata classification 251
metadata elements 252
metadata management 254
method engineering 108
method modeling language 296
methods 2
metrics 229
Micro-INC (MINC) data controller 277
model checking 330
model finding 333
model multiplicity problem 88
modeling constructs 108
multiplicity constraints 312

N

n-ary relationship 18
network traffic 236
no events value 231
no processing value 231
node 237, 252
non-binary relationship types 7
O

object base generation 48
object base usage 48
object behavior 143
object cardinality constraints 318
object class 2, 351
object clustering benchmark 34, 41
object constraint language (OCL) 330
object identifier (OID) 34
object role modeling (ORM) 307
object-orientation (OO) 293
object-oriented 104, 105
object-oriented (OO) model 2
object-oriented approach 60
object-oriented context 138
object-oriented database management systems (OODBM) 34
object-oriented databases (OODBs) 34
object-oriented systems 369
object-process diagrams (OPDs) 89
object-process language (OPL) 89
object-process methodology (OPM) 62, 87, 91, 104-105
OCB database 41
OCB operations 43
only independent and idempotent (I&I) events value 231
OO programming software 76
OO-DFDs (object-oriented data flow diagrams) 58
OO7 benchmark 38
OO7 database 39
OO7 operations 39
OPL specification 97
order number 142
organizational hypermedia documents (OHDs) 248
owner entity type 6

P

pages following the Extensible Markup Language (XML) 230
pages following the Hyper Text Markup Language (HTML) 230
pages with completely flexible content 230
paradigm 88
partial REALLOCATE 168
participation constraint 4
partitioning 150
pattern column 198
personal information carrier (PIC) 274
petri nets (PN) 137
primitive objects (PO) 40
primitive update methods 2
procedural knowledge 112
processing on the client and server 231
processing only on the client 231
processing only on the server 231
processing type 253

Q

qualified associations 316
queries 40
query language 211
query processing approach 211

R

radio music programs system 80
reconnect method 14
referential integrity 147
referential type 253
relationship instance 310
relationship type 3, 310
relationship/integrity method 67
ring constraints 318
role name 4
rule stratification 149
rule-based approach 212

S

scenarios 370
schema level operations 8
schema revision 10
semantic gap 375
semantically constraining binary (SCB) relationship 20
semantically unrelated binary (SUB) relationship 20
sequences of interrelated events 231
sequential update 44
server capacity 236
sets of I & I events interspersed with interrelated 231
simple transaction 75
source node 255
spatial constraint 358
spatial cover (SCE) 359
spatial data 341
spatial derivation 358
spatial equal (SE) 360
spatial inclusion (SIE) 359
spatial interior (SI) 359
spatial object class 351
spatial objects 358
spatial part-whole (PW) relationships 358
spatial symbol 347
spatially dependent Association 353
spatiotemporal data 341
spatiotemporal object class 351
spatiotemporally dependent association 353
specialization 6
specific value 213
specification box 346
statecharts 324
static anchor 253
static-sized objects (SO) 40
strongly structured documents 190
structural modification operations 40
structure charts (SC) 59
structure metrics 295
structure of pages 229
structured maps 192
structured system analysis (SSA) 59
structured system design (SSD) 59
sub-systems of the VSM 122
subtypes 6
subtyping 317
supporting module 254
system behavior 73
system-dependent metadata 251

T

tactical medical communications 268
Team Care Automation System (TCAS) software 276
technical commission 124
Telemedicine 266
temporal object class 351
temporal symbol 347
temporally dependent association 353
temporally dependent spatial object class 351
temporally dependent spatiotemporal object class 351
ternary relationship 19
ternary-binary combination 21
Theater Clinical Encounter Application (TCEA) 274
Theater Medical Core Services (TMCS) 268, 274
Theater Telemedicine Prototype Program (T2P2) 274
Theater Telemedicine Teams (TTTs) 269, 274
thematic symbol 347
Toshiba Libretto end user terminal (EUT) 274
transactions design 59
traversal 39

U

UML semantics 326
UML sequence diagram 325
UML specification 326
unary 4
unified software development process 366
usability metrics 241
use case 366
use case internal structure 368
use case variations 371
user-interface adjustment 235, 242

V

validation mechanisms 308
value abstraction hierarchy 213
value constraints 147
values 309
viable system model (VSM) 119
view integration 374
Violet Computing (VC) 122
Violet Operating System (VOS) 122
W

weakly structured documents 191
workflow-dependent metadata 251
working memory 109
WWW applications 229
Add these IGI Series to your personal or library IT collection today. Each series will greatly enhance your collection in information technology. You will benefit from these publications on advanced research in various areas of information technology innovation, applications, utilization, management and organizational and societal issues.

### Cases on Information Technology Series (ISSN 1537-9337)

<table>
<thead>
<tr>
<th>Vol</th>
<th>Copyright</th>
<th>ISBN</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>2002</td>
<td>1-930708-40-8</td>
<td>US$89.00</td>
<td></td>
</tr>
<tr>
<td>4-2</td>
<td>2002</td>
<td>1-930708-16-5</td>
<td>US$74.95</td>
<td></td>
</tr>
<tr>
<td>4-3</td>
<td>2002</td>
<td>1-930708-27-0</td>
<td>US$74.95</td>
<td></td>
</tr>
<tr>
<td>3-1</td>
<td>2001</td>
<td>1-878289-61-6</td>
<td>US$89.00</td>
<td></td>
</tr>
<tr>
<td>2-1</td>
<td>2000</td>
<td>1-878289-83-7</td>
<td>US$89.00</td>
<td></td>
</tr>
<tr>
<td>1-1</td>
<td>1999</td>
<td>1-878289-56-X</td>
<td>US$89.00</td>
<td></td>
</tr>
</tbody>
</table>

### Advanced Topics in Database Research Series (ISSN 1537-9299)

<table>
<thead>
<tr>
<th>Vol</th>
<th>Copyright</th>
<th>ISBN</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>2002</td>
<td>1-930708-41-6</td>
<td>US$74.95</td>
<td></td>
</tr>
</tbody>
</table>

### Advanced Topics in Information Resources Management Series (ISSN 1537-9329)

<table>
<thead>
<tr>
<th>Vol</th>
<th>Copyright</th>
<th>ISBN</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>2002</td>
<td>1-930708-44-0</td>
<td>US$74.95</td>
<td></td>
</tr>
</tbody>
</table>

### Advanced Topics in Global Information Management Series (ISSN 1537-9302)

<table>
<thead>
<tr>
<th>Vol</th>
<th>Copyright</th>
<th>ISBN</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>2002</td>
<td>1-930708-43-2</td>
<td>US$74.95</td>
<td></td>
</tr>
</tbody>
</table>

### Advanced Topics in End User Computing Series (ISSN 1537-9310)

<table>
<thead>
<tr>
<th>Vol</th>
<th>Copyright</th>
<th>ISBN</th>
<th>Price</th>
<th>Qty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>2002</td>
<td>1-930708-42-4</td>
<td>US$74.95</td>
<td></td>
</tr>
</tbody>
</table>

▸ RECOMMEND THIS IT SERIES TO YOUR LIBRARY.