Nabil R. Adam Bharat K. Bhargava (Eds.)

Advanced Database Systems

Springer-Verlag
Berlin Heidelberg New York
London Paris Tokyo
Hong Kong Barcelona
Budapest

Volume Editors

Nabil R. Adam
Department of MS/CIS, Rutgers University
Newark, NJ 07102, USA

Bharat K. Bhargava
Department of Computer Science, Purdue University
West Lafayette, IN 47906, USA

CR Subject Classification (1991): H.2, H.3, H.5.1

ISBN 3-540-57507-3 Springer-Verlag Berlin Heidelberg New York
ISBN 0-387-57507-3 Springer-Verlag New York Berlin Heidelberg

Typesetting: Camera-ready by author
Printing and binding: Druckhaus Beltz, Hemsbach/Bergstr.
45/3140-543210 - Printed on acid-free paper

Preface

Currently, database management is receiving wide interest in both academic and industrial worlds. New application areas such as CAD/CAM, Geographic Information Systems, and Multimedia are emerging. The needs of these application areas are far more complex than those of conventional business applications.

The purpose of this book is to bring together a set of current research issues that addresses a broad spectrum of topics related to database systems and applications.

The book is divided into four parts. Part I includes seven chapters that focus on object-oriented databases, Chapters included in Part II address issues related to temporal/historical database systems. Part III addresses query processing in database and is made up of four chapters. The last part of the book includes five chapters that discuss heterogeneity/interoperability/open system architectures and multimedia.

We would like to express our gratitude to the reviewers for their valuable comments.

Contents

I Object-Oriented Databases 1

1 COMPOSE: A System for Composite Specification and Detection 3

by N. Gehani, H.V. Jagadish, O. Shmueli

1.1 Introduction . . . 3
1.2 Event Expressions . . . 4
1.2.1 Basic Operators . . . 5
1.2.2 Additional Operators . . . 5
1.2.3 Regular Expressions . . . 6
1.3 Examples . . . 7
1.3.1 Simple Examples . . . 7
1.3.2 Discount Rate Cut . . . 7
1.3.3 Attributes and Masks . . . 8
1.3.4 Parameters . . . 9
1.3.5 Correlation Variables . . . 10
1.4 Composite Event Detection . . . 10
1.4.1 Design Decisions . . . 11
1.4.2 Masks . . . 11
1.4.3 Generic Automaton For Implementing Events With Parameters . . . 12
1.5 Compose System . . . 12
1.6 Examples of Automata Generated by Compose . . . 13
1.7 Conclusion . . . 14

2 Access Controls in OO Database Systems – Some Approaches and Issues 17

by E. Bertino, S. Jajodia, P. Samarati

2.1 Introduction . . . 17
2.2 Object-oriented Data Model . . . 19
2.3 Mandatory Access Control . . . 21
2.4 Discretionary Access Control . . . 25
2.4.1 The ORION Authorization Model . . . 25
2.4.2 Content-dependent Authorizations . . . 34
2.4.3 Accessing Objects Through Methods . . . 35
2.5 Research Issues in Mandatory Access Control . . . 38
2.5.1 Modeling Multilevel Entities as Single-level Objects . . . 38

2.5.2 Object Updates and Secure Garbage Collection Mechanisms 40
2.5.3 Polyinstantiation 41
2.5.4 Comparison With Relevant Work 42
2.6 Research Issues in Discretionary Access Control 43
2.7 Conclusion 44

3 The Decomposition Property of Non-Deterministic Databases 45
by K. Vadaparty, S. Naqvi
3.1 Introduction 45
3.2 Basic Notions 49
3.2.1 Entailment, and Data Complexity 49
3.2.2 Choices and Data Complexity 50
3.3 Size and Density of a Witness 52
3.3.1 Size of a Witness 53
3.3.2 Density of Witnesses 56
3.4 Modulewise Evaluation 56
3.5 Future Extensions 63

4 The Architectures of an Object Base Environment for Simulation 65
by P.C.-Y. Sheu, L.J. Peterson
4.1 Introduction 65
4.2 Related Work 66
4.3 Object Representation 67
4.3.1 Complex Objects 67
4.3.2 Active Objects and Models 68
4.4 Management of Active Objects 71
4.4.1 State Space and Criteria 71
4.4.2 Adding and Removing A State 71
4.4.3 Adding and Removing A Rule 72
4.4.4 Adding and Removing An Attribute 73
4.4.5 Adding and Removing A Method or A Class 73
4.5 Simulation 73
4.5.1 Rule Processing 73
4.5.2 Logic of The Simulator 78
4.5.3 Parallel Processing 79
4.5.4 Example 80
4.6 Object-Oriented Evaluation of Rule Networks 81
4.6.1 Structures of Extensional Databases and Query Networks 81
4.6.2 Object-Oriented Rule Evaluation 83
4.7 Conclusion 85

5 Transition from a Relation to Object Model Implementation 87

by B. Bhargava, Y. Jiang, J. Srinivasan, P. Dewan

5.1 Modeling Complex Data 87
5.2 Survey of Extended Relational Systems 88
5.3 O-Raid System Design and Implementation 90
5.3.1 Expand Query Language 90
5.3.2 Extend Data Definition Facility 91
5.3.3 Data Manipulation Language (DML) 97
5.4 Performance Studies 99
5.5 Research Issues 102

6 An Object-Oriented Knowledge Model for KBMS-Supported Evolutionary Prototyping of Software Systems 105

by S.Y.W. Su, Y. Shyy

6.1 Introduction 105
6.1.1 Motivation 105
6.1.2 Related Works 106
6.2 Knowledge Model Overview 107
6.2.1 Classes 107
6.2.2 Objects and Instances 108
6.3 Structural Abstraction 108
6.3.1 Structural Association Definitions 108
6.3.2 Encapsulation and Inheritance 109
6.3.3 Extensible Kernel Model 109
6.3.4 Structural Association Patterns 110
6.4 Behavioral Abstraction 111
6.4.1 Method Model and Control Associations 112
6.4.2 Method_model Object and Evolutionary Prototyping . . . 115
6.4.3 Rule Definition 116
6.5 Conclusions 117

7 Applying OOAD in the Design and Implementation of an Intelligent Geographic Information System 127

by R. Subramanian, N.R. Adam

7.1 Introduction 127
7.2 Modeling & Query Processing 128
7.3 Spatial Data Modeling 130
7.3.1 The Design Methodology 130
7.4 The Responsibility-Driven Approach 131
7.5 Developing the Data Model 132
7.5.1 The Exploratory Phase 132
7.5.2 The Analysis Phase 139
7.6 Implementation 144
7.7 Conclusion 145

II Temporal/Historical Database Systems **151**

8 Indexical Databases **153**

by J. Clifford

8.1 Motivation 153
8.2 The Indexical Database Model 155
8.2.1 The Structures 155
8.2.2 Discussion of the Structures 156
8.2.3 An Indexical Example: The Watergate Database 158
8.2.4 The Operations 159
8.2.5 Partial Functions 168
8.3 Instances of the Indexical Database Model 172
8.3.1 HRDM 172
8.3.2 Bitemporal Database Models 173
8.4 Summary and Conclusions 173

9 A Temporal Query Language for a Conceptual Model **175**

by R. Elmasri, V. Kouramajian

9.1 Introduction 175
9.2 Representing Time 176
9.3 The Temporal Data Model 176
9.3.1 Conceptual Objects: Entities 176
9.3.2 Temporal Objects: Roles 177
9.3.3 Temporal Constraints among Roles 178
9.3.4 Non-Temporal Attributes 178
9.3.5 Temporal Attributes 178
9.3.6 Classes and Superclass/Subclass Relationships 179
9.3.7 Conceptual Relationships 179
9.3.8 Temporal Relationships 180
9.3.9 Temporal Constraints among Relationships 180
9.3.10 An Example 180
9.4 Temporal Query Language Constructs 181
9.5 The Temporal Query Language 183
9.5.1 Temporal Projection 184
9.5.2 Temporal Selection 187
9.5.3 Temporal Version Restriction Operators 188
9.5.4 Temporal Scope Operators 190
9.6 Conclusions 190

10 A Data Model for Time-Series Analysis **191**

by A. Segev, R. Chandra

10.1 Introduction 191
10.2 Main Features of The Data Model 192
10.2.1 Relevant Research 193
10.3 Vector Based Data Model 194

10.4 Concepts 198
10.5 Rules 203
10.6 Calendar 204
10.7 Temporal Query Language 205
10.8 Special Operators for Time-Series Database 206
10.9 Handling of Missing Values 208
10.10User Environment 208
10.11Conclusion 209

11 A Relational Model and SQL-like Query Language for Spatial Databases **213**

by S.K. Gadia, V. Chopra

11.1 Introduction 213
11.1.1 Related Works 213
11.1.2 Our Concept of A Spatial Region 214
11.1.3 Weak Data Typing 215
11.1.4 Uniformity of Attribute Values 215
11.1.5 Experience From Temporal Databases 215
11.2 Our Model 216
11.2.1 Spatial Regions 216
11.2.2 Attribute Values 216
11.2.3 Value Navigation 217
11.2.4 Spatial Tuples 217
11.2.5 Spatial Relations 218
11.2.6 Weak Equality and Restructuring 219
11.3 Querying in The Model 219
11.3.1 Spatial Expressions 219
11.3.2 Boolean Expressions 220
11.3.3 Relational Expressions 220
11.3.4 Examples 221
11.4 Seamlessness of SpaSQL 222
11.5 Algebraic Nature of SpaSQL 224
11.6 Conclusion 224

III Query Processing in Database Systems **227**

12 Parallel Query Processing **229**

by P.S. Yu, M.-S. Chen, J.L. Wolf, J. Turek

12.1 Introduction 229
12.2 Preliminaries 230
12.3 Issues 231
12.3.1 Intra-operator Parallelism 232
12.3.2 Inter-operator Parallelism 232
12.3.3 Inter-query Parallelism 233
12.3.4 Remarks 234
12.4 System Architectures 234

12.5 Data Skew and Intra-operator Parallelism 235
12.5.1 Conventional Algorithm 238
12.5.2 Dynamic Algorithms 239
12.5.3 Sophisticated Algorithms 240
12.6 Complex Multi-join Query 242
12.6.1 Join Methods without Pipelining 243
12.6.2 Join Methods with Pipelining 248
12.7 Scheduling Multiple Complex Queries 252
12.7.1 A Hierarchical Approach to Inter-Query Parallelism . . . 253
12.7.2 Scheduling Independent Tasks 255
12.8 Summary 258

13 Towards Flexible Distributed Information Retrieval 259
by D.W. Flater, Y. Yesha
13.1 Introduction 259
13.2 Information Retrieval Techniques 260
13.2.1 Tag-Based Retrieval 260
13.2.2 Partial Content-Based Retrieval 261
13.2.3 Full Content-Based Retrieval 261
13.3 Thesauri and Dictionaries 261
13.3.1 Thesauri 262
13.3.2 Dictionaries 263
13.4 Fuzzy Retrieval 263
13.4.1 Partial Content-Based Methods 264
13.5 Distributed Approaches to Information Retrieval 265
13.5.1 Current Research Issues 266
13.6 Architecture 267
13.6.1 Preliminaries 267
13.6.2 Query Routing 268
13.6.3 Cooperative Caching 268
13.6.4 Simulation Results 269
13.6.5 Implementation 272
13.7 Concluding Remarks 276

14 Efficient Parallel Recovery in Replicated Databases 277
by R. Tewari
14.1 Introduction 277
14.2 Consistency Control Algorithm 278
14.3 Parallel Merge Protocol 279
14.4 Extension of the Parallel Merge Protocol 285
14.5 Incorporating Parallelism in the Merge Protocol 285
14.6 Performance Analysis of the Parallel Merge Algorithm 286
14.7 Conclusion 287

15 Document Allocation in Multiprocessor Information Retrieval Systems **289**

by H.T. Siegelmann, O. Frieder

15.1 Introduction 289
15.2 MDAP is NP-Complete 290
15.3 Related Efforts 292
15.3.1 Previous Approximations of the Mapping Problem 293
15.3.2 Related Information Retrieval Systems 293
15.4 A Genetic Algorithm for MDAP 294
15.5 Theoretical Foundations 298
15.6 Experimental Evaluation 301
15.7 Conclusions and Future Directions 309

IV Heterogeneity/Interoperability/Open System Architectures/Multimedia Database Systems **311**

16 Amalgame: A Tool for Creating Interoperating, Persistent, Heterogeneous Components **313**

by J.-C. Franchitti, R. King

16.1 Introduction 313
16.1.1 The Persistent and Heterogeneous Applications 313
16.1.2 Goals and Novelty of the Amalgame System 316
16.2 Related Work 316
16.2.1 Interoperability Support 316
16.2.2 Architectural Representation Languages and Megaprogramming 317
16.2.3 Extensible Reusable Heterogeneous Frameworks 318
16.3 An Overview of Amalgame 318
16.3.1 A Motivating Example 318
16.3.2 The Designer's View of the Amalgame Toolkit 319
16.3.3 Practical Use of the Amalgame Toolkit 320
16.3.4 The Internal Architecture of the Amalgame Toolkit . . . 325
16.4 An Arcadia Demonstration Scenario 330
16.4.1 High-level Description of the Arcadia Experiment 330
16.4.2 Implementation 331
16.4.3 Benefits of The Amalgame Approach 332
16.5 Future Directions 333
16.5.1 A Joint Arcadia and Prototech Demonstration Scenario . 333
16.5.2 An Interoperability Experiment with Chiron 333
16.5.3 An International "Library" of Deployed "Wrapped" Persistent Applications 334
16.6 Conclusion 334
16.7 Acknowledgments 334

17 Correctness and Enforcement of Multidatabase Interdependencies **337**

by G. Karabatis, M. Rusinkiewicz, A. Sheth

17.1 Introduction 337
17.2 Background 338
17.2.1 Specification of Interdependent Data 338
17.2.2 Conceptual System Architecture 340
17.3 Correctness of Dependency Specifications 341
17.3.1 Dependency Graph 341
17.3.2 Correctness Requirements Involving Consistency Predicates 343
17.3.3 Correctness Requirements Involving Dependency Predicates 344
17.4 Polytransactions 345
17.5 Consistency of Interdependent Data 348
17.5.1 States of Interdependent Data Objects 348
17.5.2 Measures of Consistency 349
17.5.3 Updatability of Objects 351
17.6 Concurrent Execution of Polytransactions 353
17.6.1 Correctness of the Execution of a Single Polytransaction . 353
17.6.2 Conflicts in Polytransactions 354
17.6.3 Polytransactions with Temporal Constraints 356
17.7 Conclusion 357

18 FEMUS: A Federated Multilingual Database System **359**

by M. Andersson, Y. Dupont, S. Spaccapietra, K. Yétongnon, M. Tresch, H. Ye

18.1 Introduction 359
18.2 FEMUS 360
18.2.1 The ERC+ Approach 361
18.2.2 The COCOON Approach 362
18.3 The Mapping Process 365
18.3.1 Mapping an ERC+ Schema to COCOON 366
18.3.2 Mapping a COCOON Schema to ERC+ 368
18.3.3 Operators Mapping 369
18.4 The Integration Process 371
18.4.1 Assertion-driven Integration 374
18.4.2 Integration Through Augmentation 375
18.5 Negotiation 376
18.5.1 Exchanging Metadata 376
18.5.2 Exchanging Data 377
18.6 Implementation Issues 377
18.7 Consistency Requirement 378
18.7.1 Detection of Relevant Updates 378
18.7.2 Differential Refresh 379
18.8 Conclusion and Future Research 380

19 Communication and Synchronization Issues in Distributed Multimedia Database Systems **381**

by S. Browne

19.1 Introduction 381
19.2 Characteristics and Requirements 383
19.3 Communication Approaches 385
19.4 Synchronization Approaches 388
19.5 Conclusions 395

20 Multimedia Database Systems **397**

by A. Ghafoor, P.B. Berra

20.1 Introduction 397
20.2 Characteristics of Multimedia Data 398
20.2.1 Text and Formatted Data 398
20.2.2 Audio and Music Data 398
20.2.3 Images and Pictures Data 398
20.2.4 Full-Motion Video Data 399
20.3 Notion of Time for Multimedia Data 399
20.3.1 The Temporal Synchronization Problem 399
20.3.2 Modeling Time 401
20.4 Conceptual Models for Multimedia Objects 402
20.4.1 Graphical Models 402
20.4.2 Petri-Net Models 403
20.4.3 Object-Oriented Models 404
20.4.4 Language Based Models 405
20.4.5 Temporal Abstraction Models 405
20.4.6 Database Models for Multimedia Synchronization 406
20.5 Some Multimedia Database Systems 407
20.5.1 Image Database 407
20.5.2 Audio Database 409
20.6 Challenges in Multimedia Database 410
20.7 Conclusion 411

Bibliography **413**

Chapter 1

COMPOSE: A System For Composite Specification And Detection

Narain Gehani*, H. V. Jagadish†, O. Shmueli‡

1.1 Introduction

An "event" is a happening of interest. Events can be simple such as, the stock price going above a certain price, the beginning of a transaction, the update of an object, or the temperature going above a specified limit. New events can also be formed as a combination of other events, for example, three successive discount rate cuts without an intervening increase, all withdrawals following a large deposit, and the temperature going above a specified limit and staying there for more than some time period. We call such events "composite events".

We have developed a model for specifying composite events [216, 215]. We were motivated to explore the specification of composite events as part of an effort to design "trigger" facilities for the Ode object database [7, 217]. Triggers are the key facility that distinguishes active databases [138, 524, 40, 378, 556, 395] from passive databases. A trigger consists of an event-action pair. When an event is detected, the associated action is executed.

The use of triggers moves code from the application to the database. This simplifies application writing because the application now does not have to check for the conditions specified by the triggers. Triggers also eliminate duplicate code since the same conditions may have to be checked in multiple applications.

A trigger facility in which triggers fire on the occurrence of composite events is more powerful than one in which triggers fire on the occurrence of simple events because it allows users to write triggers that could not be easily expressed before. Composite event specification is useful for many application domains besides databases:

1. Financial Applications: Trades can be executed in response to an observed pattern of (trading) events in a stock market.

*AT&T Bell Laboratories
†AT&T Bell Laboratories
‡AT&T Bell Laboratories and Technion -Israel Institute of Technology

2. Fraud Detection: Particular sequences of credit card purchases may point to fraudulent use.

3. Production Management: Particular sequences of defects could indicate difficulties that must be brought to the attention of a supervisor.

Composite events are specified as event expressions. Our basic notation for specifying composite events has the same expressive power as regular expressions. Thus the occurrence of a composite event is detected by a finite automaton that implements the event expression. Despite the equivalence of expressive power, our notation is specially suited for specifying composite events. For example, it allows for the easy specification of composite events whose components can overlap and allows uninteresting events to be screened out.

We extend our basic notation with "masks", correlation variables, and parameters, thereby stepping beyond the domain of regular expressions. However, we can still implement event expressions that use these facilities by using automata augmented with "mask" events and by using "generic" automata. This allows us to use finite automata optimization techniques to generate efficient implementations for recognizing the occurrence of composite events.

We have built a prototype system, COMPOSE, for specifying and detecting composite events. A real-time stock trade feed is used to experiment with specifying and detecting stock market related events.

In this chapter, we describe how composite events are specified, illustrate composite event specification, give an overview of COMPOSE, and describe the construction of the finite automata.

1.2 Event Expressions

Primitive events are events that are known to or supported by the database system. Examples of some primitive events, in object-oriented databases [215], are object manipulation actions such as creation, deletion, and update or access by an object method (member function). Events can be specified to happen just prior to or just after the above actions. In addition, events can be associated with transactions and specified to happen immediately after a transaction begins, immediately before a transaction attempts to commit, immediately after a transaction commits, immediately before a transaction aborts, and immediately after a transaction aborts. Examples of other events are time events such as clock ticks, the passage of a day, an hour, a second, or some other time unit. Finally, stock trades and the raising or lowering of interest rates are examples of financial events, and company announcements are examples of news events.

Composite events are specified as event expressions. An *event expression* can be $NULL$, any primitive event a, or an expression formed using the basic operators $\wedge$, ! (not), *relative* and *relative+*.

Formally, event expressions are mappings from *histories* (sequences of primitive events) to histories:

$$E : histories \rightarrow histories$$

The result of applying an expression E to a history h, which is also a history, is denoted by $E[h]$.

Here are the semantics of some simple event expressions:

1. $E[null] = null$ for any event E, where *null* is the empty history.
2. $NULL[h] = null$.
3. $a[h]$, where a is a primitive event, is the maximal subset of h composed of all event occurrences of the symbol a.

1.2.1 Basic Operators

Let E and F denote event expressions and h, h_1, h_2 denote event histories. Here are the semantics of expressions formed using the basic operators:

1. $(E \wedge F)[h] = h_1 \cap h_2$ where $h_1 = E[h]$ and $h_2 = F[h]$.
2. $(!E)[h] = (h - E[h])$.
3. $relative(E, F)[h]$ are the event occurrences in h at which F is satisfied assuming that the history started immediately following *some* event occurrence in h at which E takes place.

   Formally, $relative(E, F)[h]$ is defined as follows. Let $E^i[h]$ be the i^{th} event occurrences in $E[h]$; let h_i be obtained from h by deleting all event occurrences before $E^i[h]$. Then $relative(E, F)[h] = \bigcup_{i=1}^{E[h]} F[h_i]$.
4. $relative+(E)[h] = \bigcup_{i=1}^{\infty} relative^i(E)[h]$ where $relative^1(E) = E$ and $relative^i(E) = relative(relative^{i-1}(E), E)$.

1.2.2 Additional Operators

Besides the basic operators, we provide some additional operators that make composite events easier to specify. These operators do not add to the expressive power provided by the basic operators. Consequently, they can be defined in terms of the basic operators.

Let h denote a non-null history, and E, F, and E_i denote event expressions. The new operators are

1. $E \vee F = !(!E \wedge !F)$.
2. *any* denotes the disjunction of all the primitive events.
3. $prior(E, F)$ specifies that an event F that takes place after an event E has taken place. E and F may overlap. Formally, $prior(E, F) = relative(E, any) \wedge F$.
4. $prior(E_1, ..., E_m)$ specifies occurrences, in order, of the events E_1, E_2 ,..., E_m. $prior(E_1, ..., E_m) = prior((prior(E_1, ..., E_m - 1), E_m)$.
5. $sequence(E_1, ..., E_m)$ specifies immediately successive occurrences of the events E_1, E_2 ,..., E_m:
   (a) $sequence(E_1, ..., E_m) = sequence((sequence(E_1, ..., E_m - 1), E_m)$.
   (b) $sequence(E_1, E_2) = relative(E_1, !(relative(any, any))) \wedge E_2$.
   The first operand of the conjunction specifies the first event following event E_1. The second operand specifies that the event specified by the complete event expression must satisfy E_2.
6. *first* identifies the first event in a history.
   $first = !relative(any, any)$.

7. $(E|F)[h] = F[E[h]]$; i.e., F applied to the history produced by E on h. Operator | is called *pipe*, with obvious similarity to the UNIX [§] pipe operator.
8. $(< n > E)$ specifies the n^{th} occurrence of event E. Formally,
   $(< n > E) = ((E|seq(any_1, any_2, ..., any_n))|first)$,
   where each any_i is simply any.
9. $(every < n > E)$ specifies the n^{th}, $2n^{th}$, ..., occurrences of event E. Formally,
   $(every < n > E) = (E|relative + (< n > any))$.
10. $(F / E)[h] = F[h']$ where h' is *null* if $E[h] = null$ and otherwise h' is the history obtained from h by eliminating all the event occurrences before and including $(< 1 > E)[h]$. Formally, $F/E = relative((!prior(E, any) \wedge E), F)$, equivalently, $F/E = relative((E|first), F)$.
11. Suppose that E takes place m times in h. $F / + E [h] = \bigcup_{i=1}^{m} F[h'_i]$. h'_i, $1 <= i < m - 1$, is obtained from h by eliminating all event occurrences before and including event $(< i > E)[h]$ and all event occurrences including and following $(< i + 1 > E)[h]$. h'_m is obtained from h by eliminating all event occurrences before and including event $(< m > E)[h]$.
    E is used to delimit sub-histories of h, where the "delimiter" are event occurrences at which E takes place. F is applied to each such sub-history, and the results of these applications are combined (unioned) to form a single history.
12. $firstAfter(E_1, E_2, F)[h]$ specifies events E_2 that take place relative to the last preceding occurrence of E_1 without an intervening occurrence of F relative to the same E_1. Formally,
$$firstAfter(E_1, E_2, F) = (E_2 \wedge !prior(F, any)) / + E_1$$
13. $before(E) = prior(E, any)$.
14. $happened(E) = E \vee prior(E, any)$.
15. $prefix(E)$ $[h]$ is satisfied by each event occurrence e such that there exists a history h' identical to h up to event occurrence e, and E is satisfied in h' at some event occurrence following e. In other words, $prefix(E)$ is recognized at each event occurrence as long as a possibility exists that an E event will be recognized eventually. This operator is normally used in the form $!prefix(E)$, which occurs as soon as we can be sure that E cannot occur.
16. $E * T$ is a series of zero or more E events followed by a T event.
    $E * T = T \wedge !prior(!E, T)$.

1.2.3 Regular Expressions

Regular expressions are widely used for specifying sequences. The above event expression language (basic operators with or without the additional operators) has the same expressive power as regular expressions [216]. It can be shown that the operators $\wedge$, !, *relative*, and *relative+* constitute a minimal operator set; reducing it will make the expressive power less than that of regular expressions.

COMPOSE event expressions differ from regular expressions in that the focus on ordered sets rather than strings. We believe that our event expression operators are more suitable for specifying composite events. However, since our event expressions are equivalent to regular expressions we can implement them efficiently using finite automata.

[§]UNIX is a trademark of USL

1.3 Examples

We now show how composite events are specified by means of examples.

1.3.1 Simple Examples

1. All occurrences of an event a:
   `a`
2. The 5^{th} occurrence of event deposit:
   `(<5>deposit)`
3. A deposit event followed immediately by a withdraw event:
   `sequence(deposit, withdraw)`
4. A deposit event followed eventually by a withdraw event:
   `prior(deposit, withdraw)`
5. A deposit event followed eventually by withdraw with no intervening interest:
   `relative(deposit, !before(interest)) && withdraw`
6. Event expression that is satisfied when an E occurs provided there is no "non E" event before it. We are essentially recognizing a series of E events:
   `E && !prior(!E, E)`

1.3.2 Discount Rate Cut

The United State Federal Reserve Board raises and lowers a key interest rate, called the discount rate, to control inflation and economic growth. Three or more successive discount rate cuts (D) without an intervening discount rate increase (I) is a rare phenomenon and is of interest to the financial community. Many other events can occur, for example, the prime rate may be cut and the stock market can crash, but these events do not interest us here. Our problem is to write an event expression that is satisfied by such cuts in the discount rate.

Here is an example history with the dots marking the events in the history with the discount rate cut events labeled by D (decrease) and increases labeled by I (increase):

The composite event of interest occurs at the last two D events (marked with #).

Let us create an event expression that specifies a composite event satisfied when three or more successive discount rate cut events D take place without an intervening rate increase event I. We specify this composite event in steps. First, the event expression

```
prior(I, D)
```

specifies D events that are preceded by an I event. Expression

```
!prior(I, D)
```

specifies all events except the occurrences of D that are preceded by I. Expression

```
!prior(I, D) && D
```

specifies D events that are not preceded by an I event.

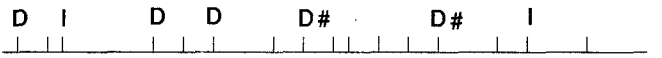


Figure 1.1: Discount rate cut

The expression
`relative(D, !prior(I, D) && D)`

specifies a `D` event followed eventually by another `D` event with no intervening `I` events. This expression gives us a pair of `D` events with no intervening `I` events. Note that in this case, the `relative` operator is used to look at the history starting after a `D` event.

Finally, the event that we are interested in can be specified as

```
relative (relative (D, !prior(I, D) && D), !prior(I, D) && D)
```

The outermost `relative` finds another `D` without a preceding `I` giving us three `D` events without an intervening `I` event.

Using the pipe operator, we can write the composite event for the three successive discount rate cuts simply as

```
(I || D) | sequence(D, D, D)
```

1.3.3 Attributes and Masks

Primitive events can have *attributes*. These could be associated with the event itself, such as the user id, transaction id, parameters to a function invocation (if the event is the function invocation). Event attributes can also be determined from the state of the world at the time the event occurred, such as by reading the system clock or by performing a database query.

Arbitrary predicates can be defined on these attributes and, when false, these predicates "mask" the occurrence of the corresponding event.

As an example of an event with attributes, consider stock trade events which have the form

stock-symbol(current price, intra-day-low, intra-day-high, volume)

Suppose we are interested in a stock trade in which the price of Apple Computer (stock symbol AAPL) has risen by 10% or more compared to it's low of the day. Such a composite event can be written as

```
AAPL(current, low, high, volume) & current > low*1.1
```

where & is the mask operator.

As an other example, define the event "large withdrawal" as the basic event "execution of the method withdrawal" qualified with the mask "withdrawal amount > 1000".

```
withdraw(Item, int q) & q > 1000
```

It is commonly the case in a database that an integrity constraint is to be checked immediately after an update (or creation) of an object. This constraint can be written as a Boolean expression, and used as a mask:

```
(after update | after create) &
Boolean-expression-specifying-integrity-constraint
```

1.3.4 Parameters

Some or all of the attribute can be designated as *parameters*. Our terminology is that attributes are immediate, referring only to the current primitive event, while parameters are attributes that have been saved over one or more event occurrences. While logically there may be little difference between the two, as we shall see in the next section, there is a world of a difference in terms of implementation effort. We have chosen to make this difference evident to the user in the interface that we provide. The implementation difference arises from the fact that non parameter attribute values need only be accessible with the occurrence of the current event while parameter and attribute values, in the worst case, must be available from the "beginning of history".

Once parameters are available, one common requirement is equality between parameter values in two different events in an event expression. Rather than move the automaton and check for equality as a distinct predicate, it is more efficient to perform the equality check immediately and the (second) move as a function of the equality test result.

In the following example, the attribute I is declared as a parameter. The first deposit of an item after a large withdrawal can be written as:

```
first (deposit(Item I, int amt)) /+ withdraw(Item I, int q) & q > 1000
```

The use of the common parameter I indicates that the item withdrawn and deposited must be the same.

1.3.5 Correlation Variables

Correlation variables are used to refer to the same event in the history in different parts of an event expression. Consider the following event expression E that contains the correlation variable x:

$$E = \exists\, x\ prior(b = x,\ c) \wedge !relative(x,\ prior(a,\ c)) \wedge relative(x,\ prior(d,\ c))$$

Consider the following histories (h_1 is a prefix of h_2 which is a prefix of h_3):

$$h_1 = e\ b\ a\ c$$

$$h_2 = e\ b\ a\ c\ d\ b\ c$$

$$h_3 = e\ b\ a\ c\ d\ b\ c\ d\ b\ c$$

We want to determine if E can be satisfied (will trigger) at the last event, a c event, in the above histories. When determining the points at which E can be satisfied in the above histories, the correlation variable x will be associated with a specific b event in each history. In case of h_1, x must be associated with the only b present; E will not trigger at c because $!relative(x,\ prior(a,\ c))$ is not satisfied. In case of h_2, there are two b events. The first has the same problem as in h_1. If we associate x with the second b in h_2, then $relative(x,\ prior(d,\ c))$ is not satisfied. In case of h_3, there are three choices of b with which to associate x. If we choose the first, $!relative(x, prior(a,\ c))$ is not satisfied. If we choose the third, $relative(x,\ prior(d,\ c))$ is not satisfied. However, if we choose to associate x with the second b, then E will trigger at the last c.

To appreciate the role played by x, consider the event expression E', given below, which is the same as E except that the last occurrence of x has been replaced by b.

$$E' = \exists\, x\ prior(b = x,\ c) \wedge !relative(x,\ prior(a,\ c)) \wedge relative(b,\ prior(d,\ c))$$

E' triggers on h_3 in the same way as E. However, it also triggers on h_2, where x is associated with the second b. $relative(b, prior(d,\ c))$ is satisfied now on account of the first b, which does not have to be associated with x.

Finally, the event expression E'', without correlation variables, given below, does not trigger on h_1, h_2, h_3, or any other history of which h_1 is a prefix. The reason is that h_1 has in it the sequence $b\ a\ c$ guaranteeing that the clause $!relative(b,\ prior(a,\ c))$ can never be satisfied.

$$E'' = prior(b,\ c) \wedge !relative(b,\ prior(a,\ c)) \wedge relative(b,\ prior(d,\ c))$$

1.4 Composite Event Detection

We detect composite events by implementing the event expressions as finite automata. These automata are fed as input the primitive events that make up the event expression. The composite event associated with an automaton is said to occur when the automaton reaches an accepting state.

Automata construction is by inductive composition of automata for sub-expressions. Primitive events are expressed in terms of a simple 3 state automaton, one of which is the start state and another the accepting state. From all states, the transition on

event a, the event to be recognized, the transition is to the accepting state. On all other events, the transition is to the non-accepting (non-start) state.

Composition rules are specified for each of the basic event expression operators. These rules are used to compose the automaton for an expression from its sub-expressions. By and large, the additional operators described are rewritten, by compiler transformation, into basic operators. However, specific direct composition rules were developed for a few key additional operators, such as *prefix*.

1.4.1 Design Decisions

1. Automata are kept deterministic at all stages of the construction. Non deterministic automata require that we keep track of the various states computation threads are in, and so are inefficient at run time. Moreover, negation is a problem for non deterministic automata.
2. Reachability analysis is used to eliminate unreachable states in the automaton.
3. The number of states in the automaton is minimized by merging equivalent states. Reachability analysis and state minimization is performed after each stage of the construction so that we always have a minimal deterministic automaton for the sub-expression at hand.
4. Most event symbols are ignored in most states: the number of transitions to be stored explicitly is minimized by explicitly recording only those that cause a transition out of a state, and letting a "self-loop transition" be the default.
5. We may have a large number of events in our system. Individual automata may be interested in small subsets of these events. Letting each automaton "work" on the whole event set is wasteful. So, we have the concept of *local* events specific to an automaton. With each automaton we associate an array *lnames* with as many entries as there are global system wide events. Entry *lnames*[*i*] contains the local name of the global event *i*. If the automaton is not "interested" in global event *i* then *lname*[*i*] contains *otherwise*, where *otherwise* is a local alphabet symbol denoting "all other events".

1.4.2 Masks

A mask predicate is treated like any other operator in terms of automaton construction. Any mask M associated with an event expression E is implemented by modifying the automaton A that implements event expression E. For each accepting state, F, of the automaton A:

1. Two new states $MTrue_F$ and $MFalse_F$ are created.
2. Any transitions from F to other states are copied to $MTrue_F$ and $MFalse_F$. These transitions are deleted from F.
3. F is changed from an accepting state to a non-accepting state and $MTrue_F$ is made an accepting state.
4. Transitions from F to $MTrue_F$ on event $ETrue_M$ and from F to $MFalse_F$ on event $EFalse_M$ are added.

Conceptually, after each event symbol is input, every mask predicate is evaluated, and a pseudo-event $ETrue_i$ or $EFalse_i$ is generated for each mask predicate. Except in a state where this mask is to be evaluated to determine further transition, these

events are ignored. Moreover, the order in which these mask predicates are evaluated is immaterial.

Practically speaking, of course, every mask predicate cannot be evaluated after every event. Instead, we mark states that have outgoing transitions on such pseudo-events. When the automaton reaches such a state, it evaluates the corresponding predicate and makes the necessary transition. The effect obtained is the same as in the conceptual scheme of the previous paragraph.

1.4.3 Generic Automaton For Implementing Events With Parameters

Event expressions with parameters can be thought as representing the union under all possible assignments of values to parameters of the expressions with those values substituted. This definition is used to implement such expressions as follows

1. Given an expression E, convert it to a finite set of expressions $E_1, ..., E_k$ whose union is equivalent to E, such that all variables occurring (as parameters) in any E_i can be assumed to represent distinct values different than any constant mentioned in the expression (i.e. if X and Y are variables and C a constant in some E_i, then X and Y are never suppose to represent the same value or the value C). This is done by rewriting. Thus an expression of the form $relative(F(X), F(4))$ becomes the union of $relative(F(4), F(4))$ and $relative(F(X'), F(4))$, $X' != 4$.
2. Each E_i is now handled separately by associating a deterministic minimal size automaton with E_i, viewing all symbols (such as $F(50)$, $F(X)$, or b) as ordinary alphabet symbols.
3. To handle E_i we keep track separately of each combination of values for parameters in E_i and the state the automaton denoted by this combination is in. We index these automata by the state they are in. We can efficiently perform state transitions in "groups" for all automata in the same state seeing the same symbol.
4. In general, there may be infinitely many sets of values associated with the parameters of an expression E. At any point in time however, we are exposed to finitely many such sets of values as the sequence of events up to this point is finite. So, we handle copies for values combinations we have seen thus far and retain a state for combinations we have not yet seen in a generic way. Specifically, suppose there are two parameters X and Y. If we have only seen $X = 4$ but no Y values yet, we keep the state associated with $4, *$ where $*$ denotes "any domain values not yet seen". We also maintain $*, *$ in that case where $*! = 4$ is assumed. When new domain values are encountered some generic automata are "instantiated" to those values and continue as "independent copies"; the generic copy continues as well under the assumption that the values denoted by a $*$ in a generic automaton is different than all domain values seen for that parameter so far.

1.5 Compose System

A prototype COMPOSE system has been written in Concurrent C/C++ [213], which is a parallel version of C [316] and C++ [563]. The event stream fed to the system consists of real-time stock trades.

The prototype COMPOSE system is structured as follows:

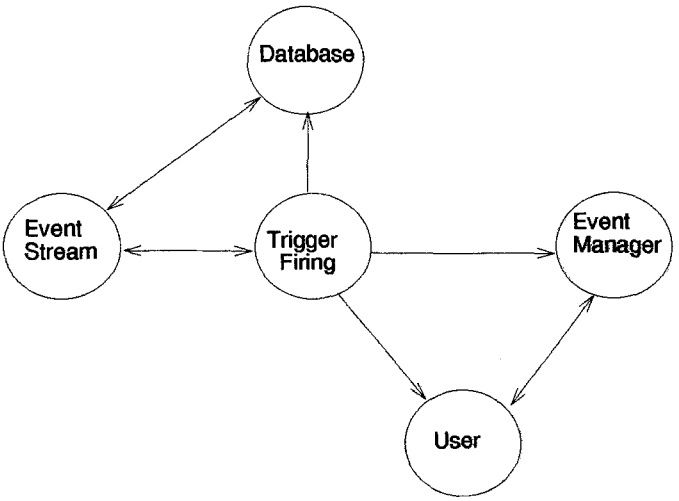


Figure 1.2: The COMPOSE System

The circles represent processes. The event manager process accepts composite event specifications from the user, and the events that occur are fed to it by the event stream processes (one for each event stream). The trigger firing process and the event manager processes share a common data structure which contains the triggers. The event manager constructs and stores the finite automata implementing the triggers and the global event mappings in the shared data structure. The trigger firing process takes the events it receives from the event stream processes and looks for active triggers in the shared data structure and then "feeds" them the events after translating events into numbers using the global mapping. To be precise, the global event numbers are mapped to local event numbers using the local mapping table associated with each automaton.

Since the event manager and trigger firing processes can be simultaneously accessing the shared data, accesses to the shared data are serialized by implemented the shared data structure as a "capsule" [214].

The trigger firing process reports triggers that fire to the user process which is the COMPOSE system's interface to the user.

1.6 Examples of Automata Generated by Compose

The composite event expression for the discount rate cut example shown earlier was `(I || D) | sequence(D, D, D)`

We now show the finite automaton for the above event expression. First, here is the automaton for the event expression `I || D`, which is the first operand of the pipe operator `|` (Figure 3):

The automaton for the second operand of the pipe operator, i.e., `sequence(D, D, D)`, is (Figure 4)

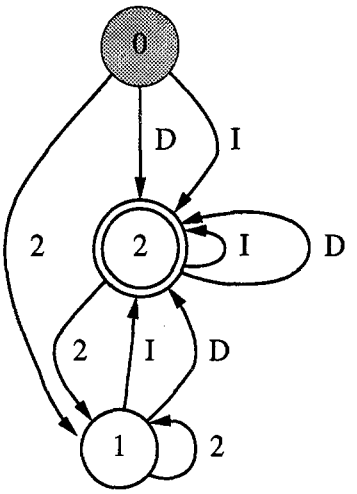


Figure 1.3: Automaton for `(I || D)`

The above two automata are combined to form the automaton for the complete expression `(I || D) | sequence(D, D, D)`. The resulting automaton is (Figure 5)

1.7 Conclusion

Our base event language has the same expressive power as regular expressions, or propositional temporal logic with quantifiers (QPLTL or SOLLO) (see [180]). However, our event specification language is more suitable to specifying trigger events because of its algebraic nature which enables free composition of events into more complex ones.

We implement our event expressions by using finite automata. These automata take as input, on a continuous basis, simple events. Their current state represents a partial (complete) detection of the associated composite event.

Masks and parameters extend our base language to a fragment of temporal logic whose expressive power is beyond that of regular expressions. We implement a mask by adding two additional states to the corresponding automaton without a mask. Transitions to these states take place based on whether the mask predicate is true or false. Evaluation of the mask takes us beyond finite automata. We implement parameters using generic automata. These automata are used to instantiate ordinary automata for each combination of new parameter values. We are currently investigating techniques for optimizing such finite automata [421].

In a distributed database, there may not be a well-defined unique system history. In [287] we discuss how to coordinate the different "views" of the history seen at different sites.

As mentioned earlier, the motivation behind our work was the design of "trigger" facilities for the Ode object database [7, 217].

We plan to integrate the COMPOSE event specification facilities into the Ode trigger facility.

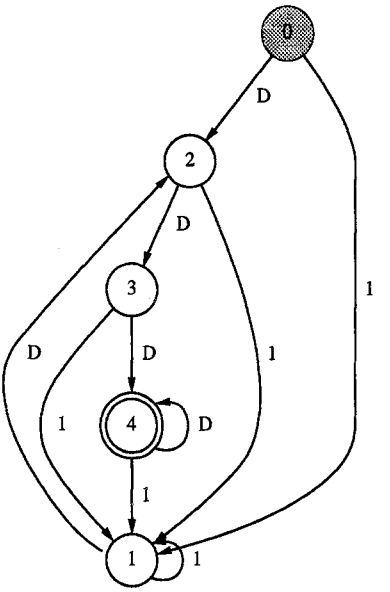


Figure 1.4: Automaton for `sequence(D, D, D)`

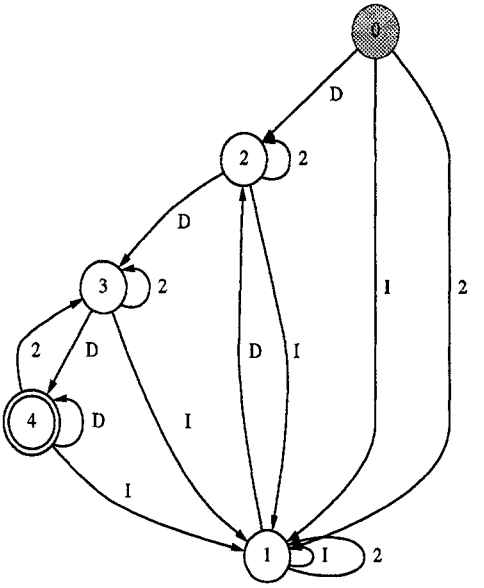


Figure 1.5: Automaton for `(I || D) | sequence(D, D, D)`

Chapter 2

Access Controls in Object-Oriented Database Systems – Some Approaches and Issues

Elisa Bertino*, Sushil Jajodia†, Pierangela Samarati‡

2.1 Introduction

Object-oriented database management systems (OODBMSs) today represent one of the most active areas in both academic and industrial worlds. OODBMSs combine object-oriented programming technology with database technology, thus combining the strengths of both. The need for these systems has been driven by several advanced applications, such as CAD/CAM, cartography, multimedia, for which relational systems have been proved inadequate. A serious problem with these systems is that they do not provide adequate access control mechanisms to prevent unauthorized disclosure of information. They do not provide for the mandatory security [146] and, in most cases, do not even provide adequate discretionary authorization facilities (a notable exception is presented by the ORION/ITASCA system [460]). We can expect, however, that the broadening of application scope of these systems will require them to enforce both mandatory and discretionary security.

Mandatory security policies govern access to information by individuals on the basis of the classifications of *subjects* and *objects* in the system. Objects are passive entities storing information, such as data files, records, field in records, etc. Subjects are active entities that access the objects. Generally, a subject is considered to be an active process operating on behalf of a user. Access classes are associated with every subject and object in the system, and the access of a subject to an object is granted iff some relationship, depending on the access mode, is satisfied between the classifications of the subject and the object.

An *access class* consists of two components: a *security level* and a *set of categories*. The security level is an element of a hierarchically ordered set. The hierarchical set generally considered consists of Top Secret (TS), Secret (S), Confidential (C) and Unclassified (U), where TS > S > C > U. The set of categories is an unordered set (e.g., NATO, Nuclear, Army, etc.). All access classes are partially ordered as follows:

*Dipartimento di Informatica e Scienze dell'Informazione, Università di Genova, Via L.B. Alberti 4, 16132 Genova, Italy.

†Center for Secure Information Systems and Department of Information and Software Systems Engineering, George Mason University, 4400 University Drive, Fairfax, VA 22030-4444, U.S.A.

‡Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39/41, 20135 Milano, Italy.

An access class c_1 dominates ($\geq$) an access class c_2 iff the security level of c_1 is greater than or equal to that of c_2 and the categories of c_1 include those of c_2. Two classes c_1 and c_2 are said to be incomparable if neither $c_1 \geq c_2$ nor $c_2 \geq c_1$ holds.

An access class is assigned to every object and every subject in the system. The security level of the access class associated with an object reflects the sensitivity of the information contained in the object, i.e, the potential damage which could result from unauthorized disclosure of the information. The security level of the access class associated with a user, also called *clearance*, reflects the user's trustworthiness not to disclose sensitive information to users not cleared to see it. Categories are used to provide finer grained security classifications of subjects and objects than classifications provided by security levels alone, and are the basis for enforcing *need-to-know* restrictions.

Access control in mandatory protection system is based on two principles that were formulated by Bell and LaPadula [43] and that are followed by all models enforcing a mandatory security policy. They are:

Simple Security Property: A subject is allowed a read access to an object only if the clearance of the subject dominates the access class of the object.

**-Property:* A subject is allowed a write access to an object only if clearance of the subject is dominated by the access of the object.

These principles, also known as "no-read-up" and "no-write-down," prevent information to flow directly from high level subjects to subjects at lower levels. High assurance systems must additionally protect against illegal indirect information flows through covert or signaling channels [146].

Discretionary protection policies govern the access of users to the information on the basis of the users's identity and the rules that specify, for any user and any object in the system, the types of accesses (e.g., read, write, or execute) the user is allowed on the object. The request of a user to access an object is therefore checked against the specified authorizations; if there exists an authorization stating that the user can access the object in the specific mode, the access is granted, otherwise it is denied. This type of access control is sometimes called a *closed policy* in that a user is not allowed an access unless he or she has been explicitly so authorized. Alternatively, an *open policy* could be applied where all accesses to be denied have to be fully specified, and users are allowed all those accesses for which they have not been explicitly denied. More recent models combine the two policies, allowing to explicitly specify both the accesses to be authorized as well as the accesses to be denied by the users. The resulting access control takes into consideration both authorizations and negations in order to decide whether a request of a user to access an object should be granted.

Discretionary protection models generally allow users to grant other users authorizations to access the objects. There are many policies that can be applied for the administration of authorizations in systems enforcing discretionary protection. Some examples are: *centralized administration* where only some privileged user is allowed to grant and revoke authorizations, *ownership* where the creator of an object is allowed to grant and revoke other users accesses on the object created, and *decentralized administration* where other users can be allowed, at the discretion of the owner of an object, to grant and revoke authorizations on the object.

Some discretionary models also admit the possibility of defining groups of users and specifying access authorizations for the groups. Authorizations specified for a group can be used by any user belonging to the group. Sometimes predicates can also be associated with authorizations, specifying conditions to be satisfied for an

authorizations to be considered valid. These conditions can be defined, for example, in terms of the *time* an access can be executed or in terms of the *content* of the object to be accessed.

Mandatory as well as discretionary security models have been formalized for the protection of information in operating systems and database systems. However, the characteristics of object-oriented data models introduce new protection requirements that make the existing security models not sufficiently adequate for ensuring security of the information.

In particular, applying the Bell-LaPadula paradigm to object-oriented data models is not straightforward for two reasons. First, while this paradigm has proven to be quite effective for modeling security in operating systems, as well as relational databases, it appears somewhat forced when applied to object- oriented systems [288]. The problem is that the notion of an object in the object-oriented data model does not correspond to the Bell-LaPadula notion of an object. The former combines the properties of a passive information repository, represented by attributes and their values, with the properties of an active entity, represented by methods and their invocations. Thus, the object of the object-oriented data model can be thought of as the object and the subject of the Bell-LaPadula paradigm fused into one. The second reason is the increased complexity of the object-oriented data models. An object-oriented data model includes notions such as complex objects and inheritance hierarchies, that must be accounted for when designing a secure object-oriented database model. As for the discretionary policy, the policy has to be extended to take into consideration the characteristics of the object oriented systems such as subtyping, aggregation, and versioning.

In spite of this complexity, the use of an object-oriented approach offers several advantages from the security perspective [397]. The notion of encapsulation, which was originally introduced in object-oriented systems to facilitate modular design, can be used to express security requirements in a way that is comprehensible to the users. Moreover, the notion of information flow in security has a direct and natural representation in terms of message exchanges; messages and their replies are the only instruments of information flow in OODBMSs. The conceptual clarity and simplicity of the model translates into simplicity of design of security mechanisms.

In this chapter we will review the current state of the art in both mandatory and discretionary access controls in OODBMSs. We will also point out some open problems in the field and outline current research directions. The chapter is organized as follows. Section 2.2 summarizes the main concepts of object-oriented data models. Section 2.3 discusses mandatory access control by presenting in detail the *message filter model* proposed recently by Jajodia and Kogan [288], followed by a review of other mandatory access control models. Section 2.4 presents some models for enforcing discretionary access control. Sections 2.5 and 2.6 list some research issues related to mandatory and discretionary protection in OODBMSs. Finally, Section 2.7 draws some conclusions.

2.2 Object-oriented Data Model

An object-oriented model can be characterized by a number of concepts [55]:

- Each real-world entity is modeled by an *object*. Each object is associated with a unique identifier (called OID) that makes the object distinguishable from other objects.
- Each object has a set of *attributes* (properties) and *methods* (operations). The value of an attribute can be an object or a set of objects. The set of attributes

together with the set of methods represent the object structure and behavior, respectively.

- The attribute values represent the object's status. This status is accessed or modified by sending *messages* to the object to invoke the corresponding methods. The set of messages that can be sent to an object is the object *interface.*
- Objects sharing the same structure and behavior are grouped into *classes.* A class represents a template for a set of similar objects. Each object is an instance of some class.
- A class can be defined as a specialization of one or more classes. A class defined as specialization is called a *subclass* and inherits attributes and methods from its *superclass(es).*

There are many variations with respect to the basic concepts, especially when comparing OODBMSs and object-oriented programming languages (OOPLs). The concepts that we have chosen to include are used mainly as a basis for our discussion; we do not claim them to be a definition of the object-oriented paradigm. However, despite all differences, it has been widely recognized that this paradigm offers several advantages. First of all, the separation between an object's status and interfaces allows *clients* to use the services provided by an object without knowing how the services are implemented (*information hiding*). Therefore, an object's implementation may change without impacting other objects or applications using the services provided by the object. The inheritance mechanism favors re-usability of both object interfaces and implementations. Moreover, in most models, a subclass may override the definition of inherited methods and attributes. Therefore, inheritance lets a class specialize another class by *additions* and *substitutions.*

In the database field, the object-oriented paradigm brings other important advantages. First, it allows complex objects to be directly represented by the model, without having to flatten them into tuples, as in the case of relational systems. Second, a traditional DBMS only centralizes data; high-level semantic operations on data are still dispersed among application programs. By contrast, a portion of the high-level semantic operations in an object-oriented database is also centralized. As a consequence, the application programming in object-oriented systems is simplified, since it often consists of invoking and assembling predefined operations – the methods.

The information hiding capability offers, in addition to the previously mentioned advantages, a great potential for data security. Surrounding an object by methods makes it possible to interpose an additional layer between the object and its users. Therefore, arbitrary complex content-based access rules can also be supported.[§] Many aspects and issues in exploiting the object-oriented approach for security will be discussed in the following sections.

In addition to the basic concepts listed above, OODBMSs often provide additional semantic concepts, such as composite objects and versions, that we will briefly discuss in the following. The reason for including those additional concepts is to illustrate their impact on the definition of a discretionary authorization model.

Composite objects

[§]A common distinction found in authorization models is between content-independent access rules, whose enforcement depends only on the object names, and content-dependent access rules, whose enforcements depends on the object information content.

For composite objects we will use the same model as the one given in [320], since this model is quite general. The model distinguishes between two types of references among objects: general, and composite. The latter is used to model the fact that a referenced object is actually a part of (or a component of) a given object. An object and all its components constitute a composite object. Composite references are further refined into shared/exclusive and dependent/independent. A shared composite reference allows the referenced object to be shared among several composite objects, while an exclusive composite reference constrains an object to be component of at most one composite object at a time. A dependent (independent) composite reference models the fact that a component object is dependent (independent) on the existence of the composite object(s) of which it is a part. Since these two dimensions can be combined, four different types of composite references are possible.

Versioned objects

Several versioning models have been proposed in the literature [310]. Here, we present some basic aspects of versioning mechanisms that should be sufficient for discussing the authorization model. In general, a *versioned object* can be seen as a hierarchy of objects, called *version hierarchy*. Each object in a version hierarchy (except for the root object) is derived from another object in the hierarchy by changing the values of one or more attributes of the latter object. Objects in a version hierarchy are first-class objects. Therefore, they have their own object-identifier (OIDs). Information about the version hierarchy is often stored as part of the root object, called *generic object*. Two types of object references are supported in most version models to denote objects within a version hierarchy. The first, called *dynamic reference*, is a reference to the generic version of a version hierarchy. It is used by users who do not require any specific version. The system selects a version (*default version*) to return to users. The default version is in most cases the most recent stable version. Commands are usually available that allow users to change the default version. The second type of reference, called *static*, is a reference to a specific version within the version hierarchy. Another important aspect concerns stability of versions in version hierarchies. In most cases, versioned objects are shared among several users. Mechanisms should be provided so that users receive consistent and stable versions. Most version models distinguish between *transient* and *stable* versions. A transient version can be modified or deleted. However, no versions can be derived from a transient version. A transient version must first be promoted to a stable version before new versions can be derived from it. By contrast, a stable version cannot be modified. However, it can be used to generate new versions.

2.3 Mandatory Access Control

In this section we present an approach to mandatory access control based on the *message filter model* proposed recently by Jajodia and Kogan [288]. The message filter model is an information flow model whose main elements are objects and messages. The chief advantages of this model are its compatibility with the object-oriented data model and the simplicity with which security policies can be stated and enforced.

In the message filter model [288, 289, 487], each object is viewed as a unit of security and, therefore, it is assigned a unique classification. Objects can communicate (and exchange information) only by means of sending messages among themselves. Even a basic object activity such as access to internal attributes, object creation, or invocation

of local methods are to be implemented by having an object send messages to itself; these messages are considered to be primitive messages.¶ This makes the information flow explicit in the form of message exchange among objects and, therefore, easy to control.

Therefore, in the message filter model security is achieved through checking message exchanges among objects. The basic idea of the approach is that *information flow control can be achieved by mediating the flow of messages among objects.* Indeed, in object-oriented systems all information flows are through message exchanges. This approach is very simple, in that it requires the security mechanism to deal only with two elements: objects and messages. It is based on two basic principles governing message exchanges among objects in the system

1. if the sender of the message is at a strictly higher level than the receiver's level, the method is executed by the receiver in restricted mode (that is, no updates can be performed)
2. if the sender of the message is at a strictly lower level than the receiver's level, the method is executed by the receiver in normal mode, but the returned value is *nil.*

The first principle ensures that no write downs occur, whereas the second one ensures that no read ups occur.

The message filter uses the filtering algorithm given in Figure 1 to mediate messages [487]. We assume that o_1 and o_2 are sender and receiver objects, respectively. Also, let t_1 be the method invocation in o_1 that sent the message g_1, and t_2 the method invocation in o_2 on receipt of g_1. The two major cases of the algorithm correspond to whether or not g_1 is a primitive message.

Cases (1) through (4) in Figure 1 deal with non-primitive messages sent between two objects, say o_1 and o_2. In case (1), the sender and the receiver are at the same level. The message and the reply are allowed to pass. The *rlevel* of t_2 will be the same as that of t_1. Note that *rlevel* is a property of a method invocation, rather than a property of an object. We will explain the significance of *rlevel* shortly. In case (2), the levels of o_1 and o_2 are incomparable, and thus the message is blocked and a *nil* reply returned to method t_1. In case (3), the receiver is at a higher level than the sender. The message is passed through; but a *nil* reply is returned to t_1, while the actual reply from t_2 is discarded, thus effectively cutting off the backward flow. For case (4), the receiver is at a lower level than the sender. The message and the reply are allowed to pass. However, the *rlevel* of t_2 (in the receiver object) is set in such a manner as to prevent illegal flows. In other words although a message is allowed to pass from a high-level sender to a low-level receiver, it cannot cause a "write-down" violation because the method invocation in the receiver is restricted from modifying the state of the object or creating a new object (i.e., the method invocation is "memoryless"). Moreover, this restriction is propagated along with further messages sent out by this method invocation to other objects, as far as is needed for security purposes.

The intuitive significance of *rlevel* is that it keeps track of the least upper bound of all objects encountered in a chain of method invocations, going back to the user object at the root of the chain. We can show this by induction on the length of the method invocation chain. To do so, it is also useful to show the related property that

¶It is important to note that the message filter model is a conceptual model telling us *what* needs to be done, rather than *how* it is to be implemented. Reference [487] contains an implementation in which primitive messages do not require any messages.

```
% let g1 = (h1,(p1,...,pk),r) be the message sent from o1 to o2

if o1 ≠ o2 ∨ h1 ∉ {READ, WRITE, CREATE} then case
% i.e., g1 is a non-primitive message
(1)  L(o1) = L(o2) :  % let g1 pass, let reply pass
                      invoke t2 with rlevel(t2) ← rlevel(t1);
                      r ← reply from t2; return r to t1;
(2)  L(o1) <> L(o2) : % block g1, inject NIL reply
                      r ← NIL; return r to t1;
(3)  L(o1) < L(o2) :  % let g1 pass, inject NIL reply, ignore actual reply
                      r ← NIL; return r to t1;
                      invoke t2 with rlevel(t2) ← lub[L(o2),rlevel(t1)];
                      % where lub denotes least upper bound
                      discard reply from t2;
(4)  L(o1) > L(o2) :  % let g1 pass, let reply pass
                      invoke t2 with rlevel(t2) ← rlevel(t1);
                      r ← reply from t2; return r to t1;
end case;

if o1 = o2 ∧ h1 ∈ {READ, WRITE, CREATE} then case
% i.e., g1 is a primitive message
(5)  g1 = (READ,(aj),r) : % allow unconditionally
                      r ← value of aj; return r to t1;
(6)  g1 = (WRITE,(aj,vj),r) : % allow if status of t1 is unrestricted
                      if rlevel(t1) = L(o1)
                        then [aj ← vj; r ← SUCCESS]
                        else r ← FAILURE;
                      return r to t1;
(7)  g1 = (CREATE,(v1,...,vk,Sj),r) : % allow if status of t1 is unrestricted relative to Sj
                      if rlevel(t1) ≤ Sj
                        then [CREATE i with values v1,...,vk and L(i) ← Sj; r ← i]
                        else r ← FAILURE;
                      return r to t1;
end case;
```

Figure 2.1: Message filtering algorithm

rlevel(t_i) $\geq$ L(o_i). For the basis case we assume that the spontaneous method invocation in the root user object has its *rlevel* set to the level of the user object. The induction step follows by inspection of cases (1), (3) and (4) of Figure 1. The use of least upper bound is explicit in case (3). In cases (1) and (4), because of the induction hypothesis, and the relative levels of o_1 and o_2, the assignment of *rlevel* can be equivalently written as in case (3).

We say that a method invocation t_i has *restricted status* if *rlevel*(t_i) $>$ L(o_i). In such cases t_i is not allowed to write to o_i (case (6) of Figure 1), or to create an object with security level below *rlevel*(t_i) (case (7) of Figure 1). A key element of the message filter algorithm is that the restricted status is propagated along with further messages sent out by a method invocation to other objects (exactly so far as is needed for security purposes). This is critical in preventing indirect information flows.

To understand how the message filter algorithm propagates the restricted status on method invocations, it is useful to visualize the generation of a tree of method invocations. The root t_0 is a "spontaneous" method invocation by a user. The restricted method invocations are shown within shaded regions. Suppose t_k is a method for object o_k, and t_n a method for object o_n which resulted due to a message sent from t_k to o_n. The method t_n has a restricted status because L(o_n) $<$ L(o_k). The children and descendants of t_n will continue to have a restricted status until t_s is reached. The method t_s is no longer restricted because L(o_s) $\geq$ L(o_k), and a write by t_s to the state of o_s no longer constitutes a write-down. This is accounted for in the assignment to *rlevel*(t_2) in case (3) of Figure 1.

The variable *rlevel* clearly plays a critical role in determining whether or not the child of a restricted method should itself be restricted. A method invocation potentially obtains information from security levels at or below its own *rlevel*. It follows that a method invocation should only be allowed to record information labeled at levels which dominate its own *rlevel*. For example, consider a message sent from a Secret object to a Confidential one (where Secret > Confidential). The *rlevel* derived for the method invocation at the receiver object will be Secret.

We now discuss the security mediation of primitive messages. Read operations (case (5)) never fail due to security reasons because read-up operations cannot occur. This is because read operations are confined to an object's methods, and their results can only be exported by messages or replies which are filtered by the message filter. Write operations (case (6)) will succeed only if the status of the method invoking the operations is unrestricted. Finally, create operations (case (7)) will succeed only if the *rlevel* of the method invoking the operation is dominated by the level of the created object. If a write or create operation fails, a failure message is sent to the sender. This failure message does not violate security since information flows upwards in level.

There has been relatively little additional work on mandatory security related issues in the object-oriented databases, although some work does exist. Meadows and Landwehr [397] are the first to model mandatory access controls using object-oriented approach, however, their effort is limited to considering the Military Message System. Spooner in [539] takes a preliminary look at the mandatory access control and raises several important concerns. In [313, 312, 582, 583], objects can be multilevel. This means, for example, that an object's attributes can belong to different security levels, which in turn means that the security system must monitor all methods within an object. We consider this to be contrary to the spirit of the object-oriented paradigm. Finally, Millen and Lunt in [397] mention some problems associated with having multilevel objects. In their model, only single-level objects are permitted; however, the notion of subjects is still retained, and subjects are assigned security levels.

2.4 Discretionary Access Control

In this section we present an authorization model for discretionary access control that has been proposed by Rabitti et al. in the ORION/ITASCA framework [460], followed by the extensions to this model that have then been proposed by Bertino and Weigand in [57]. We also illustrate how the authorization models proposed in [53] and [8] exploit the object-oriented approach in the development of a discretionary access control mechanism.

2.4.1 The ORION Authorization Model

This model enforces a discretionary protection policy which takes into consideration the relationships existing among the database objects, the access modes through which objects can be accessed, and the subjects which can access the objects. In particular these relationships are used to derive new authorizations from the authorizations specified by the users. Moreover, the model takes into consideration characteristics of object-oriented systems such as inheritance, composite objects, and versioned objects.

Subjects

The model considers, as subjects, groups of users (*roles*) into which users are organized on the basis of the activities they execute in the organization. A user may belong to more than one role. Roles are related by means of an implication relationship. A role R_1 is in implication relationship with another role R_2 if and only if the authorizations associated with role R_1 subsume the authorizations associated with role R_2. In particular, this corresponds to saying that all users belonging to role R_1 also belong to role R_2. For example, an implication link between the role "accountant" and the role "employee" indicates that accountants are also employees and therefore all authorization specified for the role "employee" are considered valid also for the role "accountant." According to the implication relationship, the set of roles forms a lattice, called a *role lattice*. An example of a role lattice is shown in Figure 2.2.

An arc directed from role R_1 to role R_2 indicates that role R_1 is in implication relationship with role R_2. The root of the lattice (topmost role) corresponds to a role which has the authorizations of any other role in the system. The bottom most role corresponds to a role which has a set of base authorizations executable by any role. On the basis of the implication relationship, and therefore of the role lattice, a partially ordered relationship ($\geq$) is defined on all subjects as follows:

Given two subjects s_i and s_j, $s_i > s_j$ if an implication link exists directed from s_i to s_j in the role lattice; $s_i \geq s_j$ if $s_i = s_j$ or $s_i > s_j$ or there exist subjects $s_1, s_2, \ldots s_n$ such that $s_i > s_1 > s_2 > \ldots > s_n > s_j$.

For the role lattice shown in Figure 2.2, we have Super_user > Chief_accountant > Accountant > Employee.

Objects

The ORION authorization model considers the following objects as objects to be protected: *databases, classes* in the database, *instances* of classes and their components (*attributes, values,* and *methods*). The model also considers *sets of objects* of the same type that have a common root (e.g., the set of instances of a class or the set of values of an attribute) as objects to be protected. In this way authorizations can be specified on the set of objects contained in a given object using the same access modes specified for the object itself, without the need of introducing further access modes. For example,

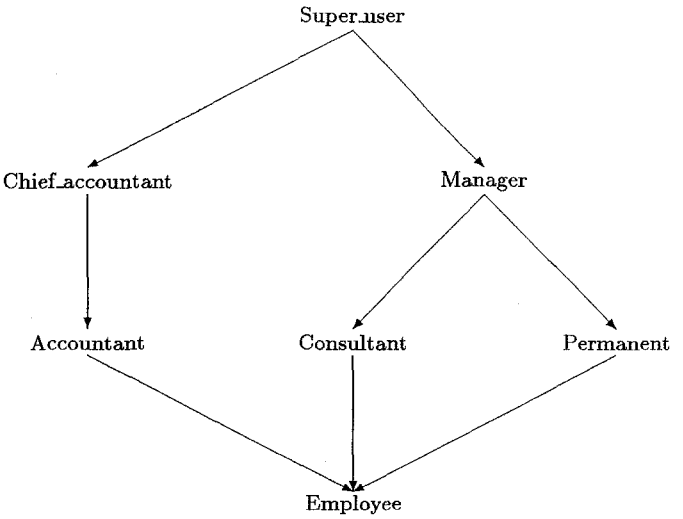


Figure 2.2: An example of a role lattice

it is not necessary to use two different access modes to refer to the read privilege on a class definition and the read privilege on the instances of the class. The meaning of the access mode, and therefore of the authorization, will depend on the type of object to which the authorization is referred.

Like the subjects, objects are also related by means of an implication relationship. An implication link from object o_1 to object o_2 indicates that the authorizations specified on o_1 can also be used on o_2. On the basis of the implication relationship two structures are defined: an *authorization object schema (AOS)*, defining the implication links between object types, and an *authorization object lattice (AOL)*, defining the relationships between the instances of the authorization objects. An AOL is therefore an instance of an AOS for a given system. Every authorization object in the AOL is an instance of only one object type indicated in the AOS. Examples of an authorization object schema[‖] and an authorization object lattice are shown in Figures 2.3 and 2.4, respectively.

On the basis of the AOL, a partially ordered relationship ($\geq$) exists among all objects:

Given two objects o_i and o_j, $o_i > o_j$ if an implication link exists directed from o_i to o_j in the authorization object lattice; $o_i \geq o_j$ if $o_i = o_j$ or $o_i > o_j$ or there exist objects $o_1, o_2, \ldots o_n$ such that $o_i > o_1 > o_2 > \ldots > o_n > o_j$.

Access modes

The model considers the following access modes:

- *Write (W)* to write to an object.
- *Write_Any (WA)* it is analogous to the Write access mode. It allows writes to an

[‖]The nodes shown in italics correspond to authorization objects representing a set of objects of the next lower level.

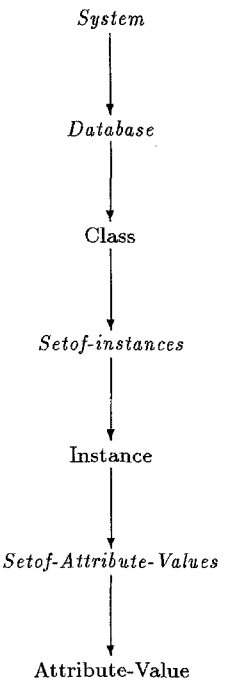


Figure 2.3: An example of an authorization object schema

object. It is considered for completeness purposes in the implication rules based on access modes.

- *Read (R)* to read an object. When referred to a method indicates that the method can be executed.
- *Generate (G)* to create instances of an object.
- *Read_Definition (RD)* to read the definition of an object.

The model does not consider administrative privileges. Any subject that has specific privileges can grant or revoke these privileges of other subjects. Therefore, the authorization for an access implies the authorization to administer (grant and revoke) the access.

Not all access modes are meaningful for every object. In particular, the access modes executable on an object depend on the object type. Given the access modes introduced earlier, an *access authorization matrix (AAM)* states, for every object type and access mode, whether the access mode is executable on objects of that type. An example of an AAM is given in Figure 2.5.

Access modes are related by means of an implication relationship. An implication link from access mode a_1 to access mode a_2 indicates that the access mode a_1 on a given object implies the access mode a_2 on the same object. For example, the implication link between the access mode "write" and "read" indicates that the authorization to write an object implies the authorization to read the same object. On the basis of the

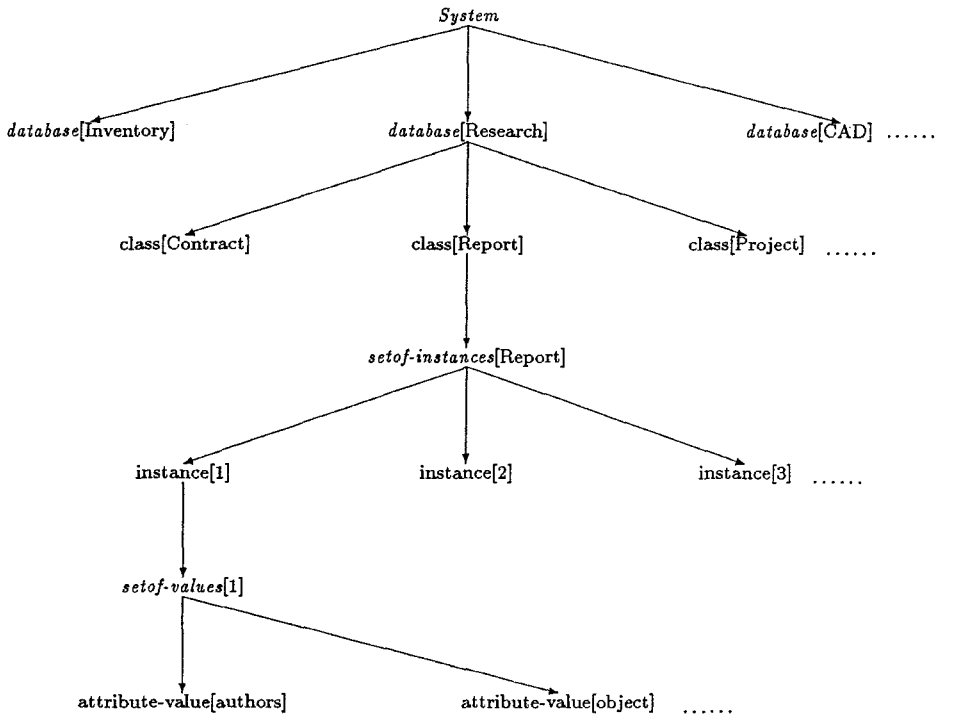


Figure 2.4: An example of an authorization object lattice

implication relationship, access modes form a lattice named *authorization type lattice (ATL)*. This lattice is shown in Figure 2.6.

A link directed from node a_i to node a_j indicates that access mode a_i implies access mode a_j. A partially ordered relationship ($\geq$) is therefore defined on the access modes: Given two access modes a_i and a_j: $a_i > a_j$ if an implication link exists directed from a_i to a_j in the access mode lattice; $a_i \geq a_j$ if $a_i = a_j$ or $a_i > a_j$ or there exist access modes $a_1, a_2, \ldots a_n$ such that $a_i > a_1 > a_2 > \ldots > a_n > a_j$.

Access modes are grouped into three classes: *A.up*, containing all access modes which are propagated from low objects to higher objects in the AOL; *A.down*, containing all access modes which are propagated from high objects to lower objects in the AOL; and *A.nil*, containing all access modes which are not propagated. These groups are as follows:

A.up = {*WA, RD*}. For example, the authorization for the RD mode on the instances of a class, which permits reading of their definition, implies the authorization for the RD mode on the class itself. Analogously, the authorization for the RD mode on a class implies the authorization for the RD mode on the database to which the class belongs.

	W	WA	R	G	RD
System	t	t	t	t	t
Database	t	t	t	t	t
System	t	t	t	t	t
Class	t	t	t	f	t
Setof-Instances	t	t	t	t	t
Instance	t	t	t	f	t
Setof-Attr-Values	t	t	t	f	t
Attribute-Value	t	t	t	f	t

Figure 2.5: An example of an authorization association matrix

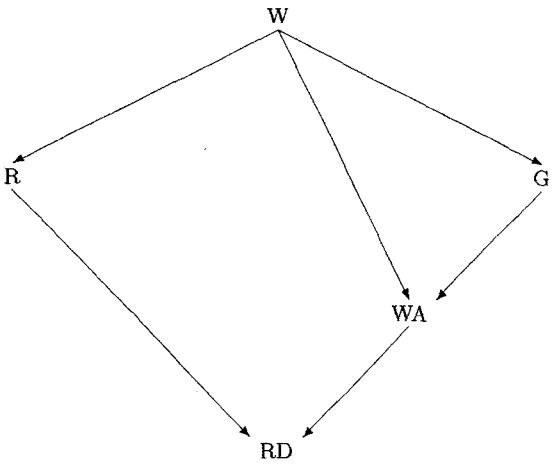


Figure 2.6: An example of an authorization type lattice

A.down = {*W*, *R*}. For example, the authorization for the R mode on a class, which allows reading of the class information, implies the authorization for the R mode, meaning reading, on the instances of the class.

A.nil = {*G*}. The authorization to create objects cannot be propagated among objects related in the AOL.

Implicit and Explicit Authorizations

The ORION authorization model allows the derivation of new authorizations from those specified by the users. The derivation of new authorizations is based on the implication relationships existing among subjects (e.g., a manager can access the information his employees can access), among objects (e.g., the authorization to read a class implies the authorization to read all instances of the class), and among access modes (e.g., the authorization to write an object implies the authorization to read the object), as expressed in the respective lattices.

Authorizations specified by users are called *explicit*, whereas authorizations derived by the system are called *implicit*. Beside this classification, two other orthogonal classifications are introduced. These concern the distinction between *positive* authorizations, stating access privileges, and *negative* authorizations, stating denial of privileges, and between *strong* authorizations which cannot be overwritten by other authorizations and *weak* authorizations which can be overwritten by other authorizations.

Authorizations are grouped into two sets: an *authorization base (AB)* grouping all strong authorizations, both positive and negative, and a *weak authorization base (WAB)* grouping all weak authorizations, both positive and negative. In the following, strong authorizations will be indicated by listing them between round brackets ( ), whereas weak authorizations will be indicated by listing them between square brackets [ ].

A positive strong authorization is described as a triple (s, o, a) indicating that subject s can access object o in access mode a. A negative authorization is described as a triple $(s, o, \neg a)$ indicating that subject s cannot access object o in access mode a.

The system ensures that the set AB, consisting of strong authorizations, is free of (1) inconsistency, i.e., there do not exist two authorizations such that both are derivable from the authorizations in AB and one is the negation of the other, and (2) redundancy, i.e., an authorization should not be in AB if it can be derived from other authorizations already present in the AB.

The set WAB, consisting of all weak authorizations, groups all authorizations, positive and negative, which are classified as weak, i.e., which can be overwritten by strong authorizations. A weak positive authorization is characterized by a triple $[s, o, a]$ stating that subject s can execute access mode a on object o. A weak negative authorization is characterized by a triple $[s, o, \neg a]$ stating that subject s cannot execute access mode a on object o.

To avoid having an access request for which neither a positive authorization nor a negative authorization is derivable from the system, the system ensures the completeness of the authorization bases, i.e., for any possible access that users can request, the corresponding negative or positive authorization can be derived from the authorization bases. Moreover the system ensures that the set of weak authorizations is free of any inconsistency, i.e., an authorization and its negation cannot be both derivable at the same time from the set of weak authorizations.

Unlike AB, redundancy is allowed in WAB, i.e., an authorization can exists in WAB even if it is implied by existing authorizations. Therefore, an authorization already implied by some authorizations in WAB can be inserted in WAB.

A further property, required on the union of WAB and AB, is that a weak authorization must not be present, either as implicit or explicit, as a strong authorization as well. Since weak authorizations are used to complement strong authorizations, the system avoids insertion of those weak authorizations that are already present, either explicitly or implicitly, in the strong authorization base.

Rules for the Derivation of Implicit Authorizations and Access Control

Implication rules determine how new authorizations, called implicit, are derived from the authorizations explicitly defined by the users. These rules are based on the relationships existing among subjects, objects, and authorization types.

Implication rules, summarized in Figure 2.7, are defined for strong authorizations. Implication rules for weak authorizations are derived from the implication rules defined

for strong authorizations. Authorizations derived from strong (weak) authorizations are also considered as strong (weak). The implication relationship among strong authorizations is denoted by $\rightarrow$. The implication relationship between weak authorizations is denoted by $\mapsto$.

Rule 1 $\forall s_l, s_k \in S, o_i, o_j \in O, a_n \in A.\text{down}, a_m \in A :$
$s_k \geq s_l, o_i \geq o_j, a_n \geq a_m \Rightarrow (s_l, o_i, a_n) \rightarrow (s_k, o_j, a_m)$

Rule 2 $\forall s_l, s_k \in S, o_i, o_j \in O, a_n \in A, a_m \in A.\text{up} :$
$s_k \geq s_l, o_i \geq o_j, a_n \geq a_m \Rightarrow (s_l, o_j, a_n) \rightarrow (s_k, o_i, a_m)$

Rule 3 $\forall s_l, s_k \in S, o_i, o_j \in O, a_n \in A.\text{nil}, a_m \in A :$
$s_k \geq s_l, o_i = o_j, a_n \geq a_m \Rightarrow (s_l, o_j, a_n) \rightarrow (s_k, o_i, a_m)$

Rule 4 $\forall s_l, s_k \in S, o_i, o_j \in O, a_n \in A.\text{down}, a_m \in A :$
$s_k \geq s_l, o_i \geq o_j, a_n \geq a_m \Rightarrow (s_k, o_j, \neg a_m) \rightarrow (s_l, o_i, \neg a_n)$

Rule 5 $\forall s_l, s_k \in S, o_i, o_j \in O, a_n \in A, a_m \in A.\text{up} :$
$s_k \geq s_l, o_i \geq o_j, a_n \geq a_m \Rightarrow (s_k, o_i, \neg a_m) \rightarrow (s_l, o_j, \neg a_n)$

Rule 6 $\forall s_l, s_k \in S, o_i, o_j \in O, a_n \in A.\text{nil}, a_m \in A :$
$s_k \geq s_l, o_i = o_j, a_n \geq a_m \Rightarrow (s_k, o_i, \neg a_m) \rightarrow (s_l, o_j, \neg a_n)$

Figure 2.7: Implication rules for strong authorizations

The implication rules for the derivation of strong positive authorizations can be summarized as follows:

- *Authorizations with access mode belonging to A.down are propagated for subjects at higher levels, and for objects and access modes at lower levels as described in the corresponding lattices (Rule 1).* Since every subject, object, and access mode is $\geq$ and $\leq$ itself, this rule allows the derivation of, from an authorization, authorizations involving the same or different subject, object, and access mode.
- *Authorizations with access mode belonging to A.up are propagated for subjects and objects at lower levels, and for access modes at higher levels as described in the corresponding lattices (Rule 2).* Again, since any subject, object, and access mode is $\geq$ and $\leq$ itself, this rule allows us to derive, from an authorization, authorizations involving the same or different subject, object, and access mode.
- *Authorizations with access mode belonging to A.nil, therefore not propagatable in the authorization object lattice, are propagated for subjects at higher level and for access modes at lower level as described in the corresponding lattices (Rule 3).* Therefore, starting from an authorization, new authorizations on the same object, with the same or different user and access mode can be derived.

From these implication rules, according to the property that given two predicates p and q: $p \rightarrow q \Leftrightarrow \neg q \rightarrow \neg p$, analogous implication rules for negative authorizations are defined.

The implication rules for weak authorizations are the same as those for strong authorizations. The only difference is that, since weak authorizations can be overwritten, the derivation of authorizations from an explicit weak authorization stops where another more specific explicit authorization starts.

Access control

Derivation of authorization is used in the access control as follows. Given a request of a subject to exercise an access mode on an object the strong authorizations are examined. If there exist a strong authorization, either explicit or implicit, which authorizes, or denies, the access, then the access is authorized, or denied, respectively. If there does not exist any strong authorization for the access, the weak authorizations are examined, and the access is either granted or denied based of the outcome.

To illustrate how the implication of authorizations work, consider the role lattice shown in Figure 2.2, the AOS shown in Figure 2.3, and the ATL shown in Figure 2.6, and suppose that the strong authorization base contains the authorization (Permanent, database[Research], W). Suppose now that the authorization (Manager, instance[1] of class[Report], R) needs to be checked. Along the subject domain, we have that Manager > Permanent. Along the objects domain we have that database[Research] > class[Report] > instance[1]. Finally, along the authorization type domain we have W > R, with W $\in$ *A.down*. Therefore, by applying Rule 1 we have that (Permanent, database[Research], W) $\rightarrow$ (Manager, instance[1] of class[Report], R). Therefore, the authorization is satisfied.

Inheritance Hierarchies, Composite Objects, and Versions

In this section, we illustrate how the ORION authorization model takes into consideration characteristics of the object oriented systems such as inheritance hierarchy, composite objects, and version.

Inheritance hierarchies

When a class is defined as subclass of another class, there are two approaches which can be taken concerning authorization on instances of the subclass.

The first approach is that the creator of a class should have no implicit authorization on the instance of the subclasses derived from his class by some other user. For example, in reference to the AOL shown in Figure 2.4, if class Technical_Report is defined as a specialization of class Report, the creator of class Report should not be able to read or update the instances of class Technical_Report, unless explicitly authorized for that from the creator of class Technical_Report (or other authorized user). This approach allows users to reuse existing classes without compromising the protection of the subclasses generated.

A second approach is that the creator of a class should have implicit authorizations on instances of a subclass. For instance, in the above example, the creator of class Report will be implicitly authorized to update and read instances of class Technical_Report.

With respect to query processing, the first approach implies that an access whose scope is a class and its subclasses will be evaluated only against those classes for which the user issuing the query has the read authorization, whereas in the second approach, it would be evaluated against the class and all its subclasses.

The ORION authorization model adopts the first approach as default, and supports the second as a user option. This choice is motivated by the reason that under the

second approach, a user wishing to derive a class from another class would not have any privacy on the instances of the subclass (which are readable by the creator of the superclass). Therefore, users could be discouraged from reusing existing classes not taking advantage of the characteristic of inheritance.

When multiple inheritance is allowed, implicit authorizations along the class hierarchy may give rise to conflicts. Conflicts are handled by rejecting insertion of new authorizations when it conflicts with some other authorization already present in the system.

To allow authorizations for generating subclasses, the access mode *subclass-generate (SG)* is added to the set of access modes. If a user is authorized for the SG access mode on some class, the user can define subclasses on it. Access mode SG belongs to the set A.nil, i.e., it is not propagated in the AOL. Moreover, the following relationships hold: W > SG > RD. Given these relationships, authorization along the specialization hierarchy can be derived according to the rules given in the previous section. For example, if a user has the write authorization on a class, the user is implicitly authorized to generate subclasses from the class; if a user has the SG authorization on a class, then the user has implicitly the RD authorization on the class. Indeed, a user, in order to create a subclass from a class C, must be able to read the definition of C. Therefore, the authorization to generate a class from a given class C implies the authorization to read the definition of C.

Composite objects

Composite objects are taken into account in the model by considering a composite as an authorization unit. This allow a single authorization granted on the root of a composite object to be propagated to all components without any additional explicit authorization. This can simply be enforced by representing the composite relationship among objects in the authorization object lattice. The defined implication rules can therefore be used to derive authorizations across composite objects. For example, if a user can read a composite object, then the user is automatically authorized to read the objects which compose it. It should be noticed that the implicit authorization only holds for the objects which belong to the composite object. For example, suppose a class C is defined on class C1 and C2. Access authorization to C's instances implies the authorizations on the instances of C1 and C2 which compose some object of C. No authorizations for instances of C1 and C2 which do not compose any object of C are derived.

In this context, negative authorization may give rise to conflicts in the authorization of implicit authorizations. Therefore, care must be taken that authorization conflicts will not arise. For example, the positive authorization to read a composite and the negative authorization to read one of its component cannot be present at the same time, unless the authorization on the component is a weak authorization and can therefore be overwritten. As in the case of inheritance hierarchy conflicts are avoided by accepting insertion of a new authorization only in case it does not conflict with authorizations already specified.

Versions

Authorizations can also be specified on a versioned object and on individual versions of the object. To represent version hierarchy and enforce derivation of authorization along the hierarchy, the model extends the authorization objects to include generic instances and versions. An implication link is therefore defined between the generic instance of an object and the set of versions of the objects. The implication rules can

then be used to derive new authorizations. For example, a read authorization an a generic instance of an object implies the same authorization on all the versions of the object; a write authorization on the set of versions of an object implies the same authorization on the versions described by the generic instance. The write authorization on the set of versions of an object is also the authorization to create a new version from a working version of the instance. The write authorization on a generic instance allows the user to modify the generic instance (e.g., by changing the default version) and implies the write authorization on the version objects described by the generic instance.

2.4.2 Content-dependent Authorizations

The ORION authorization model has been extended in [53]. In [53] different access modes have been introduced in reference to different object types, and some implication rules for the derivation of implicit authorizations have been revised.

An important extension introduced in [53] is the consideration of content-dependent authorizations, i.e., authorization depending on some properties of the objects to be accessed. Indeed, in [460] a user is either authorized or denied for an access on an object. Instead, in [53] it is possible to specify that a user is allowed (denied) for an access on an object if some conditions on the object are satisfied. Therefore, authorizations are extended to the consideration of conditions which must be verified for the authorization to hold. Conditions can be put on any of the objects' attributes. In particular, they may involve class-attributes, e.g., attributes that characterize the classes themselves and are not inherited by instances, or instance attributes.

For example, consider the AOL shown in Figure 2.4 and suppose that attribute status is added to the attributes of instances of class Report. An authorization could be specified by stating that Employee can read only instances of class Report which have status "released".

The implication rules defined in [460] are applied also to content-dependent authorizations. Authorizations derived from content-dependent authorizations inherit also the conditions upon which the access has to be controlled. For example, it is possible to specify content-dependent authorizations on a class to be evaluated against the instances of the class and to specify content dependent authorizations on a versioned object to be evaluated against all versions of the object.

In the case of composite objects, the situation is a little different. Indeed, composite objects can have components of different classes and their types may be different. Therefore, conditions may not be evaluable on all the components. In this case the conditions are considered only in reference to the component against which they can be applied.

A main issue when dealing with content-dependent authorizations is how to efficiently evaluate conditions associated with authorizations. Since conditions have to be evaluated over object's attributes, which can change over time, conditions have necessarily to be evaluated at run-time, therefore necessarily increasing the response time of the system. In particular, enforcing satisfaction of the conditions expressed in the authorizations by filtering the data prior to the user access, would require a double access to the objects (one to evaluate the conditions and the other to satisfy the user query). The solution considered in relational database system is to simply add conditions expressed in the authorization to the user query. This approach, known as *query modification mechanism* has the advantage of ensuring the satisfaction of the protection requirements and not overloading the access control.

In object-oriented databases, where objects are accessed through methods, which can be nested, implementation of content-dependent condition is not straightforward. A possible solution would be incorporating conditions in the method. This approach has the drawback of having method specification to be dependent on authorizations, therefore a change in the authorizations would require to change specification of methods.

A further aspect that must be taken into consideration when considering content-dependent authorization is that since satisfaction of the conditions in the authorizations depends on the values of the objects' attributes and can therefore change over the time is more difficult to ensure the consistency and completeness of the authorizations. For example, consider the class Documents having as attributes date and status. Suppose then that subject Employee has to be authorized at the same time for both the positive authorization to read all documents with date later than "March 3, 1992" and the negative authorization to read all objects with status "protected". If no object exist with status protected and date later than "March 3,1992", the authorization state is consistent. However, since attribute values can change and new objects can be added authorizations may become inconsistent. Consistency and completeness criteria and mechanism to satisfy them have therefore to be extended to the consideration of content-dependent authorizations.

2.4.3 Accessing Objects Through Methods

The model presented in [460] and extended in [56] takes into consideration many of the characteristics of object-oriented data models such as inheritance hierarchy, versions and composite objects. However, it does not exploit the potential of encapsulation typical of the object-oriented approach. In fact, all accesses made during a method execution are further checked against the user who invoked (directly or indirectly) the methods. For example, if during the execution of a method invoked by a user, an attempt is made to modify the attribute of an instance, the authorization for the user to update the attribute must be checked. In some cases, where encapsulation is meant to provide protection, it is desirable not to give the users the authorization to execute some accesses directly but allow at the same time the accesses to the users through the execution of some method. For example, users may not be authorized to write an attribute of an object but can be authorized to run a method which, during its execution, modifies the attribute. Therefore, since users should not be directly authorized for the access, no authorization for the users to execute the access executed by the method should be provided. An authorization model which takes into account this principle has been presented in [57].

According to many object-oriented languages, the model distinguishes between *public* and *private* methods. Private methods of an object can be invoked only by other methods of the same object, whereas public methods can also be invoked directly by the users of the object (i.e., end- users, application programs, other objects). That previous notion of public/private methods has been further refined by allowing methods to be defined public with respect to some other methods, of the same or different objects. The method is then considered private for all methods for which it has not been explicitly defined as public. In this way, it is possible to specify that some methods cannot be invoked directly by the users of the object but can be invoked during the execution of some other methods. The declaration of the methods for which a specific method is public is provided as part of the class definition to which the method belongs. The set of methods for which a specific method is public is called the *invocation scope* of the

method.

The model is based on authorizations for users to execute methods on objects. Authorizations specify, for each user, the set of methods the user can invoke on which objects. Therefore, authorizations have the form $< o, u, m >$, where o is an object, u is a user, m is a method. Such a tuple specifies that user u is authorized to execute method m on object o. Authorizations can be specified only on public methods, i.e., on methods directly invokable by end-users. In order for a user to execute a method, the method must be public for the end-users and the user must have the authorization to execute the method. If both these conditions are satisfied, the user can execute the method. However, the fact that the user is authorized to run a method does not imply that the user will be able to always execute all actions that are part of the method. Indeed, other methods can be invoked during execution of the method called by the user, and, therefore, several access controls may be performed during the execution. In particular, if during the execution of a method m another method m' is invoked, the invocation is allowed if either m' is public for end-users and the user has the authorization for it or m' is private for end-users and m belongs to the invocation scope of m'.

The model allows users to grant other users authorizations to execute methods. A user can grant such authorizations on an object if the user is the creator or one of the owners of the object. Each object is associated with a creator, i.e., the user who created the object, and some owners. The creator of an object is always unique but can change during the life-time of the object. In fact, it is allowed for the creator of an object to give the privilege of being creator to some other user. Since the creator must be unique, so doing the first user looses the creator privilege on the object, which is passed to the other user. The creator can also add and delete owners for the object. Any owner of an object can grant and revoke authorizations to execute methods on the object to other users. A user can revoke only authorizations that he granted.

The model introduces also the notion of *protection mode* for method execution authorizations. If user u grants user u' the authorization to execute method m in protection mode, then when u' executes m, all invocations of methods public for end-users made by m are checked for authorizations not against u', who called the method, but against u, i.e., against the user who granted u' the authorization on the method**. In this way users can grant other users the privilege of executing some methods on an object not directly, but by using some other methods.

A model applying a similar approach has been proposed in the context of the Iris DBMS [8]. There, objects (and their data) are encapsulated by a set of functions, i.e., to access an object, users call the appropriate functions ††. Authorizations specify for every user, the set of functions the user is allowed to call. Authorizations can be referred to single users as well to groups of users. A user can belong to zero or more groups, and groups can be nested.

The model supports the concept of ownership, in particular, the user who creates a function is considered the *owner* of the function and can grant other users the authorization to call the function. This authorization can also be given with the grant option. If a user has the grant option on the authorization to call a function, the user can grant other users the authorization to call the function and the grant option on

**The concept of protection mode is very similar to the set user-id on execution concept considered in the Unix operating system.

††In the Iris data model both attributes and methods are represented as functions. In particular, attributes are defined as stored functions, while methods are defined as derived functions.

it. The grant privilege is also enforced as a function: the authorization for the grant privilege on a function f is represented by the authorization to call the grant function with f as argument.

Functions can be defined on other functions. Authorizations on derived functions can be defined as *static* or *dynamic*. If a user has the dynamic authorization on a derived function, in order for the user to successfully call the function, the user must have the call privilege on all the underlying functions. By contrast, in order to call a derived function on which he has the a static authorization, a user does not need to have the call authorization to the underlying functions. The concepts of static and dynamic authorizations correspond to the concepts of protection and non-protection execution modes for methods discussed earlier [57].

When a user creates a derived function, the user must specify whether the authorizations on the defining functions must be checked statically or dynamically. In either case, the creator of the derived function must have the call authority on all the underlying functions. If the creator specifies that the function must have dynamic authorization, the user can grant other users the authorization to call the function. By contrast, if the function is specified to have static authorizations, the user can grant other users the authorization to call the derived function only if he has the grant privilege on all the underlying functions. Derived functions can also be used to support content-dependent authorizations. In this case, users are not authorized directly for a function, but on a function derived from it which enforce some constraints. For example, suppose to have an class Employee storing information about the employees and a function "Salary", defined on it, returning the salary. Thought some employees can be authorized to read the salary of everybody, some employees could be restricted to see their own salary. This condition can easily be enforced by defining a derived function "Self_Salary" which takes into consideration the caller of the function and calls function "Salary" to return the user's salary. Since users cannot be authorized to call directly function "Salary", the authorization to be specified on function "Self_Salary" is a static one. However, the application of derived function to enforce content-dependent authorizations has the drawback of embedding authorizations in the function implementation, therefore a change of authorizations would imply a change in the implementation of some derived function.

The authorization model takes also into consideration the characteristic of polymorphism of object-oriented systems. Polymorphism allows to specify functions, called *generic functions* which have associated a set of *specific functions* that are defined on different types [‡‡]. When a generic function is called, a specific function is selected for invocation (*late binding*). Authorizations can be specified on generic or on specific functions. A user authorized to call a generic function is automatically authorized to call all specific functions of that generic function. When a user calls a generic function, the corresponding specific function is selected and the user is allowed for it only if he has the authorization on the specific function. The specific function can be selected regardless of the user's authorizations (*authorization-independent resolution*) or by taking into account the user's authorizations (*authorization-dependent resolution*). The authorization-dependent resolution has the disadvantage that the query semantics is in this case dependent on the authorization-policies.

Functions can also be specified as having a guard function. If a function has a guard function associated with it, the function can be executed only if the guard function

[‡‡]The concept of type in the Iris data model is equivalent to the concept of class in discussed in Section 2.

returns value "true". Guard functions can therefore be used to specify conditions which have to be satisfied for the users to execute some function. Since guards enforce conditions by evaluating them prior to the execution of the function they control, they are really useful for evaluating preconditions, i.e., conditions independent on the values returned by the controlled function. Indeed, in order to enforce conditions on values returned by the controlled function, the controlled function itself should be called and its results evaluated by the guard.

Another concept introduced by the model is that of *proxy* functions. Proxy functions provide different implementations of specific functions for different users. A function may have associated several proxy functions. When a user calls a function, the appropriate proxy is executed in place of the original function. Therefore, the result of a function may change depending on the user calling it. Proxy functions have the advantage of allowing to enforce constraints on function execution by users without any impact on the function implementation.

2.5 Research Issues in Mandatory Access Control

We now discuss ongoing research aiming at extending in various directions the message filter model presented in Section 2.3.

2.5.1 Modeling Multilevel Entities as Single-level Objects

In the message filter model, all objects are single-level in that a unique classification is associated with the entire object. This constraint is essential in order to make the security monitor small enough so that it can be easily verified. However, entities in real world are often multilevel: some entities may have attributes of different levels of security. Much modeling flexibility would be lost if multilevel entities could not be represented in the database.

A preliminary approach that maps multilevel entities in terms of single-level objects is given in [288]. It is based on using inheritance hierarchies. Unfortunately, this approach suffers from several problems. First, it leads to a replication of information. Since a multilevel entity is modeled as several single-level objects in a class hierarchy, some attributes of high level objects are replicas of attributes of low level objects (because of inheritance). Second, if not carefully monitored, updates may lead to mutual inconsistency of replicated data. To illustrate, suppose that an update is performed on an attribute of a low level object. This update cannot be propagated to the corresponding attribute of the high level object because the low level object does not store any reference to the high level object. (Note that although writing up is permitted under the Bell-LaPadula paradigm, an important requirement is that the existence of high level objects must be hidden from low level objects. Therefore, low levels objects cannot have references to high level objects.) The third problem with this approach is that the notion of inheritance hierarchy becomes overloaded since it is used both for conceptual specialization and for supporting multilevel entities.

To solve the above problems, a different approach based on composite objects and delegation has been recently proposed by Bertino and Jajodia [54]. The notion of composite object is a modeling construct that allows to consider an object and a set of component objects as a single object [320]. Delegation allows an object to perform

some of its functions by simply delegating their executions to other objects. In this approach, an entity E, with attributes of n_l different levels of security, is modeled by a number n_l of single-level objects. An object O_{l_i} would contain all attributes having security level l_i. Moreover an object O_{l_i} would contain a composite attribute whose value is the object $O_{l_{i-1}}$. Thus, a multilevel entity would be decomposed in several single-level objects such that an object at a level l_i has as component objects the objects whose security levels are lower than l_i. Whenever an object needs to retrieve values of attributes of lower level objects, this object can delegate the appropriate component object the execution of the retrieval operations. An attractive aspect of this approach is that it allows the same object interfaces. An interface of an object is the set of messages that are defined for the object. to be provided to users as if multilevel objects were directly provided, while retaining at the same time the simplicity of the message filter approach and of the single- level objects.

There are several aspects of composite references that have been refined to take into account security requirements. As previously discussed, different types of composite references have been identified by Kim, Bertino, and Garza [320]. They can be categorized as follows:

1. exclusive dependent reference
   if an object O is component of an object O', it cannot be component of another object; moreover if O' is removed, O is also removed
2. exclusive independent reference
   if an object O is component of an object O', it cannot be component of another object; the deletion of O' does not imply the deletion of O
3. shared dependent reference
   an object O can be component of several objects; O is removed when all parents objects, on which O depends for existence, are removed
4. shared independent reference
   an object O can be component of several objects; the deletion of the parent object does not imply the deletion of O.

Some of these categories may result in violations of security requirements. For example, the exclusivity constrains can be used by a low user to infer the existence of a high object. To overcome this problem, two additional forms of exclusive composite references have been introduced. The first form consists of a composite reference which is exclusive with respect to a class. That is, no two instances of the same class may share a component, however, there could be instances of other classes with references to that component. The second form is similar to the first, with the difference that the exclusivity constraint is with respect to a class hierarchy. That is, no two members of the same class can share a reference to the same component. (The members of a class are the instances of the class and the instances of all its subclasses.) The motivation for introducing those additional form of exclusivity constraints is to support some form of semantic integrity for composite objects. The approach proposed by Bertino and Jajodia [54] also covers multilevel entity types that are organized in specialization hierarchies, including the case of multiple direct supertypes.

Finally, it is important to note that while the use of composite objects combined with inheritance hierarchies allow to model a large variety of application entities, when dealing with real applications the number of entity types and specialization hierarchies among them can be quite large. Therefore, it is crucial that the process of generating

an object-oriented schema be supported by some automatic tool. A security specification language has been proposed by Bertino and Jajodia [54] whose purpose is to describe the entity types and the specialization hierarchies together with their security requirements. Those specification can then be translated in terms of an object-oriented schema through a set of translation rules, based on the approach we have previously discussed.

2.5.2 Object Updates and Secure Garbage Collection Mechanisms

Under the approach described in the previous subsection, updates do not pose any obstacles to security. If an object o is part of some composite object, any updates to the attributes of o are directly visible to the composite object.

By contrast, the delete problem is more difficult to deal with. There are basically two ways in which the delete operations have been implemented in various OODBMSs: systems allowing users to perform explicit delete operations (like ORION [319], and Iris [189]), and systems using a garbage collection mechanism to remove objects that are no longer reachable from other objects (like GemStone [79] and O_2 [150]). In systems belonging to the second category, an explicit delete operation is not available to the users.

Systems with explicit delete operations allow an object to be deleted even if there are references to it. If a message is sent to a deleted object, the system returns a notification to the invoker object. Therefore, the invoker object must be ready to deal with the exception arising from a dangling reference. This approach is used by the ORION system, and it is also suggested by Zdonik [642]. Note that in those systems, OIDs of deleted objects are not re-used. This approach works well with the composite object approach. For example, consider objects o_1 and o_2, such that o_2 is a component of o_1. If object o_2 (a component of object o_1) is removed, the next time a message is send from object o_1 to object o_2, object o_1 will be notified that the referenced object does not exist.

The above approach has been refined in two directions. The first, called *upward cascading delete*, is similar to the approach proposed by Jajodia and Sandhu [292] for the delete operations in the multilevel relational secure model. In that approach, each time a tuple t of a given security level l is removed, all polyinstatiated tuples corresponding to t and having a security level greater than l are also removed. In our framework, this approach means that the deletion of a component should cause the deletion of its parent object. For example, if object o_2 (a component of object o_1) is removed, then also object o_1 should be removed. Note, however, that object o_2 does not know the OID of object o_1 (because o_2 is an object whose security level is lower than the one of o_1). Therefore this approach cannot be implemented by simply having object o_2 sending a delete message to object o_1.

The second approach, called here *conservative delete*, is the opposite of the previous one, in that it aims at preserving information for high level objects. Under this approach, the low level object would still be deleted; however, a new corresponding high level object would be created. For example, if object o_2 (a component of object o_1) is removed, then a new object $o_{2'}$ is created having the same attribute values as o_2 but having the same security level as o_1. Note that o_1 will still be notified that o_2 has been removed. However, all information contained in o_2 will be still available to object o_1.

The approach for supporting both the previous delete modalities is based on using the message filter as an active component; it will need to notify objects at high levels that events have occurred concerning related objects at low levels.

In systems based on garbage collection an object is automatically removed by the system when it is no longer referenced by any other object. This approach, if not properly modified, would cause some problems in a secure environment. Indeed, a low object would not be removed if a reference exists from a high object to this low object. Therefore, a low user may infer the existence of a high object referencing the low object. Note that even though the low user will not be able to infer the OID of the high object, a signaling channel could be established. Another serious drawback is that the garbage collector would have to access objects at various levels of security. This would require the garbage collector to be a trusted component.

We are investigating a different approach which does not require the garbage collector to be trusted. The approach requires a garbage collector for each security level. The garbage collector associated with a level l removes an object at level l only if all references from other objects at level l have been removed. Since the garbage collector at level l does not see references from objects at levels higher than l (because it does not see those objects at all), it will remove an object o at level l as soon as all references from objects at the same level as o have been removed. Note that removing a reference from a high object to a lower object cannot cause the low object to be removed. Indeed, suppose that removing a reference from a high object o to a low object o' causes o' to be removed. This means that the reference from o to o' is the last existing reference to o'. However, this situation cannot arise because the garbage collector at the level of o' would remove o' as soon as the last reference to o' from an object at the same level has been removed. Therefore, if a reference from a high object to a low object is removed, we have two cases: (i) the low object has already been removed; (ii) the low object has another reference from its same level and then it is not removed. In both cases, the removal of the reference from the high object does not cause any change in the status of the low object. It is important to point out that information needed by the garbage collectors are also partitioned on the basis of security. This means that the information that a high object has a reference to a low object is kept at high level. This is automatically achieved because putting a reference from an object o to an object o', where the level of o is greater than the level of o' can only be executed by a subject with the same level at o. Therefore, all information generated as side-effects of this update (such as the information for garbage collection) are classified at the same level of o.

Note that this approach causes the problem of dangling references. Indeed, a low object can be removed even if it has some references from high level objects. One possibility is to allow dangling references and to return a notification, whenever an object sends a message to a low removed component which has been removed, as in the case of explicit delete. Another possibility is to use the approaches of upward cascading delete and conservative delete. Both those approaches, however, require that all deletions issued by the garbage collector pass through the message filter. This is automatically achieved if the garbage collector internally uses messages.

2.5.3 Polyinstantiation

The problem of polyinstantiation has been studied in the framework of multilevel relational data model [292]. A multilevel relation is a relation containing tuples that have different security levels. Polyinstantiation arises when there are two or more

tuples in a multilevel relation having the same value for the primary key. Enforcing the uniqueness of key value in multilevel relations would lead to a covert channel. A possible solution consists of requiring that the key be unique not with respect to the entire relation, but with respect to the set of tuples having the same security levels.

As discussed by Jajodia and Sandhu [292] tuples at different levels in a multilevel relation are related. Indeed, each instance at an access class c represents the version of reality appropriate for the access class c. It is important to note that a user with clearance for a given level may see all tuples of levels equal or lower. One problem is therefore that the user must understand which low level information are correct and which one have been inserted to provide a cover story, and thus may be discarded. A consequence of this fact is that discrimination of correct information against cover information is left to the application programming.

We are currently investigating approaches to this problem in the framework of object-oriented data models. In particular, we are investigating the use of methods as a way to embed and centralize the appropriate knowledge for distinguishing between correct information and cover stories. As an example consider an entity type 'Employee' with attributes Name, Skill, Salary, and Age. Furthermore suppose that Name and Age are unclassified, while Salary is classified. Moreover, suppose that Skill is a multivalued attributes. Remember that in object-oriented data models multivalued attributes can be directly represented, without any need for normalization. that can assume both classified and unclassified values (for example, the fact that an employee has certain skills is secret, while employee's other skills are not). A possible design for the entity type 'Employee' in terms of single-level object would be by defining two objects. A first object o, whose level is unclassified, would contain the attributes Name, Age, and Skill. The Skill value for the unclassified object will contain only those skills that do not need to be kept secret. A second object o', whose level is classified, would contain as attributes Salary and Skill, and in addition a composite attribute containing a reference to object o (according to the composite object model defined in the first subsection). Note that the Skill attribute in the classified object (i.e. o') would only contain the skills that must be kept secret. Now suppose that the unclassified skills are not a cover story. In this case, whenever the values of the attribute Skills must be retrieved from the classified object (i.e. o'), the method retrieving the skills must retrieve the values of the attribute Skills from the unclassified object and unioning them with the values of the attribute Skills from the classified object. By contrast, if the classified skills are a cover story, the method retrieving the skills from the classified object will only need to retrieve the values of the attribute Skills from the classified object. This example shows that the encapsulation feature of object-oriented data model actually allows to shield the applications from having to deal with discriminating correct information against cover information. This discrimination is embedded into the methods encapsulating the object. Note that this solution can be improved by also considering methods expressed in a declarative language. This makes it easier to formulate, understand, and manipulate the discrimination criteria for cover information.

2.5.4 Comparison With Relevant Work

The problem of security in object-oriented databases has been previously addressed by Millen and Lunt [397] and by Thuraisingham [583]. The approach of Millen and Lunt [397] is based on single-level objects. The strategy proposed by Millen and Lunt for handling multi-level entities is based on using references to relate objects corresponding to the same entity. Our approach is based on composite objects, and therefore is simi-

lar, since composite objects are obtained by imposing the part-of semantics on normal references [320]. However, our approach differs in several aspects. First, we make full use of the features of object-oriented data models by showing how, through the use of methods, it is possible to define objects which, even though they are single-levels, are able to provide the same interfaces, as if multi-level objects were directly supported. Moreover, we have introduced some extensions to the composite object model to better modeling the notion of exclusive references, so that no security breaks are introduced. Then, we have provided an extensive analysis of the use of composite objects when multi-level entities are organized in type hierarchies, taking into account also the case of multiple direct supertypes. Moreover, we have investigated the object delete problem. The approach proposed by Thuraisingham [583] mainly discusses rules stating the security policy that must hold among the various objects in an object-oriented database, such as for example that the security level of a subclass must dominate the security levels of its superclasses. However, no discussion is presented on the additional complexity of the security monitor due to the enforcement of the security policy rules. Moreover, [583] does not discuss the problem of handling multi-level entities.

2.6 Research Issues in Discretionary Access Control

An important research issue is whether the content-dependent authorization mechanism, illustrated in Subsection 4.2, is redundant when user-defined methods implement authorization rules as part of their execution. A main difference is that content-dependent authorization rules defined by a constraint language, like one mentioned in Subsection 4.2, are declarative, while authorization rules defined as part of methods are expressed in an imperative language. The usage of the constraint language simplifies the definition of authorization rules by users, and saves the users from writing several methods. However, the expressive power of the constraint language is limited with respect to the expressive power of a general programming language. Therefore, both declarative content-dependent and procedural content-dependent authorizations seem to be useful. However, more investigations are needed on this question. In particular, there is the need of a comprehensive formal model of discretionary authorization for object-oriented databases encompassing both content-independent and content-dependent authorizations. The model should also address administration and ownership issues that are not addressed in the model defined for the ORION system [460]. Related to this there is the definition of methodological guidances supporting the authorization administrators and database designers in the task of designing the proper authorization rules for a given database.

Another relevant problem with the discretionary authorization models previously discussed, as well as other discretionary authorization models defined for relational systems, is that their implementations do not provide assurance against Trojan horses. As discussed in [308], Trojan horses require relatively simple mechanisms to subvert the discretionary protection mechanisms. For example, a malicious user wishing to illicitly access some data offers an attractive program to a user who is authorized to access these data. The program contains some code that performs the advertised service for the second user, while simultaneously performing the illegal action wished by the first user. The program could for example copy the data into a file to which the first user is authorized to read.

We are currently looking at ways of providing protection from Trojan horse attacks in mechanisms meant for discretionary access controls. The approach we are investigating is based on keeping for each object an access list and on using the message filter as a trusted agent able to verify that information flows among the objects in the system are legal. Since in object-oriented systems all exchanges of information among objects is based on messages, it should be possible to determine whether illicit transmission of information is about to take place. It is also important to note that in an object-oriented database system some of the high-level semantic operations are stored into the database as methods, instead of being dispersed into the application programs. By making formal verifications of some of these methods, it should be possible to rely on some *trusted methods*, that is, on methods that actually perform the advertised services. Note that the verification of some of the methods represents a compromise between the verification of the entire collection of application programs as well as of all the methods and no verification at all. While verifying all application programs and all methods is not feasible because the code to verify would be too large, the option of no verification at all may lead to severe restrictions in the possible computations that can occur in the system. The possibility of relying on trusted methods will allow us to relax some of the restrictions that must be placed in a system in which all methods are assumed to potentially contain Trojan Horses.

2.7 Conclusion

Access control is a crucial functionality for any data management system. It is not sufficient that a system makes information available to users. The system has also to ensure that the information is protected from unauthorized accesses and modifications. Many security models have been proposed for conventional data models. By contrast, security in object-oriented database systems still presents several open issues. Features of object-oriented data models and languages, such as inheritance, subtyping, and encapsulation, bring new protection requirements which make the existing security models not adequate. Protection of information must be provided without compromising the advantages provided by the object-oriented approach over conventional systems. A model is yet to emerge that satisfactorily covers all protection aspects related to object-oriented database systems, and many problems still remain to be solved. However some interesting, but not complete, approaches have been investigated addressing the development of security models specifically targeted to object-oriented database systems.

In this chapter we have illustrated the protection problems in object-oriented database systems and reviewed some security models recently proposed. We have discussed both mandatory as well as discretionary security issues. In the presentation of the models, we have stressed how some features of the object-oriented paradigm, which automatically provide some form of protection, can be exploited for security purposes. Finally, we have outlined some open problems in the field and illustrated current researches aimed at their resolution.

Chapter 3

The Decomposition Property of Non-Deterministic Databases

Kumar Vadaparty*,Shamim Naqvi†

3.1 Introduction

Motivated by the need for increased modeling power for advanced applications involving design, scheduling, planning, etc., a number of attempts have been made to extend database technology [188, 339, 351, 33, 7, 3, 600, 264]. Indeed a major selling point for the newly emerging area of object-oriented databases is the increased modeling power provided by such systems. Although well motivated, such increased modeling power comes with a price: query evaluation is more expensive in the database programming languages (DBPLs) associated with these extended models. Traditionally, database technology has striven to develop declarative query languages in which the user specifies the query and the system evaluates the query in the best possible way. However, implicit in this evaluation is a guaranteed upper bound on time (performance guarantees). Obviously, such a goal can not be reached if the DBPLs are extended from restricted languages (relational algebra, Datalog, etc.) to full-fledged programming languages (such as C++). Thus, we are faced with two competing goals: increase the modeling power of declarative query languages, and performance guarantees for DBPLs.

One approach to deal with these competing goals is to extend the modeling power incrementally (accounting for the most important needs first), and see if query languages maintaining performance guarantees can be built for these extensions. One such extension to modeling power was described in [274, 597, 408] wherein it is shown that dealing with choices enables us to model more expressive domains. For example, in design situations we can state that the implementation medium of a certain part can be chosen from the set $[Cobalt, Nickel]$. Such a set of choices is called an OR-object. A database with OR-objects corresponds to several possible worlds; for instance, the previous database fact corresponds to two possible worlds: "Part#1 can be implemented using Cobalt" and "Part#1 can be implemented using Nickel". In [274] OR-objects model data involving choices, e.g., the above choices can be stored as $Implement$(Part#1,o) where $Dom(o) = \{nickel, cobalt\}$ gives the domain of the OR-object o. The two possible worlds corresponding to the choices can be obtained by replacing the OR-object with a member of its domain: $Implement(Part\#1, Cobalt)$

*Department of Computer Engineering and Science, Case Western Reserve University,OH 44106

†MRE-2E-342, 445, South Street, Bellcore, Morristown, NJ07960

and $Implement(Part\#1, Nickel)$. Such a refinement is the database analog of making a choice. Note that OR-objects encapsulate a specific kind of disjunction, namely, disjunction in one or more attributes of the predicate. Thus, a disjunction such as $P(a) \vee Q(b)$ can not be captured by OR-objects. However, it was observed in [274, 275, 597] that even this restricted disjunction can improve modeling power considerably. However, utility aside, such an extension of the relational model with OR-objects, increases the data complexity of conjunctive queries to coNP-complete[276, 277]. This exacerbates the second of our two competing goals, namely the problem of performance guarantees.

Combinatorial optimization applications (such as scheduling) also consider choices in data, and consequently address the issue of attacking intractable queries. One useful method of attack is to use domain specific heuristics to prune the search space. The question at hand, thus, is how can the idea of heuristics be exploited in the context of databases with OR-objects? In other words, if we assume that our users are domain specialists and have considerable knowledge about the domain of application, can we provide formal tools to operationalize their domain heuristics? *An important property of any such tool is that the tool itself should be domain independent.* This chapter addresses precisely this issue and provides some solutions.

A notable step in this direction is the work of [273] who consider the following question. Given a query Φ, can we determine values for certain design parameters such that, as long as databases conform to these values of the parameters, Φ can be evaluated in PTIME (i.e., what values of the design parameters force Φ to be tractable)? The queries are limited to positive existential conjunctive queries with no predicate occurring more than once in the query. Two design parameters were identified for this purpose:

1. the typing function, and
2. the degree of co-referencing.

For the class of databases that satisfy certain conditions imposed on these two design parameters, Φ is guaranteed to have polynomial data complexity. However, if these conditions are violated, then Φ is guaranteed to become intractable (coNP-complete). In other words, the conditions on the design parameters are maximal (assuming $P \neq NP$) for maintaining tractability of Φ. Informally, the typing function specifies which attributes are allowed to have OR-objects. The degree of co-referencing is a measure of the inter-relationships of the elements of the OR-object. The values of both design parameters can be set by a user who is familiar with the underlying semantics of the application domain. For example consider the query $\Phi \equiv \exists x[Inexpensive(x) \wedge Researchy(x)]$ where the two predicates specify properties of universities. Note that one can have choices of universities in either of the predicates. In [273] it is shown that Φ has PTIME complexity if at most one of the two predicates contains choices, and has coNP-complete complexity otherwise. Thus, database designers may disallow choices in one of the two predicates by an appropriate change in the typing function. In this way, heuristics of the application domain can be operationalized in the database system. The particular predicate in which OR-objects are disallowed is determined by the designer, using the domain knowledge (thus operatlionalizing the domain knowledge).

However, the approach of [273] has some limitations: it applies only to proper queries, i.e., queries in which a predicate does not appear more than once, and, more importantly, it does not address the issue of what to do if we are confronted by a coNP-complete query and a fixed typing function. For instance, suppose both the predicates in the above example are required to have choices in the data because of

lack of complete information. Then, for this class $\mathcal{D}$ of databases that allow choices in both predicates, [273] simply declares that the query is coNP-complete. Can we do anything more?

Note that even though the data complexity of Φ for $\mathcal{D}$ is coNP-complete, there may exist subclasses of $\mathcal{D}$ for which Φ can be evaluated in PTIME. The questions we address in this chapter are: can we guarantee PTIME behavior for the coNP-complete query Φ for certain subclasses of $\mathcal{D}$? What kind of approaches can we expect to use in order to obtain such subclasses? Are those approaches domain-independent? In other words, can the same approaches be used on any other query Φ' and class of databases $\mathcal{D}$ (where Φ' is coNP-complete for $\mathcal{D}$)?

It is important to understand the domain independent nature of the heuristic support that we desire. A central assumption in databases is that the data in a database is uninterpreted. It has no inherent semantics. It is only through interactions with a query that data acquires a meaning. Thus, when we seek tractable subclasses of $\mathcal{D}$ we would like to respect this central assumption. Our techniques that identify such tractable subclasses are independent of the query so that they will work for any query and class of databases.

We believe that this chapter makes some initial progress towards obtaining answers to the above questions. Our methodology works as follows. We identify a property called "decomposition property" and show that the classes of databases satisfying this property enable polynomial time query evaluation. This property can be defined as follows. A class of databases $\mathcal{D}$ has decomposition property for a query Φ, if for any $D \in \mathcal{D}$, it is the case that $D \models \Phi$ can be determined by first determining if $D_i \models \Phi$ for every $1 \leq i \leq m$, where $D \equiv D_1 \cup \ldots \cup D_m$. We refer to D_i's as modules or "clumps". These modules depend on the query Φ and also on the database D. Thus, the problem of evaluating the query in the entire database can then be reduced to evaluating it for each of the modules. Interestingly, it seems that there is a strong relation between the decomposition property and *locality* of choices. Because choices occur naturally in many situations, our approach seems to have practical impact as well.

Example 3.1.1 (An Example)
Suppose that a database has facts such as $f_1 = Inexpensive(u1) \vee Inexpensive(u2)$, $f_2 = Researchy(u3) \vee Researchy(u2) \vee Researchy(u4)$, *i.e., there is choice data in both the predicates. Suppose that a domain specialist knows that the choices are* localized: *that is we may have facts such as* $f_3 = Inexpensive(rutgers) \vee Inexpensive(Princeton)$ *where both rutgers and Princeton are in NJ, but we do not have facts such as* $Inexpensive(rutgers) \vee Inexpensive(Standford)$ *where the universities involved are far apart (across several state boundaries). Thus, the domain expert predicts that facts over choices are localized at the state level, with a few facts containing choices stradling state boundaries. Such localized information gives rise to "clumps" or "modules".*

We can depict the clumpiness of data by constructing a database-graph, as follows: facts such as f_1, f_2 *can be represented by nodes of a graph with an edge between two nodes if the corresponding facts have a possible world that entails the query. In our example above, the nodes corresponding to* f_1 *and* f_2 *will have an edge because the set* $\{Inexpensive(u2),\ Researchy(u2)\}$ *is a possible world of the facts, and entails the query* $\exists x[Inexpensive(x) \wedge Researchy(u2)]$. *Figure 3.1 shows such a graph for an example database. There are seven facts* $f_1, \ldots, f_7$ *that constitute "clump1", and nine facts* $f_8, f_8, \ldots, f_{15}$ *constituting "clump2".*

In this chapter we formally define the notion of a clump, show the allowed interactions between clumps, and show how to determine $D \models \Phi$ by composing the results

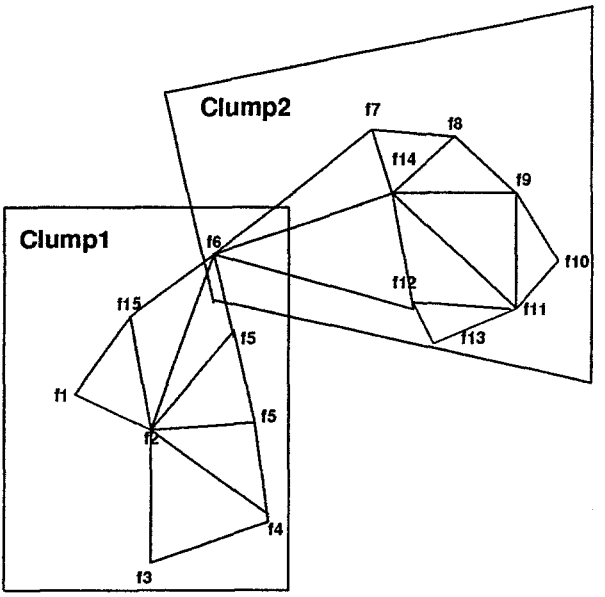


Figure 3.1: An example of "clumpiness".

of entailments of the query on each of the clumps of D. Clearly, in such an analysis the time spent is bound by the number of clumps and the processing time per clump. Assuming that a domain specialist can make a prediction *a priori* about the size k of the largest clump (size is the number of disjunctive facts within a clump), our analysis enables one to identify a class $\mathcal{D}_k$ of databases for which Φ is tractable ($\mathcal{O}(n^k)$). Note that a domain specialist often can determine such k's. In the above example, a specialist who knows about universities can predict that no more than 10 such facts (nodes) may lie in a given "clump". Thus, our analysis enables one to evaluate the query $\exists x[Inexpensive(x) \wedge Researchy(x)]$ in polynomial time using this heuristic (even though it is coNP-complete in general).

Note that such clumps occur very commonly in many domains where locality is a common factor: in testing of circuits, one can localize the mistake to a particular zone. Thus a disjunctive fact such as $Wrongoutput(node1) \vee Wrongoutput(node2)$ will have both of the nodes $node1$ and $node2$ from the same zone or PCB. In scheduling problems, one can localize the machines to be assigned to a task t by their capabilities: thus, $Assignedto(t, cpu1) \vee Assignedto(t, cpu2)$ would involve $cpu1$ and $cpu2$ that belong to the same class of machines (comparable FLOPs, frequency, etc.)

We would like to emphasize that our approach depends crucially on the presence of locality in choice data. Thus, it can be used in those places where this locality plays an important role. We would like to caution the reader that there exist queries in which locality does not work. For example, consider the database to represent cities in India, a country with dense cities; suppose that the query involved population: $\exists xy[City(x) \wedge City(y) \wedge Population(x, y, 10)]$ asking if there are two cities with total population more than 10 million. In this case, almost any two (distant) cities could participate in the graph and clearly, we can not hope to obtain clumps (or, in other words, the clumps will be very large).

An interesting approach is that this locality principle can be used retroactively in designing the circuit or plan so that it may be possible to diagnose or understand the design in an incremental fashion. Indeed it may be argued that in order to facilitate understanding of the application, we must design the database in such a way that most of the complexity is localized rather than spread throughout the database. Consider a database representing a complex piece of software or circuit design and that we are interested in understanding or diagnosing the design, i.e., we know the sorts of queries that users will be asking. If the complex behaviors of the design are a function of the entire design then a query may not be evaluable in any semantic subset of the database. However, if complex behaviors were localized, we may be able to isolate semantic modules of the database that could be responsible for that complex behavior. In such a case, the modules can be evaluated incrementally, yielding a PTIME evaluation of an otherwise intractable query. One factor that causes complexity in design and planning situations is the set of available choices for a task at hand. The locality principle then suggests that facts about choices be localized.

This chapter is organized as follows. Section 3.2 motivates the need for data complexity, and provides the required background results. Section 3.3 defines the notions of size, density and witness of a query, and shows the use of these notions. Section 3.4 develops the modulewise evaluation strategy by proving the decomposability results for a number of classes of databases. Section 3.5 gives the possible future extensions.

3.2 Basic Notions

In this section we motivate the need for considering *data complexity*, *OR-databases*, and show how the data complexity jumps from PTIME to coNP-complete in the context of OR-databases. This discussion is intended to provide the desired background for the discussions in succeeding sections.

3.2.1 Entailment, and Data Complexity

By a query we mean a closed conjunctive formula in First Order Logic, with existential quantifiers only. Such formulae capture simple boolean queries. For example, consider the boolean query " Q = Is John the manager of some department manufacturing Toys?". Let

$$D \equiv \{Manager(John, dep1), Manufacturer(dep1, 'toys'), Manager(Jacob, dep2)\}$$

Then clearly, the answer to the query is "yes." Consider the formula

$$\Phi \equiv \exists d\ [Manager('John', d) \wedge Manufacturer(d, 'toys')]$$

We say that the formula Φ is true in D (D entails Φ) iff there is a mapping μ for the variables in Φ such that when μ is applied to Φ, we get a subset of the database. This notion of entailment is denoted by $D \models \Phi$ (standard notion of entailment). The above database D entails the formula Φ through the mapping $\langle d/'dep1' \rangle$. It is easy to see that $D \models \Phi$ iff the query Q is true in D. We say that Φ captures Q and refer to Φ as a query. Thus, in the rest of the discussion, "formulae" and "queries" are used interchangeably.

We will be interested in *Data Complexity* [598, 95] of queries. Given a query Φ and a denumerable set $\mathcal{D}$ of databases, the data complexity of Φ with respect to $\mathcal{D}$ is the

complexity of

$$\{ D \mid D \in \mathcal{D} \text{ and } D \models \Phi \}$$

The motivation for data complexity is as follows: suppose we are given a database D and a query evaluation algorithm for evaluating a query in D. It is natural to expect that the query evaluation algorithm works correctly even if the database is updated. In fact, we expect that the algorithm is invariant under all or some updates. This notion can be made precise as follows: suppose $[D]^+$ denotes the closure of D under a set of pre-specified update operations. Then, we expect the algorithm evaluating Φ to evaluate it correctly for any database in $[D]^+$. The complexity of any such algorithm is bounded from below by the complexity of the set

$$\{ D \mid D \in [D]^+ \text{ and } D \models \Phi \}$$

which is the data complexity of Φ with respect to $[D]^+$. Thus, data complexity provides a lower bound on any algorithm that we can hope to design for evaluating queries in an update transparent manner. The data complexity of positive conjunctive existential queries for relational databases (closed under the update operations of *DELETE* a tuple and *INSERT* a tuple) is in PTIME (actually in LOGSPACE) [598, 95]. The following observation reconstructs this result.

Observation 3.2.1 ([598, 95])
Let Φ be a positive conjunctive existential query. Then the complexity of

$$\{ D \mid D \text{ conforms to the arities of relations in } \Phi \text{ and } D \models \Phi \}$$

is in PTIME.

Proof: The idea is as follows: let k be the number of variables in Φ and let M be the set of all domain constants in any database D. Then, $D \models \Phi$ iff there exists a mapping μ from the variables in Φ to the constants M such that $\mu(\Phi)$ is a subset of D. Clearly, the number of possible mappings is a fixed polynomial in the size of M and hence even the straightforward approach of testing these mappings one by one gives polynomial time complexity. ■

However, unfortunately, when we allow disjunctive information in databases, this nice tractable property no longer holds. In fact, it was shown in [276, 272] that the data complexity of simple conjunctive queries is in coNP-complete even if we allow restricted disjunctive information denoted by "OR-objects".

3.2.2 Choices and Data Complexity

It was observed in [274] that choices in the form of "OR-objects" plays an important role in scheduling, planning, and design applications. We illustrate this here and show how the issues of entailment and data complexity can be discussed in the context of databases with OR-objects (called OR-databases). In the following the table TRAVEL shows a travel schedule for various employees of a company. The objects o_1 and o_2 denote OR-objects, and their domains are shown next to the table. The first entry, specifies that "*John has a choice of going to CA or NJ.*" The second entry has a similar meaning. The third entry involves a complete specification: "*James goes to CA.*"

Travel

Employee	*Place*
John	o_1
Jack	o_2
James	*CA*

Domains

$Dom(o_1) = \{NJ, CA\}$
$Dom(o_2) = \{MA, CA\}$

A possible World of **Travel**

Employee	*Place*
John	*NJ*
Jack	*MA*
James	*CA*

The table TRAVEL has several possible worlds, each obtained by substituting the OR-objects by a value from their respective domains, i.e., by making a choice from an OR-object. One such possible world is shown in the adjoining figure. In general, then, a database containing choice data represents a set of possible final instances or worlds. These *possible worlds* are also referred to as *models.*

We say that a query is true in an OR-database if it is true in every possible world of that database. To illustrate the notion of entailment in disjunctive tuples, consider the database $D \equiv \{P(o), Q(a), Q(b)\}$ with $domain(o) = \{a, b\}$ and the query $\exists x\ [P(x) \wedge Q(x)]$. The database corresponds to two possible worlds, each corresponding to a particular choice of the OR-object, o. It can be observed that in each of the possible worlds, the query is true; hence we say that the query is entailed by the database.

The following observation proves that indeed there exist queries whose data complexity is coNP-complete in the context of disjunctive information.

Observation 3.2.2 ([276, 272])
There exist queries that have coNP-complete data complexity in the context of databases with disjunctive information.

Proof:
Consider any graph $G(V, E)$ with V as vertices and E as edges. Let $\{R, G, B\}$ indicate three colors. We say that the graph G is colorable if there is a mapping from the vertices to the colors such that no two endpoints of an edge are colored by the same color.

The following set is known to be coNP-Complete [212]:

$$\{\ G \mid G \text{ is not colorable }\}$$

In other words, the problem of determining if for any coloring scheme, it is the case that at least two vertices connected by an edge are colored by the same color is coNP-complete. We use this problem to show that the data complexity of a particular query is coNP-complete. Suppose that from the graph G we construct a database $D(G)$ consisting of two relations $Vertex$ and $Edge$ such that $Vertex \equiv \{\ v \mid v \text{ is a vertex of } G\}$ and $Edge \equiv \{\ \langle i, j\rangle \mid \langle i, j\rangle \text{ is an edge of } G\}$. Also we construct a relation with OR-objects as follows:

Color

vertex	*possible colors*
v_1	o_1
.	.
.	.
.	.
v_n	o_n

The domains

$Dom(o_1) = \{R, G, B\}$
.
.
.
$Dom(o_n) = \{R, G, B\}$

Clearly, the query

$$\Phi \equiv \exists xyz\ [Vertex(x) \wedge Vertex(y) \wedge Edge(x,y) \wedge Color(x,z) \wedge Color(y,z)]$$

is true in every possible world of the database $D(G)$ iff G is not colorable. This shows that the data complexity of Φ is coNP-complete. ■

Thus, in general, the complexity of evaluating queries in OR-databases is coNP-complete. In [273] data complexity of a particular class of positive existential conjunctive queries was analyzed: this class is called *proper* queries. In a proper query, no two literals have the same predicate. The results of [273] can be summarized as follows: first identify two design parameters called (i) typing function and (ii) degree of coreference; typing function specifies whether a particular column can take OR-objects, while the degree of coreferencing restricts the extent to which OR-objects can appear repeatedly. Now, given any proper query Φ, the analysis of [273] enables us to determine the values of the design parameters such that for the class of databases conforming to these parameters, the data complexity of the given proper query is in PTIME. Thus, for example, consider the query $\Phi \equiv \exists x P(x) \wedge Q(x)$. Then the analysis of[273] states that the data complexity of Φ is in PTIME if either P or Q is restricted not to take any OR-objects, and OR-objects can repeat in the column that they are allowed to occur. *The interesting aspect of [273] is that if the restrictions stipulated are not conformed to, there is a guarantee that the data complexity of that query will be coNP-complete.* Thus, we obtain what is called *complete syntactic characterization.* This approach was called *complexity tailored design* in [273].

However, the above approach has some limitations: if a query Φ is found to have coNP-complete data complexity for a class $\mathcal{D}$ of databases, then, [273] does not provide a way to deal with it any further. This chapter addresses the issue of identifying subsets of $\mathcal{D}$ such that for those the data complexity of Φ is in PTIME.

3.3 Size and Density of a Witness

As stated earlier, our aim is to identify tractable subclasses of $\mathcal{D}$ for the query Φ, where it is known that Φ has coNP-complete data complexity for $\mathcal{D}$. We identify such classes for any conjunctive query Φ and any class of databases $\mathcal{D}$ that has distinct OR-objects. Thus, the queries we consider need not be *proper*, unlike [273]. However, we assume that the OR-objects are distinct[‡], and each predicate has at most one OR-argument. The latter is not a serious restriction because, a large class of queries with more than one OR-argument in their predicates can be split (using a join attribute) into predicates with at most one OR-argument[273].

Our methodology works as follows. We identify classes of databases that satisfy a property called "decomposition property", defined as follows. A class of databases $\mathcal{D}$ has the decomposition property for a query Φ, if every $D \in \mathcal{D}$ can be written as a union of D_i for $1 \leq i \leq m$ such that the following holds: whether or not $D \models \Phi$ can be determined by composing the results of $D_i \models \Phi$ for every $1 \leq i \leq m$, where $D \equiv D_1 \cup \ldots \cup D_m$. We refer to D_i's as modules. These modules depend on the query

[‡]Two OR-objects are distinct if there is no constraint that relates the choice over their elements. For example, let $o1$ and $o2$ be two OR-objects that contain choices of courses for John and Mary. Consider the constraint that John and Mary choose the same course. Such a constraint relates the choices over $o1$ and $o2$ and are natural in many applications. We disallow such constraints in this chapter.

Φ and also on the database D. The decomposition property immediately guarantees that the data complexity of Φ for $\mathcal{D}$ is determined by the size of the modules of D. If k is the largest cycle (as we define later) in a module, then the data complexity of Φ is given by $\mathcal{O}(2^k \times D^{querysize} \times modulesize^{(querysize+k)} \times$ number of modules) time to determine if the module entails the query (here k is a fixed-constant, specifying the size of maximal cycle in the module, as described later). Thus, the data complexity of Φ with respect to the class of databases $\mathcal{D}$ is a polynomial in the size of the database (D). We identify different such classes $\mathcal{D}_0, \ldots, \mathcal{D}_k, \ldots$ for different parametric values of k. Note that this situation of assigning a maximal size of the module arises very often in practice as described in Section 3.1.

Thus, there are two issues: modules and their inter-connections. Having agreed to pay a fixed time (parametric in k) $\mathcal{O}(2^k \times modulesize^{querysize})$, to determine if a module entails the query, we expect the modules to be as "big" and as "dense" as possible (trying to get the maximum out of what we are willing to pay).

In this section, we first define a notion of "size of a witness of a query" and show that in the case of OR-databases, this size is not bounded, whereas in the case of relational databases, this is bounded. We show that bounded size of the witness yields (a straight forward) proof of the polynomial time evaluation of queries in relational databases, whereas the unboundedness of the witness makes that straightforward algorithm exponential. We show that although size is an important aspect, it alone is not sufficient for our purposes. We next introduce a notion of density of a witness. It is this notion that is used in the next section to develop the module-wise evaluation.

3.3.1 Size of a Witness

We first define the notion of a witness which is often used in the logic programming and database terminology, although not always explicitly.

Definition 3.3.1 (Witness)
Let Φ be a closed conjunctive query, D a database, and μ a mapping from the variables of the query to the constants of the databaseD. We call a set $S \subseteq D$ to be a witness of Φ in D if every ground atom corresponding to applying μ to the literals of Φ is a member of S. Clearly, $S \models \Phi$. S is minimal if for no subset S' of S is a witness of Φ in D.

Suppose that the size of the minimal witness is bounded by the size of the query (denoted by $|\Phi|$). Then, clearly, evaluating if the query Φ is true in D is in polynomial time in the size of the database: basically, consider all subsets of D of size $|\Phi|$ and see if any subset of atoms entails the query Φ. Fortunately, the following ensures that if D is indeed a set of atoms, then it is the case that any minimal witness is bounded by the size of the query.

Observation 3.3.1 *If D is a set of positive atoms and Φ a positive conjunctive closed query, then the size of the minimal witness of Φ is bounded by the number of literals in Φ.*

The proof of the above observation follows easily from the definition of a witness. Note that the largest size of any mapping is bound by the number of literals in the query.

However, the above observation does not hold if we allow OR-objects in the database. First we would like to make the notion of minimal witness precise in the context of

OR-objects. We assume throughout the chapter that OR-objects are distinct. Thus, given a database D with OR-objects, it can be re-written as a First Order Theory by expanding the disjunctions implied by the OR-objects. From our definition of OR-objects, it is clear that this theory is in a special conjunctive normal form: each of the disjuncts are formed from the same predicate symbol. Clearly, D can be viewed as a set of clauses.

Now, a witness S to a query Φ in a database D with OR-objects can be defined as follows: $S \subseteq D$ is a witness of Φ in D iff every possible world (see Section 3.2.2) of S entails Φ. We call S minimal, if for no subset S' of S the above holds. The following lemma shows that the size of a minimal witness is not bounded by the number of literals in a query in the context of OR-databases. In fact, it shows a stronger result.

Lemma 3.3.2 *There exists a query Φ such that for any natural number $i \in \mathcal{N}$ there exists a database D_i such that the minimal witness of Φ in D_i is larger than i.*

Sketch of Proof:
Consider the query $\Phi \equiv \exists x\ [P(x) \wedge Q(x)]$.

For the case of 1, the desired database is $D_1 \equiv \{P(a), Q(a)\}$

For the case of i, consider the database $D_2 \equiv \{P(a_1) \vee \ldots \vee P(a_i), Q(a_1), \ldots, Q(a_i)\}$. Note that D_i constitutes minimal witnesses. ■

Thus, a straightforward approach of testing a fixed number of tuples of the database to see if they entail the query does not work in the case of databases with OR-objects. Since the size of the minimum witness can be as large as the database itself, it follows that a straightforward approach leads to an exponential algorithm. The following lemma, however, shows that an unbounded witness does not necessarily mean coNP-completeness.

Lemma 3.3.3 (Unbounded witness does not imply coNP-Completeness) *There exists a query Φ and a class $\mathcal{D}$ of databases such that the size of witnesses for Φ in $\mathcal{D}$ is unbounded but still the data complexity of Φ for $\mathcal{D}$ is in PTIME*

Sketch of Proof:
Consider the query $\Phi \equiv \exists x[P(x) \wedge Q(x)]$ as before, and $\mathcal{D}$ the desired class of databases to be the class of databases in which OR-objects do not occur in the relation Q. From the previous lemma it follows that Φ does not have a bounded witness for this class.

We show that the data complexity of Φ for $\mathcal{D}$ is still in PTIME. Consider a clause c of P. Clearly, c is of the form $P(a_1) \vee \ldots P(a_k)$. We see if the relation Q has tuples of the form $Q(a_1), \ldots, Q(a_k)$. Note that Q has no disjunctions. If there are no such tuples in Q, then we conclude that the clause c does not contribute to the minimal witness, and take the next clause of P. If there is at least one clause of P that satisfies this requirement, then we conclude that $D \models \Phi$; else we conclude that $D \not\models \Phi$. Clearly, the above algorithm is in PTIME ($\mathcal{O}(n^2)$). ■

Thus, the query $\Phi \equiv \exists x[P(x) \wedge Q(x)]$ does not have bounded witness even for the class of databases in which only P is allowed to have OR-objects. But still, Φ could be evaluated in PTIME for that class. Thus, bounded witness is only a sufficient condition for tractability.

We now develop a notion of density of a witness. In the previous example, the following holds:

$$\{P(a_1) \vee \ldots \vee P(a_k), Q(a_k)\} \models P(a_1) \vee \ldots \vee P(a_{k-1}) \vee \Phi$$

Thus, in order to construct a countermodel (i.e., a possible world that does not entail the query), we need to consider only the first $k-1$ disjunctions of $P(a_1) \vee \ldots \vee P(a_k)$. In other words, for the purpose of constructing a counter-model, we might as well remove the disjunct $P(a_k)$ from $P(a_1) \vee \ldots \vee P(a_k)$ to form $P(a_1) \vee \ldots \vee P(a_{k-1})$.

Thus, using the above stepwise refinement, the new clause has only $k-1$ disjuncts. Arguing similarly with the other definite tuples of Q, one at a time, we conclude that there is no counter-model for Φ (we eventually conclude that any such counter-model will have zero disjuncts of P – a contradiction).

We call the above database a "low density" database because one can use a "stepwise" or *incremental* reasoning to determine if the query is entailed by the database. The following example shows that this limited or incremental reasoning can be applied to databases which are not necessarily as restricted as above.

Example 3.3.1 (stepwise procedure applies to more complex databases) *Consider the same query $\Phi \equiv \exists x[P(x) \wedge Q(x)]$ and the database shown in Figure 3.2. The database $D \equiv \{Q(a), Q(e), P([a,b]), P([d,e]), Q([b,d])\}$. (The square brackets denote choice data, e.g., $P([a,b])$ denotes $P(o)$ where o is an OR-object whose domain is $\{a,b\}$. This notation is used only as a shorthand.) This database is constructed in the form of a tree. Note that the vertices are essentially the clauses of the database, and two vertices are connected by an edge if they together have a possible world in which the query is entailed.*

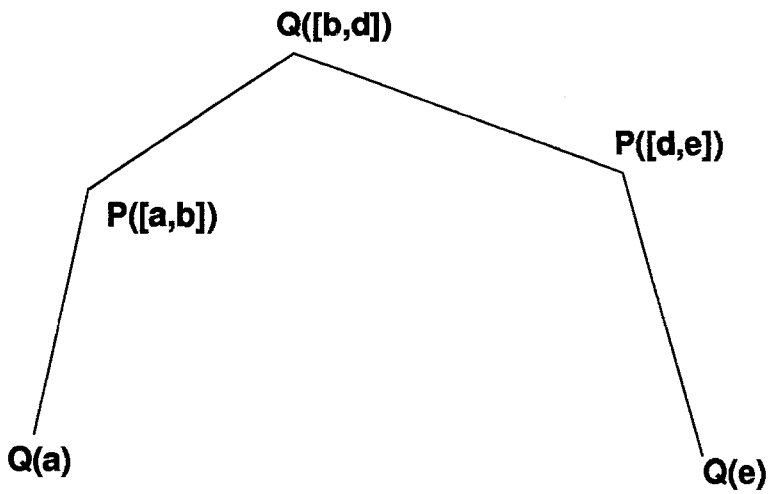


Figure 3.2: A "tree like" database enabling stepwise reasoning.

Note that one can perform a bottom-up stepwise refinement in the above database instance as follows: starting with the left most child, one can conclude from $Q(a)$ and $P([a,b])$ that the only counter-model, if there is any, for the query should be the one in which a is removed from $P([a,b])$. Similar stepwise reasoning yields that e can be removed from $P([d,e])$. Continuing this further, we conclude that at the root we do not have any choices left for $Q([b,d])$ from whence it can be concluded that there is no counter-model for the above database.

Note that the above database D constitutes a minimal witness of Φ. Thus, the above tree-structure and the associated stepwise refinement indicate that even though the size of a witness is unbounded, what is more important is the ability to apply the stepwise refinement. *We call a database dense if stepwise refinement can not be performed directly on it to determine if it entails a query.* By stepwise refinement we mean, a procedure that uses only bounded set of tuples to conclude if a particular choice can be removed, for the purpose of computing a counter-model (the bound is the number of literals in the query). This stepwise refinement is analogous to, but different from the extended Chase-mechanism discussed in [271].

The next subsection identifies "dense" instances which can not be evaluated in a stepwise manner.

3.3.2 Density of Witnesses

We say that a database D has density property for a query Φ, if stepwise reasoning is not applicable to determine if $D \models \Phi$. We illustrated in the previous subsection that "tree like" databases do not have the density property (see Figure 3.2). In the following we show that database with "cycles" satisfy the property of being "dense". To understand the notion of cycle, consider the following example:

Example 3.3.2 (Why cycles?)
Consider the query $\Phi \equiv \exists xy[H(x,y) \wedge H(y,x)]$ *where the second argument of* H *is an OR-object. Consider the database*

$$D = \{H(a,[c,e]), H(e,[a,f]), H(c,[a,f]), H(f,[e,a]), H(f,[c,a]), H(a,[b,f])\}$$

See Figure 3.3 in which this database is represented as a graph using the same approach as in the previous example: if two nodes have a possible world that entails the query, then put an edge between them.

It is easy to see that the above database does not support stepwise refinement. In fact, we need to look at all possible worlds of the entire database before we make any decision. More precisely, $D \models H(a,b) \vee \Phi$, *and for the purpose of constructing counter-models, we can delete the choice* f *from the choices of the tuple* $H(a,[b,f])$. *Thus, in order to perform the refinement (i.e., reducing the size of an OR-object), we need to consider the entire database, not a bounded set of tuples.*

Thus, modules with cycles seem to capture the interesting property that the module as a whole needs to be considered for refinement. In other words, a cycle can not be further broken into smaller components for the purposes of refinement. Consequently, any straightforward algorithm needs to reason with the entire module (of size k) requiring $\mathcal{O}(2^k)$ time. Thus, we choose components with maximal cycle size as k as the desired "clumps" or "modules" for constructing $\mathcal{D}_k$. We would like to remark that we do not claim that cycles imply an exponential lower bound. We use modules consisting of cycles as units because they are amenable to stepwise evaluation.

3.4 Modulewise Evaluation

Recall that our methodology is to define "decomposition property" and use it to identify subclasses $\mathcal{D}_1^\Phi, \ldots, \mathcal{D}_1^\Phi, \ldots$ that enable polynomial time evaluation for a given query Φ;

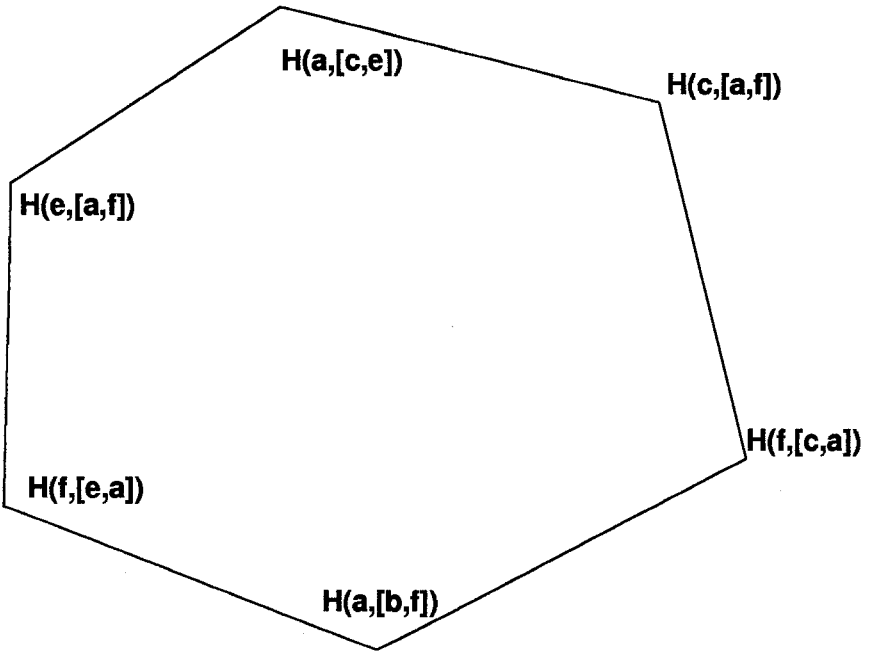


Figure 3.3: A "cycle" like database that does not enable stepwise reasoning.

here the subclasses are subsets of $\mathcal{D}^{\Phi}$ for which Φ is known to have coNP-complete data complexity.

A class of databases $\mathcal{D}$ has decomposition property for a query Φ, if for any $D \in \mathcal{D}$, it is the case that $D \models \Phi$ can be determined by determining $D_i \models \Phi$ for every $1 \leq i \leq m$, where $D \equiv D_1 \cup \ldots \cup D_m$. We refer to D_i's as modules. There are two issues: the kind of modules, and the kind of "connection" or interface between modules in a given database D of a given class $\mathcal{D}_k$.

In the previous section we showed that the modules are collections of nodes that have a maximal cycle of size k where k is a pre-fixed parameter as discussed in Section 3.1. The next question is how do we expect to break the database into components. In other words, how do we decompose a database into modules? We define what is called "acyclic" collection of components. In other words, $D \in \mathcal{D}_k$ iff D can be broken into an "acyclic collection" of components or modules of size k. Then we prove that indeed $D \models \Phi$ can be determined by reasoning with only the components.

This section formalizes the notions of database graphs, nodes, edges, cycles, modules, and finally, establishes the desired decomposition property.

Definition 3.4.1 *Given a database D and a query Φ, we construct a hyper-graph $G(D, \Phi)$, called instance-graph as follows. The nodes of G are the atomic formulae of D. $\{f_1, \ldots, f_k\}$ is an arc of G iff it has* **a possible world** *that entails the query Φ.*

If the graph happens to be a multi-graph, we expect the nodes to be numbered (in some lexicographic order) so that distinct edges are maintained.

Definition 3.4.2 *Degree of a node in a hyper-graph is the number of arcs incident at that node. A path (simple) is a sequence $\langle v_1, A_1, \ldots, v_p, A_p \rangle$ such that all v_i are*

the vertices of the hyper-graph, and all A_i are the arcs of the hypergraph; furthermore, $2 \leq j \leq p$, $v_j \in A_{j-1}$.

Henceforth, we assume that the instance graph is connected. Otherwise, we can apply our analysis to each unconnected component of the graph.

Definition 3.4.3 *A path is a cycle if $v_1 = v_p$. A hyper-graph is acyclic if it has no cycles. Every acyclic hyper-graph corresponds to a "tree" in a natural sense. The degree of an arc is the number of other arcs that share its nodes. A leaf arc is one whose nodes are shared by exactly one other arc. The arc that is connected to a leaf arc is called its parent arc. The node(s) shared by a leaf arc and its parent are called "bridge" nodes.*

See Figure 3.5 for an illustration of these notions.

Observation 3.4.1 *No two arcs in an acyclic hyper-graph share more than one node. Otherwise it becomes a cycle. See Figure 3.4.*

Proof:

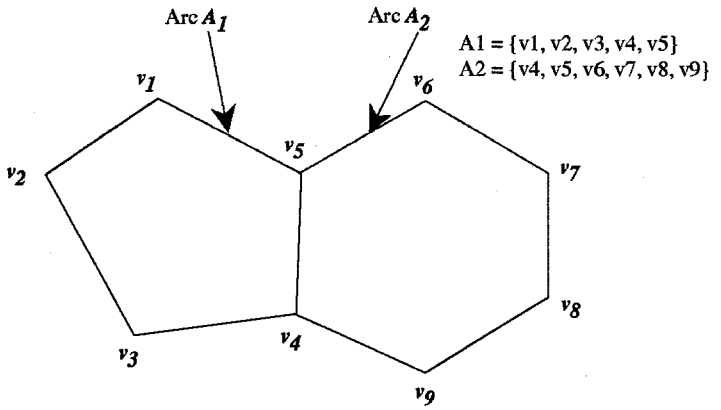


Figure 3.4: If two or more nodes are shared by two arcs, then there is a cycle.

Note that the desired cycle is $\langle v_5, A_1, v_4, A_2, v_5 \rangle$. ■

Definition 3.4.4 *A query is singly-matching if for any set of tuples, there is at most one possible world that entails the query.*

For example, the query $\exists x[P(x) \wedge Q(x)]$ is not a singly matching query because $P([a,b]), Q([a,b])$ entails the query in two possible worlds. However, it can be proved that the query $\exists xy[P(x,y) \wedge Q(y,x)]$ restricted so that the second argument of P and the first argument of Q are allowed to take OR-objects, is a singly matching query, and yet has coNP-complete data complexity.

Now, we prove that for any singly-matching query Φ and the class of acyclic databases $\mathcal{D}_0^\Phi = \{ D \mid G(D,\Phi)$ acyclic$\}$, it is the case that the data complexity of Φ is in PTIME. In order to prove this fact we first show that this class of databases satisfies the decomposition property on its modules.

Definition 3.4.5 (decomposition property)
For a given query Φ, a class of databases $\mathcal{D}^{\Phi}$ has decomposition property, if there is a k such that the following holds for every $D \in \mathcal{D}^{\Phi}$: D can be decomposed into databases $D = \bigcup_{i=1}^{m} D_i$ and $|D_i| \leq k$ and the problem of $D \models \Phi$ can be determined by composing the results of $D_i \models \Phi$.

Note that the decomposition property immediately establishes the PTIME property of Φ for $\mathcal{D}^{\Phi}$.

Theorem 3.4.2 *For any given query Φ, the class $\mathcal{D}_0^{\Phi}$ has decomposition property.*

Proof:
Let $G(D, \Phi)$ be the instance graph. We show that D has the decomposition property where the desired modules of D are the arcs of the instance graph. The desired constant of the decomposition property is the size of any arc of $G(D, \Phi)$, which in turn is bound by the size of Φ.

We use induction on the number of arcs in G to show this result. Base case: G has at most one arc. Trivial.

Assume that the induction hypothesis holds for every database with the number of arcs of at most m. Consider a database with $m+1$ arcs. We show how to decompose this database. Let l be a leaf arc of D. Let p be the bridge of l. Let $l = \{v_1, \ldots, v_k\}$. There are the following cases:

- all nodes of l_1 are definite. Then D trivially entails Φ.
- At least one vertex of l (other than the bridge) is an OR-formula. Call it v.
- All but the bridge of l is an OR-formula.

See Figure 3.5.

Case: At least one vertex other than the bridge of l is an OR-formula:

Let $v \in l$ be an OR-formula $H([a_1, \ldots, a_j], \vec{b})$. Let m_1 be the model of l such that $m_1 \models \Phi$. Let a_i of H correspond to the model m_1. Choose another $a_i' \neq a_i$ and $a_i' \in \{a_1, \ldots, a_j\}$ and construct $m_1' = m_1 - \{H(a_i, \vec{b})\} \cup \{H(a_i', \vec{b})\}$. Note that $m_1 \not\models \Phi$. This is because Φ is singly matching.

Consider the graph G' obtained from G by deleting all the vertices but the bridge of l and deleting the leaf-arc. Let D' be the database corresponding to G'. We show below that $D \models \Phi$ iff $D' \models \Phi$. Thus, whether $D \models \Phi$ can be determined by decomposing D into D' and the arc l. Since $G(D', \Phi)$ has fewer arcs than $G(D, \Phi)$, the induction hypothesis guarantees that it can be decomposed into modules satisfying the decomposition property. Hence the theorem will follow if we prove that $D \models \Phi$ iff $D' \models \Phi$.

Suppose $D' \models \Phi$ for every possible world. Since $D' \subset D$, the desired result follows. Suppose that $D' \not\models \Phi$. Then, let m' be the counter model of D' i.e., $m' \not\models \Phi$. Now consider the database $m = m' \cup m_1'$. Clearly, m' is a model of D. Furthermore, $m \not\models \Phi$. For, suppose m models Φ. Then, since neither m' nor m_1' model Φ, it must be that some tuples of m_1' and some other tuples of m_1' together model Φ. In other words, the arc l is not connected to just one arc, but two. Thus, there is a cycle in G, a contradiction.

Case 2: The bridge p is an OR-formula, and every other node of l is definite:

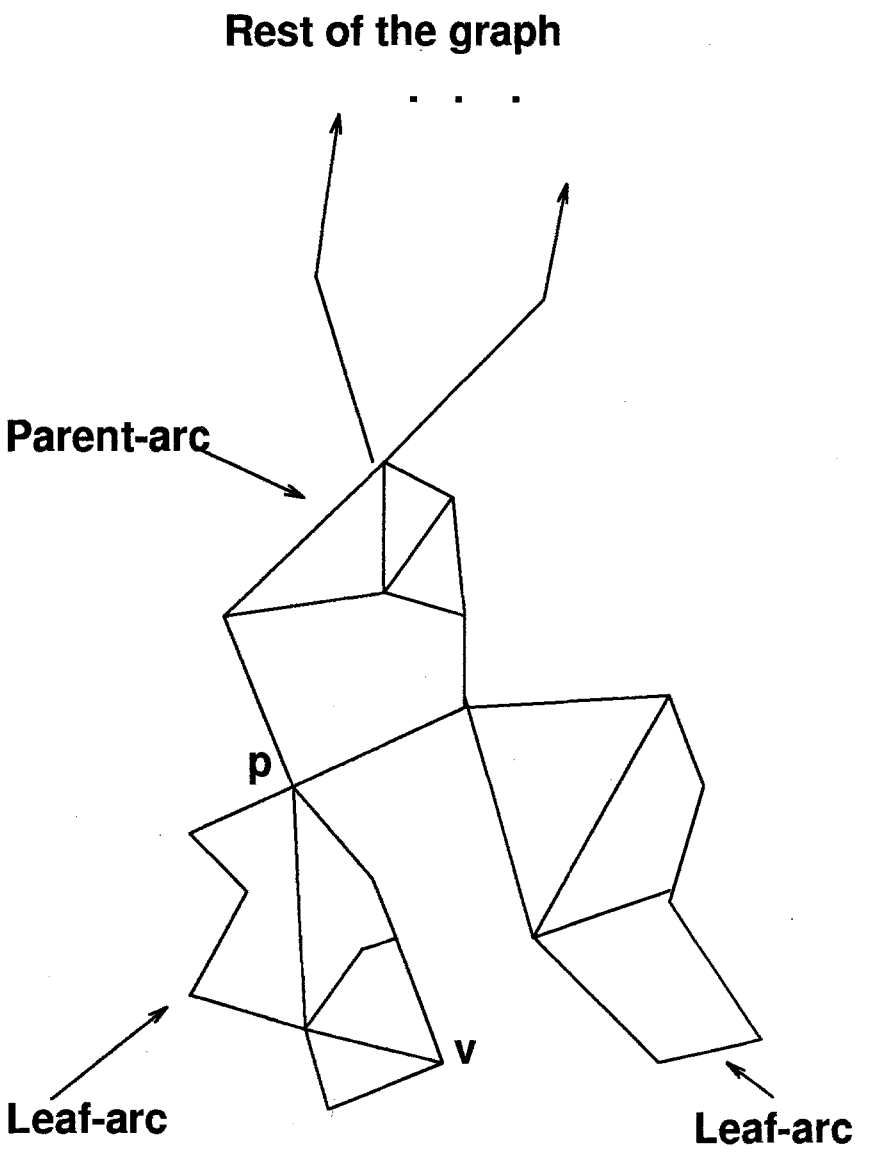


Figure 3.5: A graph corresponding to a database.

Let p be denoted by $H([a_1, \ldots, a_p], \vec{b})$. Let m_1 be the model of l containing $H(a_i, \vec{b})$. Then $m_1' = m_1 - \{H(a_i, \vec{b})\}$. Clearly, m_1' is a set of all definite formulae that exist in every model of D. This is because all the formulae of l except p are definite. Let $p' = H([a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_p], \vec{b})$. In other words, p' is obtained from p by deleting a_i from the OR-object of p. Let G' be the graph obtained by deleting the leaf arc l from G and deleting all the nodes associated with l from G, and adding the new node p'. Let D' be the database corresponding to G'. We show that $D \models \Phi$ iff $D' \models \Phi$. This will in turn prove, as in the previous case, the theorem.

Suppose that $D' \models \Phi$. Then we need to show that $D \models \Phi$. Note that $D' \not\subseteq D$. In fact, all formulae of D' are in D except for p'. Instead of p', D has the formula p. Thus, to show that $D \models \Phi$, we should consider also those models of D in which p is entailed by $H_1(a_i, \vec{b})$. Since p' does not have a_i in its OR-object, $H_1(a_i, \vec{b})$ is not in any model of D'. However, in any model of D that contains $H_1(a_i, \vec{b})$, $m_1' \cup \{H_1(a_i, \vec{b})\}$ entails the query Φ, because it constitutes the model of l. Thus every model of D entails Φ.

Suppose $D' \not\models \Phi$. Then, there is a model m' of D' such that $m' \not\models \Phi$. Then, the database $m_2 = m' \cup m_1'$ is such that m_2 is a model of D and $m_2 \not\models \Phi$. Thus, $D \not\models \Phi$, as desired. ■

In other words, the above theorem proves that if D is acyclic, and Φ is singly matching, we can determine if $D \models \Phi$ by considering the arcs of $G(D, \Phi)$ one at a time, in a modular manner. The PTIME characterization of $\mathcal{D}_0^\Phi$ follows immediately, as the theorem below records the result.

Theorem 3.4.3 *The data complexity of any query Φ for $\mathcal{D}_0^\Phi$ is in PTIME.*

We extend the above result in two ways, (i) the case when there are cycles in $G(D, \Phi)$, and (ii) when the query is not singly matching.

In the case of multiply-matching queries, each arc can have more than one model that entails the query. Hence, we can not extend the above proof directly to multiply-matching queries.

If the graph has cycles, we can divide it into components such that each component has at most a k-size cycle. Thus when we collapse each component containing a maximal cycle of size k into a single arc (called "super arc"), the original graph becomes acyclic in these super-arcs. If a database is such that its instance graph is acyclic in k-sized cycles, then we say that D is in $\mathcal{D}_k$. Note that since each super-arc can have multiple models that entail the query, in this case also, we can not extend the above proof directly.

Thus in both the cases we need to account for multiple models of a (super) arc entailing the query. The same approach can be used in both the cases, and the following theorem establishes the first result.

Theorem 3.4.4 *Let Φ be a multiply-matching query, and*

$$\mathcal{D}_0^\Phi = \{\ D \mid G(D, \Phi) \text{ is acyclic}\}$$

Then, $\mathcal{D}_0^\Phi$ has the decomposition property for Φ.

Proof: As in the previous theorem, there are essentially two cases: (i) the parent of a leaf arc is an OR-formula, (ii) the parent of a leaf arc is a definite formula but there is some other node in the leaf-arc that is an OR-formula. Let l be a leaf-arc, and p be its parent.

Case (i):
Let $l = \{A_1, \ldots, A_k\}$ where A_i's are the arcs of the leaf-arc l, and $V = \cup_{A_i \in l} A_i$ denote the set of all nodes in the leaf-arc l. Recall that each arc is a set of nodes. Let $p = H_1(o_1, \vec{b})$ be the parent of the leaf-arc l where the domain of o_1 is given by $Dom(o_1 = \{a_1, \ldots, a_m\}$. Let $l' = l - \{p\}$; thus, l' is the set of all nodes of the leaf-arc except the parent. Let

$$S = \{\ a \mid\ a \in Dom(o) \text{ and } (\{H(a, \vec{b})\} \cup m) \models \Phi \text{ for every model } m \text{ of } l'\}$$

Thus, $S \subset Dom(o)$ is such that for any model of l involving $a \in S$ substituting o in l entails the query Φ. Let p' be the new parent node defined as follows: $H(o', \vec{b})$ where $Dom(o') = Dom(o) - S$.

Consider the new graph $G'(V', A')$ obtained from $G(v, A)$ as follows: to obtain V' from V, delete all the nodes of l and add p'; to get A' from A, delete the arc l from A. Thus, $A' = A - l$, and $V' = V - l \cup \{p'\}$. Let D' be the database corresponding to $G(V', A')$. We show that $D \models \Phi$ iff $D' \models \Phi$. As in the previous theorem, this would prove that D satisfies the decomposition property.

Suppose that $D' \models \Phi$. Every model m' of D' entails Φ. We need to show that every model of D also entails Φ. Consider any model m of D. It either contains a model of D' in which case it entails Φ, or it does not contain a model of D'. Therefore, consider the latter case: from construction of D' it follows that m is of the form $m' \cup \{H(a, \vec{b})\} \cup l'_1$ where m' is a model of D', $H(a, \vec{b})$ is a model of p the parent of the leaf-arc, and finally, l'_1 is a model of l', the leaf-arc except the parent p. Since m is not a superset of m', it follows that $H(a, \vec{b})$ is such that $a \notin o'$. Therefore, from construction of o' from o, it follows that $a \in S$ where S is as defined before. From the definition of S it follows that $\{H(a, \vec{b})\} \cup l'_1 \models \Phi$. Hence, clearly, m models Φ because $\{H(a, \vec{b})\} \cup l'_1 \subseteq m$.

Now suppose that $D' \not\models \Phi$. We show that there is a model of D such that it does not entail Φ. Since $D' \not\models \Phi$, there exists a model m' of D' such that $m' \not\models \Phi$. ¿From the construction of D', it follows that the model m' can be written as $m' = m'_1 \cup \{H(a, \vec{b})\}$ for some $a \in Dom(o')$. This is because, D' is obtained from D by removing all the nodes of l and adding $p' = H(o', \vec{b})$. Thus $a \in Dom(o') = Dom(o) - S$. Now consider a model l'_1 of l' such that $l'_1 \cup \{H(a, \vec{b})\}$ does not entail Φ. From the construction of o' it follows that such a model of l' exists. Now, we claim that $m'' = m'_1 \cup l'_1 \cup \{H(a, \vec{b})\}$ is such that m'' is a model of D and $m'' \not\models \Phi$. Since $m'_1 \not\models \Phi$ and $(l'_1 \cup \{H(a, \vec{b})\}) \not\models \Phi$, it follows that for m'' to entail Φ, it must be the case that some nodes of l'_1 together with some nodes of m'_1 entail Φ. Then, this contradicts the assumption that l is a leaf-arc (and in turn contradicting the assumption that G is acyclic). Hence, it must be the case that $m'' \not\models \Phi$.

Case (ii):
In this case p is a definite formula. The proof of this case is analogous to the corresponding case of the previous theorem. ■

Next, we extend the above two theorems for graphs involving cycles. As stated earlier, graphs with cycles are viewed as acyclic graphs with "super-arcs". A super-arc is, essentially, a maximal component of the graph with a cycle of size k. The set $\mathcal{D}_k^\Phi$ is defined as the set of all databases whose instance graphs (with respect to Φ) can be viewed as acyclic when each maximal component containing k-sized cycle is collapsed into a single "super arc". The proof of the following theorem is analogous to that of the previous theorem (with the obvious and minor modifications).

Theorem 3.4.5 *For a query Φ (singly or multiply matching), the class $\mathcal{D}_k^\Phi$ has decomposition property.*

3.5 Future Extensions

We are currently working in a number of directions.

- First note that our characterization works only when the OR-objects are distinct. An immediate extension is to see how to extend this for multiply occurring OR-objects. This will be useful because we would like to capture natural constraints over choices such as "John and Mary would like to choose the same course."
- Our characterization of acyclic graphs prohibits more than one node of sharing between one leaf arc and a parent arc. However, in the context of hyper-graphs, more than one node sharing should be allowed, at the same time disallowing arbitrary sharing. We have some ideas on how to restrict this sharing. We would like to work in this direction so that we can capture bigger classes of tractable databases.
- We would like to see how to perform this characterization in the context of queries with choices.

Chapter 4

The Architecture of an Object Base Environment for Simulation

Phillip C-Y. Sheu*,Larry J. Peterson†

4.1 Introduction

Object-oriented computation in the broad sense is computation described as a sequence of requests to objects through a single access method such as message passing. It is generally accepted that object-based systems such as Smalltalk have provided a simple and elegant paradigm for general-purpose programming which can be meshed well with data models. In brief, the class/method mechanism handles well the requirements for type definition and information hiding. The message protocol provides a useful way of controlling the updates that can be performed on a data object. In addition, the inheritance mechanism makes database schemas easy to modify, and new variants can be constructed easily as subclasses.

The above features naturally lead to the use of the object-oriented paradigm in instrumenting and supporting simulation activities. On the other hand, the nature of object-oriented representations suggests that an object-oriented simulation program can be executed in parallel. To our knowledge, more than a dozen object-oriented simulation languages/systems have been developed; some have considered parallel processing. A survey of such systems/languages can be found in [572] [200]. Most of such systems/languages extend an object-oriented programming language with the necessary constructs for simulation, in particular "...the notion of simulation time and mechanisms for entities in the language to manipulate simulation time." [326]. However, the issues of object management and object retrieval have not been fully addressed. As for parallel processing, focus has been placed on developing efficient algorithms which allow events be processed parallelly to the maximal extent.

This chapter describes the design of a parallel object-oriented simulation environment. An object base is defined to be a system which contains a large set of active as well as passive objects. For active objects, not only the data portion, but also

*Department of Electrical & Computer Engineering, Rutgers University, Piscataway, NJ 08855

†Naval Command, Control and Ocean Surveillance Center RDT&E Division, Code 421, San Diego, CA 92152-5000.

This work is supported in part by the Office of Naval Technology, Code 227, Computer Technology Block Program through the Office of Naval Research ASEE Summer Faculty Research Program.

the control and knowledge portions of an object are stored and managed. Parallel evaluation of simulation programs is accomplished by compiling objects into sets and production rules so that they can be evaluated with parallel, set-oriented operations which effectively utilize the capacity of parallel processors with minimal communication overhead. It is organized into the following sections: Survey of Related Work, Object Representation, Management of Active Objects, Simulation, and Conclusion.

4.2 Related Work

Work related to the simulation environment described in this chapter can be classified into three categories: parallel simulation systems, object-oriented databases, and active databases.

Parallel Simulation Systems The problem of parallel processing of simulation systems has attracted much attention recently. A number of parallel computation models and their associated problems have been investigated [200] [326]. The models can be classified into two categories: synchronous and asynchronous. In a synchronous, parallel simulation system, processes and events are scheduled and executed by the simulator with a global clock. On the contrary, each process in an asynchronous, parallel simulation system maintains its own clock; in addition, processes and events are scheduled and executed in a fully distributed fashion. This means there exists no scheduler to synchronize the events globally. According to [200], "...few simulator events occur at any single point in simulated time; therefore parallelization techniques based on lock-step execution using a global simulation clock perform poorly or require assumptions in the timing model that may compromise the fidelity of the simulation". Accordingly, "Concurrent execution of events at different points in simulated time is required, but ..., this introduces interesting synchronization problems...". Most of such synchronization problems are resulted from data dependencies among different processes which run under different speeds. Approaches to the synchronization problems can be in turn classified into two categories: conservative and optimistic. A conservation approach prevents any synchronization problem from happening, but it degrades performance. An optimistic approach allows synchronization problems to occur, and rollbacks are often necessary once these problems are detected.

Object-Oriented Databases In the past, several object-oriented databases have been proposed. In brief, researchers and developers have approached object-oriented database implementation along two directions: extending the relational model (e.g., POSTGRES [384] [342], GENESIS [164], Iris [134], and PROBE [592]) or applying the ideas of object-oriented programming to permanent storage (e.g., GemStone [135]). Most of the systems in the first category have been designed to simulate semantic data models by including mechanisms such as abstract data types, procedural attributes, inheritance, union type attributes, and shared subobjects. Most of the systems in the second category extend an object-oriented programming language with persistent objects and some degree of declarative object retrieval.

Both approaches have drawbacks in processing a large number of active objects. The first approach suffers from the unstability problem resulting from the separation of control and data. The second approach, on the other hand, loses the advantages provided by fact-oriented database operations.

Active Database Systems The idea of incorporating rules into a database system has exist as integrity constraints and triggers as early as in CODASYL, in the form of ON conditions. More recently, the idea of combining rules and data has received much serious consideration. The term "active database" has been used frequently in referencing such database. For example, rules has been built into POSTGRES [342]: there is no difference between constraints and triggers; all are implemented as a single rules mechanism. In addition, POSTGRES allows queries be stored as a data field so that it is evaluated whenever the field is retrieved. In HiPAC [138], the concept of Event-Condition_Action (ECA) rules was proposed. When an event occurs, the condition is evaluated; if the condition is satisfied, the action is executed. It can be shown that ECA rules can be used to realize integrity constraints, alters, and other facilities. Rules have also been included in the context of object-oriented databases. In Starburst [285], for example, rules can be used to enforce integrity constraints and to trigger consequent actions. In Iris [134], a query can be monitored by first defining it as a function and later creating a monitor for the function.

On the other hand, there has been a growing interest in building large production systems that run in a database environment. The motivations are derived from two areas. First, expert systems have made an entry into the commercial world. This has brought forth the need for knowledge sharing and knowledge persistence. These are features found in current databases. Secondly, many emerging database applications have shown the need for some kind of rule-based reasoning. This is one of the principal features of expert systems. Production systems is a commonly used paradigm for the implementation of expert systems. The confluence of needs from the areas of AI and database has made the study of database productions very important.

Traditionally production systems have been used in AI, where data are stored in main memory. Various needs, as mentioned above, have lead production systems designers to use databases for data storage. We refer to these as *database production systems (DPS)*. Commercial DBMS's do not have the necessary mechanisms to provide full support for such systems. Views can be used in lieu of rules, but only in a limited way. Recent work has produced more powerful mechanisms to handle a large class of rules [31] [93]. However, the focus has been on retrieval, especially evaluating recursive predicates, and proposed approaches do not handle updates as in systems like OPS5 and HEARSAY-II. Recent efforts by [571] [341] [375] [84] have addressed this issue, and much attention has been placed on parallelizing the evaluation of production systems (see, e.g., [234] [283]). To our knowledge, little effort has been made for production systems that work on objects or distributed evaluation of production systems.

4.3 Object Representation

To illustrate the concept of object base for simulation, we have chosen to extend C++ as the object representation language. It is chosen based on the observation that C++ has acquired enough attention and acceptance as the object-oriented language in the computer community. These extensions are described in the following subsections.

4.3.1 Complex Objects

A complex object is an object consisting of a set of (possibly complex) objects in the sense that (1) The domain of an attribute can be any class; and (2) The value of an

attribute can be a set of objects. A complex object is an abstraction of its component objects. Consequently, a method associated with a complex object implements a function of its component objects as a whole; so are the attributes of the complex object. Non-complex objects are called *simple* objects. To realize the concept of complex object, it is necessary to explicitly incorporate the notion of "set" in the object language:

Set Classes Given a class α, the class of all possible ordered sets which can be derived from instances of α is declared as:

class set_of_α {
...
methods
...
}

The following declaration defines a set a of class α:

set_of_α a;

Set Projection Given a set or an object a of class α, the following notation designates the projection of a on attributes $A1$, ..., An:

$a|A_1, ..., A_n$

4.3.2 Active Objects and Models

The constructs provided in C++ are sufficient to describe *passive* objects, i.e., objects whose activities are triggered when a method associated with the object is called. In real applications, a special class of objects need to be defined in order to describe objects which are continuously active according to some control mechanism. Such objects are called *active* objects [85].

Control An active object can be characterized by a set of states and a set of state transition rules. In the extended language, a class of active objects is declared as a subclass of the class *active*, for which any attribute, once defined, can be declared to be a state as follows:

state attribute,...,attribute;

The control portion of an active object is expressed as a set of production rules, which is designated as the attribute *control* (whose domain is *set-of-production*) of the object. A production is asserted in the following form:

condition $\Rightarrow$ *statement;*

where *condition* is any logical expression over states and inputs, and can include any quantifier over sets:
Universal Quantifier
A variable in a logical expression can be universally quantified by the quantifier:

(forall <variable_id> in <set_id>)

Existential Quantifier
A variable in a logical expression can be existentially quantified by the quantifier:

(exist <variable_id> in <set_id>)

Membership
The following function returns 1 if *<variable_id>* is an element of *<set_id>*:

<set_id>:member(<variable_id>);

Similar to complex objects, we can define a *complex active object* to be a set of (possibly complex) active objects. With this definition, a complex active object can be regarded as a *concurrent production system*[‡]

Communication For a complex active object, we classify the styles of communications among its component objects into two categories: synchronous and asynchronous. Communication between two objects is defined to be synchronous if:

1. The calling object suspends its execution after a message is sent to the other object; and
2. The calling object resumes execution immediately after a reply is received from the called object.

Communication between two objects is said to be asynchronous if the calling object continues its execution after a call is made. Asynchronous communication is achieved in the extended language via messages. The class *message* is defined as follows:

```
class message {
public:

time time-stamp, reference;
object sender, recipient;
set-of-object arguments;
void send();
boolean receive();
};
```

An asynchronous call to method a associated with object c with arguments is made by first creating a message object, assigning appropriate values to its attributes, followed by sending the message with the *send* operation:

δ.*send*();

where δ is the message just created. On the other hand, any message sent to an object is picked up by the boolean function *receive*, which is called in the form of:

m.receive();

[‡]Briefly, a production system consists of a set of rules, or "productions", which is of the form (*condition*) $\rightarrow$ (*action*), a database or "context", which maintains the state/data of the system, and a rule interpreter. The condition portion of each rule (LHS) is composed of some logical combination of the results obtained from comparing some "state variable(s)" to a fixed value(s) or to some other state variable(s). They are tested continuously. If the condition is true, the consequent action (RHS) of the rule is executed.

The function returns true if a message has been received; in this case m is instantiated to the message received. It returns false otherwise. A message m is regarded as the reply to a previously sent message δ if:

$m.reference = \delta.time\text{-}stamp$
$m.sender = \delta.recipient$
$m.recipient = \delta.sender$

According to the above, an ordinary C++ function call $c.a(x_1, ..., x_n)$ implements a synchronous communication session; it is equivalent to a *send* operation followed immediately by a *receive* operation.

Inputs, Outputs, and Links In the extended language, some attributes of an active object can be chosen to be the inputs and outputs of the object as follows:

classifier attribute,...,attribute;

where *classifier* can either be the keyword *input* or the keyword *output*. The assignment of inputs and outputs allows different objects be connected directly in order to form an interconnected complex object. A linkage between two objects can be established by the operation *link*:

link(a.r,c.s)

This operation connects $a.r$, presumably to be an output of object a, to $c.s$, presumably to be an input of object b, so that any assignment to $a.r$ is made to $b.s$ as well instantaneously.

Clock and Model To support simulations, the attribute *clock* is associated with each active object, and a set of methods is available to manipulate the clock. Finally, any subclass of the class *active* is called a *model*.

Class Template In summary, the general form of the class declaration for an active object class is:

class <class_id> {
<class_id> <variable_id>,[...<variable_id>];
...
<class_id> <method_id> (parameter_1:domain_1,...,
parameter_n:domain_n);
...
classifier attribute, ..., attribute;
...
int clock;
set-of-production control = {
<logical-expression> ⇒ actions;
....
}
}

In the above, a classifier can be one of the following keywords: *state, input*, and *output*.

Associative Object Retrieval The availability of sets as described in the above also allows objects be retrieved in an associative fashion. It is assumed that the following functions/statements are used to access the elements in a set:

1. *<set_id>:insert(<variable_id>);*
2. *<set_id>:delete(<variable_id>);*
3. *(foreach <variable_id> in <set_id>) statement;*

4.4 Management of Active Objects

Given a set of active objects, the problem of object management is concerned with the impact created by any change made to the system. It is desirable that adjustments can be made automatically according to any change so that the system is always consistent.

4.4.1 State Space and Criteria

Given an active object P with n states s_1,...,s_n, we define the state space of the object to be $Domain(s_1) \times Domain(s_2) \times \cdots \times Domain(s_n)$. Among the states, we assume that one is chosen as the initial state and a number of them are chosen as the final states. We define the set of reachable states of P to be the set of all possible states which can be reached, either directly or indirectly, from the initial state. For the purpose of discussion, the following criteria are chosen as the constraints when an active object is updated:

Liveness An active object should have no state which is a dead-end state, where a dead-end state is a state from which no further state transition can occur and it is not a final state.

Consistence An active object should be consistent in the sense that, in any state, there exist no conflicting actions, where two actions conflict each other if their effects logically violate each other.

Other criteria, such as reachability and deadlock-freeness, can be considered in a similar way.

4.4.2 Adding and Removing A State

If an active object is live and consistent, these two operations are processed as follows:

Adding A State Assuming s_{n+1} is added to object P and the object becomes P', the state space of P is enlarged to $Domain(s_1) \times Domain(s_2) \times \cdots \times Domain(s_n) \times Domain(s_{n+1})$. Any state v in the original state space now corresponds to $Domain(s_{n+1})$ states (v,u_1).....(v,u_r), where $r = Domain(s_{n+1})$ and the set of u_i's spans all possible values of s_{n+1}. Let us assume that v is not a final state. Since v is not a dead-end state in P, there must exist a rule in P for which v satisfies its left hand side. Clearly, in P', each of (v,u_1).....(v,u_r) still satisfies the LHS of the same rule. Consequently, P' remains to be live. On the other hand, since no new rules (and actions) are added to P', no conflicting actions may be taken in each of the new states. In summary, adding a state variable to a live and consistent object does not damage such properties.

Deleting A State Deleting a state is more complicated than adding a state. Assuming s_i is removed from object P and the object becomes P', the state space of P is shrunk to $Domain(s_1) \times \cdots \times Domain(s_{i-1}) \times Domain(s_{i+1}) \times \cdots \times Domain(s_n)$. $Domain(s_i)$ states $(v,u_1).....(v,u_r)$, where $R = Domain(s_i)$ and the set of u_i's spans all possible values of s_i, now converge to a single state v. Since none of the original states $(v,u_1).....(v,u_r)$, where $R = Domain(s_i)$, is a dead-end state, it is clear that v is not a dead-end state. Consequently, P remains to be live after s_i is deleted.

Deleting a state is complicated due to the requirement of consistence. The complication arises from the fact that the left hand side of some rules may include conditions on s_i.Simply dropping such conditions from those rules can create the following problems: (a) An updated rule can violate the intention of the user; (b) Actions which used to be taken in states $(v,u_1).....(v,u_r)$ are now collectively taken in the state v; and some of them may be conflicting. Instead of inspecting every state for possibly conflicting actions, the following procedure can be taken: For each pair of conflicting actions a_i and a_j, identify $conditions(a_i)$ and $conditions(a_j)$. In the above, $conditions(a_i)$ designates the set of LHS's of those rules whose actions include a_i; $conditions(a_j)$ can be defined in a similar way. The intersection of $conditions(a_i)$ and $conditions(a_j)$ identifies the states in which conflicting actions a_i and a_j can be taken at the same time. Due to these factors, the user is consulted when a state is deleted and some rules are affected by this change. The conflicting actions are reported in the mean time, assuming all conditions including the deleted state are dropped. Subsequent actions from the user are handled according to the procedures for adding rules and deleting rules (see below).

4.4.3 Adding and Removing A Rule

If an active object is live and consistent, these two operations are processed as follows:

Adding A Rule Assume a rule R of the form $C_R \Rightarrow A_R$ is added to object P. The state space clearly remains to be the same. Let $states(R)$ be the set of states in which C_R can be satisfied. It is possible that executing R from a state in $states(R)$ can result in a state v from which no rule is applicable: a dead-end state. Such states can be detected by identifying all the states which may be directly reached from the states in $states(R)$ and followed by inspecting each of such states and looking for applicable rules. If a dead-end state can be discovered, the addition of R is not safe.

The addition of R may as well create conflicting actions, since the actions associated with R may be in conflict with actions of some rules which are applicable in a state of $states(R)$. The procedure described in the section "Deleting A State" can be applied to detect such states.

Deleting A Rule Assume a rule R of the form $C_R \Rightarrow A_R$ is removed from object P. The state space clearly remains to be the same. Let $states(R)$ be the set of states in which C_R can be satisfied. It is possible that removing R can result in a state which used to be directly reachable from a state in $states(R)$ no longer satisfies the LHS of any remaining rules: a dead-end state. Such states can be detected by identifying all the states which may be directly reached from the states in $states(R)$ and followed by inspecting each of such states and looking for applicable rules. If a dead-end state can be discovered, the removal of R is not safe.

On the other hand, the removal of R from P causes no problem as far as consistence is concerned. This is because each state v of the original state space is consistent, and

the removal of R does not create any new action in v.

4.4.4 Adding and Removing An Attribute

If an active object is live and consistent, these two operations are processed as follows:

Adding An Attribute Adding an attribute of a class in an object causes no problem since the states and the rules of the object remain intact.

Deleting An Attribute Deleting an attribute from a class of an object has no impact on the liveness of the object. Since some actions of the rules may include the attribute to be deleted, the removal of the attribute may make the action part of such rules incomplete. Simply dropping such updates from such rules may create problem as the updated rules may violate the intented semantics of the object, although the object remains consistent (as no new actions are taken). User involvement is required in this case.

4.4.5 Adding and Removing A Method or A Class

Adding a method is like adding an attribute; and deleting a method is like deleting an attribute. Adding a class is equivalent to adding a set of states, attributes, methods, and rules. Deleting a class is equivalent to deleting a set of states, attributes, methods, and rules.

4.5 Simulation

As discussed in Section 2, most of the existing object-oriented simulation systems provide an object-oriented user interface so that a simulation program can be described in an easy and friendly fashion. Execution of an object-oriented simulation program can be completely sequential or fully distributed as the program specifies. Although looks attractive, executing a simulation program as a fully distributed, object-oriented system could be inefficient due to the shortage of physical resources and the overhead associated with process management. Bearing this in mind, our approach compiles an object-oriented simulation program, in which each active object is represented as a production system, into a (production) rule network. Treating each active object as a passive object, each node of the network corresponds to a set-oriented operation. The compiled network (or the set of operations of the network) is evaluated in parallel. Changes to objects are generated at the terminals of the network. The overall architecture of the object-oriented simulator is shown in Figure 4.1.

4.5.1 Rule Processing

In general, processing of production rules or integrity constraints can become a serious performance bottleneck when a large number of objects and rules are integrated. Since multiple instances of the same class share the same copy of production rules, it is viable to compile a set of rules into one system in which some set-oriented operations can be employed to process the data (treated as sets) collectively. Furthermore, given a set of rules, it is viable to merge those expressions that are common to more than one rule so that duplicated effort can be avoided. We take a network approach for this

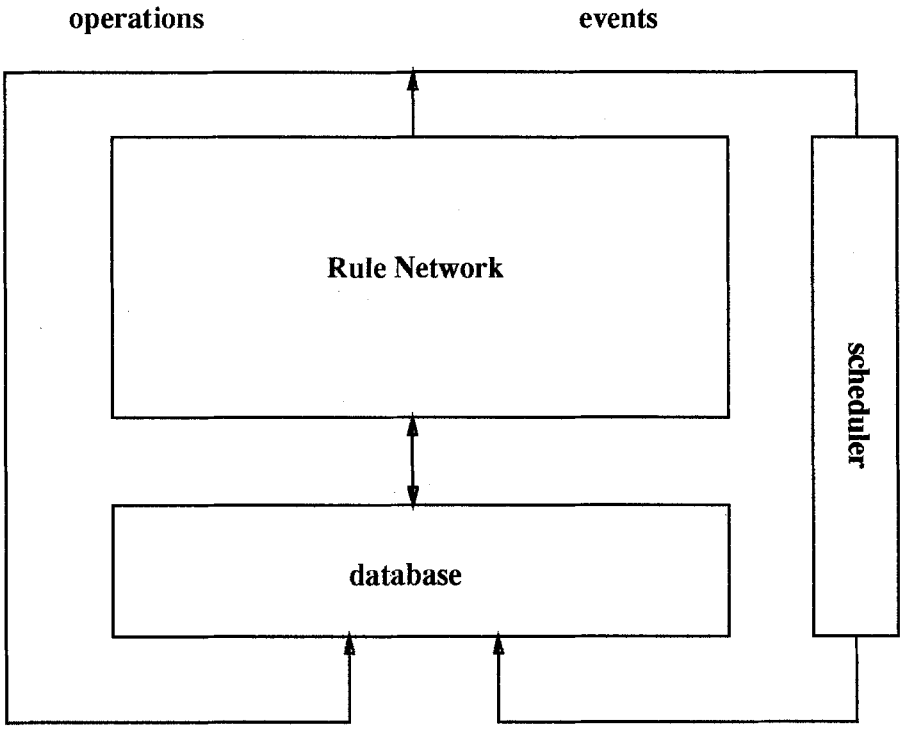


Figure 4.1: Simulator Architecture

purpose, which is similar to the RETE algorithm ([235] [340]) but is more general in treating logical formulas and structured objects. Although integrity constraints and production rules are treated slightly differently, both are processed based on a network that is compiled from a set of logical formulas.

Processing Integrity Constraints Given a set of constraints $\{f_1 \rightarrow r_1, \ldots, f_n \rightarrow r_n\}$, we can first convert each constraint $f_i \rightarrow r_i$ into the form $f_i \wedge \sim r_i$(i.e., the negation of the original rule). Subsequently, all the converted rules can be compiled into a network (see below), where each rule corresponds to a a terminal at the bottom of the network, and there is no violation of the rule if no results can "flow" out from that terminal.

Processing Production Rules Given a set of productions $\{f_1 \Rightarrow r_1, \ldots, f_n \Rightarrow r_n\}$, we can first compile all the left-hand-side formulas $\{f_1, \ldots, f_n\}$ into a network (see below), where each formula corresponds to a a terminal at the bottom of the network. Subsequently, each r_i (created as an action node) is connected to the corresponding terminal, and all the qualified objects for each production will "flow" into the action node.

Compiling Logical Formulas Given a set of logical formulas, a conjunct may be used in different formulas and those conjuncts having the same head and the same arguments can share the set of instantiations once they are computed; also, we can produce the result for a conjunct from the result of another conjunct if the first conjunct is more "general" than the second one. Here we define a conjunct P to be more general than another conjunct Q if

1. P and Q have the same head,
2. all the constants in P can be matched with those in Q, and
3. all the variables in P can be consistently unified by the variables or the constants in Q.

Now we define a rule network as a directed graph, where each node represents a conjunctive formula. A rule network consists of a set of nodes, each is labeled by a logical formula and corresponds to a set of objects. If a set of nodes n_1,...,n_k are connected to a node n, each with an out-going arc, it means the sets produced at the n_i's are input to n so that n can produce a set that corresponds to the logical formula it carries. A node without any input arc is called an input node; typically it is a class. A node which does not have any output arc is called a terminal node. A rule network can be constructed with the following procedure:

Constructing Rule Networks

Input: A set of conjunctive formulas Q_1, Q_2,...,Q_n for which each element corresponds to the LHS of a production rule.

Output: A rule network

Step 1

For each Q_i, identify the set of predicates which correspond to classes, i.e., those predicates which are class names. Create a node n, called a *join node*, for the predicate set if it has not been created; otherwise identify the node that has been created (by

some Q_j, $j < i$) and call the node n. Create another node and label the node with the other predicates, which is collectively called a *modifier*, and establish an arc from n to the newly created node. The arc goes from n to the modifier node since the outputs produced by n should be processed, and selected, by the modifier node; any output produced by the modifier node naturally corresponds to a tuple (of objects) that satisfies the LHS of the rule. This step basically eliminates any duplicated effort of creating a natural join of object classes among the rules.

Step 2

For any pair of join nodes n_1 and n_2, if the set of predicates carried by n_1 is a superset of that of n_2, establish an arc from n_2 to n_1. This step guarantees that the set produced by n_2 can be used by n_1. Since subsets may overlap, a join node can choose, among different combinations, the best input sets to combine.

Step 3

At the end of step 1, a number of nodes that perform natural joins should have been created. In the mean time, each rule corresponds to a modifier that selects from the corresponding natural join those qualified objects. Consequently each join node is connected to a number of modifiers. If there exists any predicate p which is common to more than one modifier, a new node, called a common factor node, is created so that it contains the MGP among such p's; an arc is created from n to the common factor node. Finally, the modifier node corresponding to each rule connects itself to those common factor nodes whose corresponding predicate subsumes one of its predicates. If after this a modifier m is connected to a set of common factor nodes c_1,...,c_k, the intersection of the outputs produced by c_i's is taken before the final selection is performed.

■

As an example, given the following formulas, the resulting rule network is shown in Figure 4.2.

class_1(X) && class_2(Y) && p1(X,Y) && p2(Y)
class_1(X) && class_2(Y) && p2(Y)
class_2(Y) && p3(Y)

The procedure described above cannot guarantee that the computational effort resulted from combining duplicated formulas be the minimum; even for the case of relational databases it has been proved to be difficult. It does, however, guarantee that no formula that is duplicated among several rules be evaluated more than once. Once a network is built, it is evaluated incrementally. Specifically, each operation is evaluated once based on the initial state of the system. In the mean time, for each operation, the results are stored. Subsequently, as the state of the system is changed, only those rules which are affected by a changed object need to be evaluated during each iteration. When an update of the database is made, operations are performed from the bottom of the network. Only those nodes containing the corresponding class predicate as the updated fact and whose arguments can be unified by the arguments of the updated fact are activated. After the common factors and modifiers are activated, the operations associated with the qualified rules are performed. For each join, common factor, or modifier node, the content of the stored result is changed according to the change(s) in its input set(s).

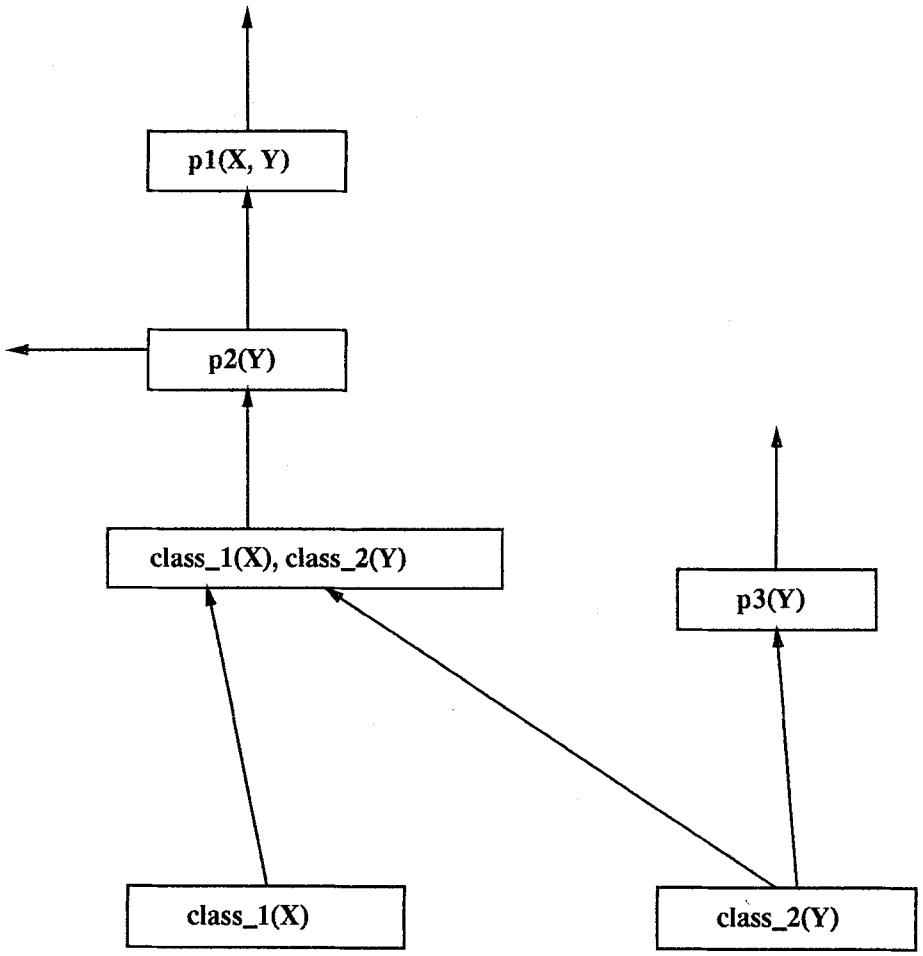


Figure 4.2: A Rule Network

4.5.2 Logic of The Simulator

As described, our approach merges a set of active objects into a (much) smaller set of operation processes. Simulation can be performed synchronously or asynchronously. In the synchronous mode, a global clock is employed so that each active object is synchronized with respect to the global clock; the system proceeds according to the discrete events produced. In the asynchronous mode, each active object proceeds at its own speed. In this case, joins in the rule network have to be performed on objects with different clock values. Consider a decision α which has to be made based on the states of two objects a and b, and a runs faster than b (i.e., $clock(a) > clock(b)$). If α is a decision for b (i.e., if the associated production is evaluated for b), clearly a previous state of a has to be used in making the decision. On the other hand, if α is a decision for a, and since in this case the "current state" for b is not available, a may assume that the state of b will not change during the period ($clock(a)$,$clock(b)$) and use the most recent state of b in making the decision so that in the worst case, it can roll back to this decision point if the relevant states of b are indeed changed once it catches up. The same reasoning can be extended and applied to decisions which have to be made based on more than two objects. Each node in the rule network has to use the correct versions of the objects in the computation. Even if the simulator runs in the synchronous mode, it is required that the states of each object be recorded and rollbacks be performed whenever causality errors are detected. It is necessary since messages may be mis-ordered. Based on the above, the simulator executes a loop with the following steps:

Event Execution In the case of synchronous simulation, the events with the smallest scheduled time with respect to the global clock are executed; in the mean time the global clock is adjusted to the events' scheduled time. In the case of asynchronous simulation, the events with the smallest scheduled time with respect to each object are executed; in the mean time the logical clock associated with the object is adjusted to the events' scheduled time.

Object Selection The rule network is evaluated. At the terminals of the network, actions are generated. This step basically selects those objects which have one or more productions eligible for firing based on their current states. As discussed earlier, in the case of asynchronous simulation, each join node of the network need to select an appropriate version of the object states for each object to join due to speed mismatches. To be specific, assume a join node α joins n classes of objects c_1, ...,c_n. Also assume at one instance of time an object r, which is an instance of class c_i, changes state (i.e., an updated r comes into α), with a new clock value T. Since the join should be incremental, r should be joined with $c_1(r)$,...,$c_{i-1}(r)$, $c_{i+1}(r)$,...,$c_n(r)$, where $c_j(r)$, $j \neq i$, consists of every instance of class c_j whose logical clock value is the maximal possible one which is smaller than T in its history. Each tuple (of objects) produced, say, (s_1,...,s_n), should be labeled by the augmented tuple ((s_1,T_1),..., (s_n,T_n)), where T_i is the clock value of object s_i chosen to be joined, so that it becomes the "cause" for the tuple produced.

Production Firing For each object selected, the actions associated with each firable production rule are taken. Such an action could be an operation which changes the value of an object, a communication operation (send and/or receive), or an event

(which will be executed in the future). If necessary, the value of the clock associated with the object is updated based on the operation(s).

Detection and Performing Rollbacks In a simulation program, rollbacks are required when some causality errors - usually due to speed mismatches in an asynchronous environment - are detected. The term "rollback" implies that each active object should maintain the history of itself so that it can go back from its current state to some previous state if necessary. Specifically, an active object need to record its state transitions. For any causality error detected, it need to locate the state in which a previously received message took effect, go back to that state, and undo any intermediate actions. More importantly, it should "unsend" any message that was sent in a state along the rollback state chain; this may cause cascading rollbacks take place on other objects.

Let us assume that each operation or event α is labeled by (P, T, ((s_1, T_1),..., (s_n, T_n)), s), where P is the production that triggered the action/event, T is the logical clock value of the target object (of the operation/event) at which it should take place, each (s_i,T_i) designates the state of object c_i and its time base on which the production was fired, and s is the state of the target object after the operation/event was taken. Clearly, α need to be undone if each of c_i, $1 \leq i \leq n$, for which $T_i < T$, is advanced to a logical clock value that is greater than T, and if S_i designates the state of c_i just right before T, all together S_1,...,S_n does not satisfy the LHS of P. Cascading rollbacks (backward in time) with respect to the target object are possible since all the relevant objects have caught up. This means a fast object may become a slower object after rollbacks. Since rollbacks for an object never occur until all the relevant "slower" objects have caught up, rolling back an object should not affect those objects which are faster than itself for which some operations/events were produced based on its current state; those objects may need to be rolled back once the currently-being-rolled-back object catches up later.

■

Since our object model allows objects be shared among different processes (although they are accessed through messages), it is important that *serializability* [442] be maintained all the time. This means the effects created by multiple processes which are executed concurrently should be the same as those created by a (any) serial schedule among the processes. To assure this, our design employs the two-phase locking protocol, which requires all objects accessed by a process be locked before accessed, all locks be acquired before any unlock, and all objects be unlocked before the process terminates. Clearly, two-phase locking cannot be implemented at the method level, as two consecutive method calls can violate the two phase requirement. Consequently, we require each method lock any object it may access but not unlock it. The list of locked objects should be returned to the calling process so that the process can unlock the locked objects before it terminates.

4.5.3 Parallel Processing

The following approaches can be taken in order to evaluate a rule network, depending on how logical objects are packed into physical objects:

Class-Level Parallelism In this approach, each node of the network is implemented as a physical object, where each input node is a class object and each internal node is an operation object. The network is evaluated as an active network, operating in a pipelined fashion. Specifically, each operation object retrieves inputs from its input object(s) and produces the outputs, which are available to the operation object(s) at the next higher level. Unlike operation nodes, each class object functions as a data store from which data can be retrieved by operation objects.

Set-Level Parallelism In this approach, each terminal node is implemented as a set of objects, where each of them corresponds to a subset of a class. The network is transformed into an equivalent one in which each terminal node corresponds to a subclass object. The transformation can be done in a straight-forward fashion based on the following principles:

1. $(R = R_1 \cup R_2) \wedge (S = S_1 \cup S_2) \Rightarrow R \bowtie S = (R_1 \bowtie S_1) \cup (R_1 \bowtie S_2) \cup (R_2 \bowtie S_1) \cup (R_2 \bowtie S_2)$
2. $(R = R_1 \cup R_2) \Rightarrow select_F(R) = select_F(R_1) \cup select_F(R_2)$

Clearly, this approach can achieve a higher degree of parallelism; however it is more complicated to implement. In addition, the number of operator objects can grow exponentially as each class is split into smaller and smaller subsets.

4.5.4 Example

This example consists of a number of divisions divided into two sides: blue and red. The divisions are initially located on the border of a battlefield which is modeled as a square of grid tiles. The scenario is set up so that all the red divisions are spread on the east border of the battlefield and the blue divisions are spread on the west side. Once initiated, the blue divisions march to the west and the red divisions march to the east, during which each division is characterized by its strength, speed, direction of movement, and its location. When two divisions of opposite sides encounter each other, the strength of the weaker is reduced to zero; in the mean time the strength of the stronger is reduced by the that of the weaker. Any division whose strength is 0 is removed from the system. At any instance of time, the number of divisions in each grid tile cannot exceed two.

The system as described can be expressed as a production system as follows. For simplicity, locking and unlocking operations for shared resources are not included in the rules.

division(d) && (d.color = red) && grid(g) && neighbor(d,g) && (g.capacity < 2) ⇒ march_left(d), (d.clock = clock + 2), (g.capacity = g.capacity + 1) (Rule 1)

division(d) && (d.color = blue) && grid(g) && neighbor(d,g) && (g.capacity < 2) ⇒ march_right(d), (d.clock = clock + 2), (g.capacity = g.capacity + 1) (Rule 2)

division(d) && (d.color = red) && division(e) && (e.color = blue) && same_grid(d,e) && (d.strength leq e.strength) ⇒ (d.strength = d.strength − e.strength), (e.strength = 0) (Rule 3)

division(d) && (d.color = red) && division(e) && (e.color = blue) && same_grid(d,e) && (d.strength < e.strength) ⇒ (e.strength = e.strength − d.strength), (d.strength = 0) (Rule 4)

division(d) && (d.strength = 0) && grid(g) && in_grid(d,g) ⇒ d.delete()
&& (g.capacity = g.capacity − 1) (Rule 5)

The rule network constructed for the above network is shown in Figure 4.3. As shown, after Rule 1 and Rule 2 are evaluated, all divisions which can move are scheduled to move into their new locations after two time units. After Rules 3 and 4 are evaluated, the strength of any moved division is adjusted; and those with 0 strength are removed after Rule 5 is evaluated. As can be observed, in the rule network, each rule is evaluated for those objects whose states are just changed by the other rules. In other words, the focus can always be placed on those objects whose states were just changed. This is different from the fully object-oriented approach. If the simulation system is fully distributed and object-oriented, each of the above rules need to be evaluated by each active object during each cycle, even if the object is not qualified to move.

4.6 Object-Oriented Evaluation of Rule Networks

This section presents an object-oriented approach to rule processing. It is "object-oriented" since each entity in the system is an active object, which acts with its own dedicated control sequences according to its functionality (e.g., constant nodes, predicate nodes, variable nodes). Upon receiving a message from another object, it executes its dedicated control sequence which may return a result or initiate an operation of other objects for further processing. All operations in objects are proceeded asynchronously. Since no object has global knowledge about the database, cooperation among objects is necessary in interpreting a production. Compared to the fully object-oriented approach, each object in this approach is much simpler in the sense that it does not need to evaluate a production system as required by the fully object-oriented approach. It does, however, have the minimal intelligence to participate and contribute to the rule evaluation process.

4.6.1 Structures of Extensional Databases and Query Networks

In order to store an object (which is represented as a predicate), three types of nodes are employed: object constant nodes, object predicate nodes, and object attribute position (OAP) nodes. Specifically, an object of class p with attributes $a_1,...,a_n$ is stored as an object predicate node p, a set of object constant nodes $a_i,...,a_n$, and a set of OAP nodes $p_1,...,p_n$. For each class p, the set of objects of p, designated as p_c, is collectively stored as an interconnected network, which consists of the following: an object constant node for each distinct constant which may appear as an attribute of an instance of p, n OAP nodes p_1 ,...,p_n, and $\mid p_c \mid$ object predicate nodes so that for each object of p_c, assuming its attributes are $a_1,...,a_n$, the object constant node a_i, $1 \leq i \leq n$, is connected to an object predicate node (which stores the predicate symbol p) and each a_i is connected to the OAP node p_i.

It is assumed that each object constant node knows the addresses of those object predicate nodes and OAP nodes which are connected to it. Similarly, each OAP node and each object predicate node know the addresses of its associated attributes. Inside an object constant node, the set of addresses to its connected object predicate nodes are grouped based on attribute positions; these groups are called address groups.

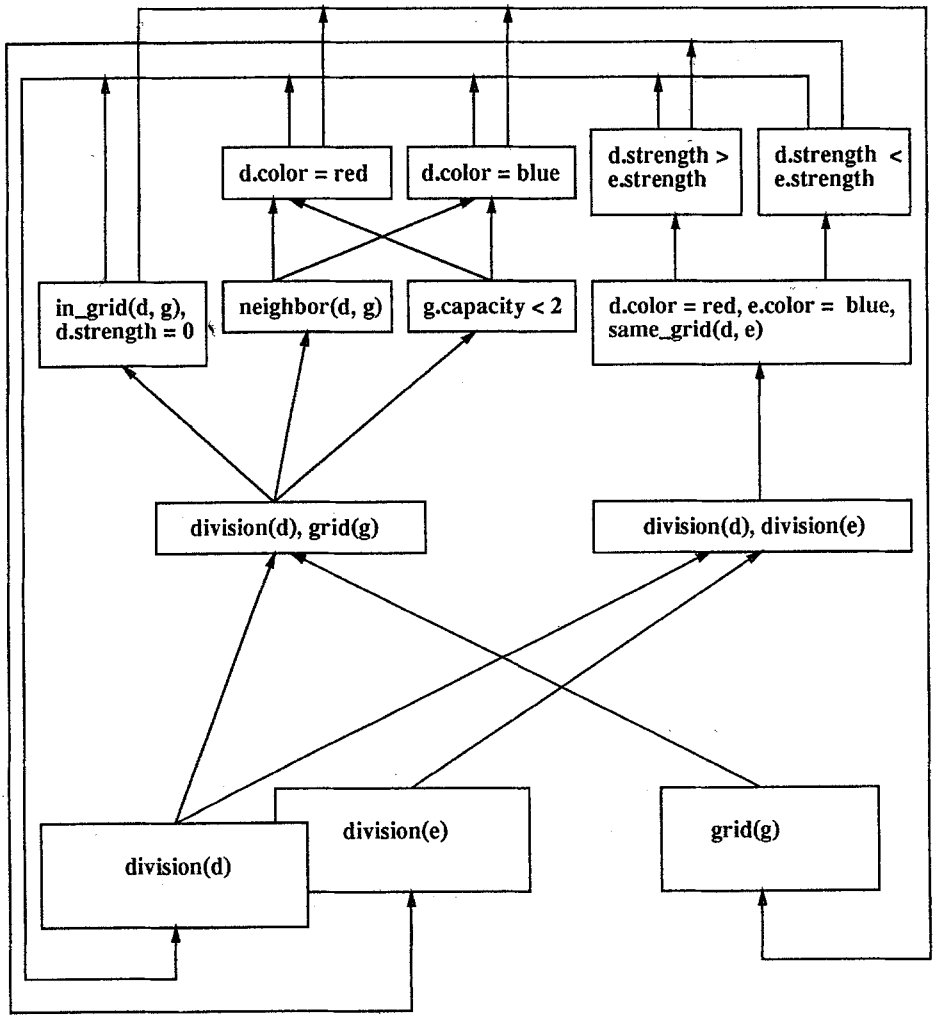


Figure 4.3: A Rule Network

Consequently, an OAP node is used to interface the object network and the external environment. It acts as a buffer node which gets a message from the outside of the network and regenerates the same message to the connected nodes in the network. It also combines the replies from the connected nodes in the network and sends a reply to the outside of the network. With OAP nodes, we can hide the actual fact network from the outside of the network. Conceptually, the use of OAP nodes employs an indirect addressing mode in order to avoid the establishment of connections to object constant nodes whenever a rule is evaluated. Consequently, when an operation is evaluated, only the connections from each predicate argument nodes in the operation network (see below) to the OAP nodes in the object network need to be established. Within each node, it stores the identity (node type) of the node, its value (if it is an object constant node), some dedicated control sequences, and the addresses of the nodes it is connected to. Inside an object constant node, the addresses of its connected object predicate nodes are recorded and grouped according to their positions. Note that each distinct object is uniquely stored.

On top of the objects, a join operation $f = f_1 \wedge ... \wedge f_n$ is built as an operation network, where each f_i denotes an operation predicate node which is connected to a set of operation argument nodes (where each of them could be an operation constant node or an operation variable node). A logical expression can be associated with each variable argument node which designates any select criteria for the argument. Depending on its position in the argument list, is connected to an appropriate OAP node, which should exist in the object network already. We shall assume that a controller exists on top of all the objects and each object knows the address of the controller. However, the controller only knows the address of each operation predicate node when an operation network is constructed.

4.6.2 Object-Oriented Rule Evaluation

Consider an operation and an object of class p with attributes $a_1,...,a_n$. If the object knows the structure of the operation, we can have the following observations:

1. For each constant a, note that a can be an attribute of more than one object, assume we know which variables of which operation predicate nodes can be instantiated by a. The object predicate $p(a_1,...,a_n)$ can instantiate the operation predicate node $p(x_1,...,x_n)$, where each of x_i, $1 \leq i \leq n$, is a constant or a variable, and c_i is the select criteria associated with x_i if it is a variable, if for each i, $1 \leq i \leq n$, the following conditions are held:

   (a) $a_i = x_i$, if x_i is a constant;

   (b) $c_i(a_i)$ is true, i.e., a_i can instantiate x_i, if x_i is a variable.

   Furthermore, for any x_i and x_k, $1 \leq j,k \leq n$, $j \neq k$, if $x_j = V$ and $x_k = V$ for some variable V, we say there is an *intra-predicate tupling condition* between x_j and x_k. Clearly, for any intra-predicate tupling condition between x_j and x_k, a_j should be equal to a_k. In this case, we say a_j satisfies all the intra-predicate tupling conditions for p.

   The above implies that each object predicate object, which is connected to its attributes, can determine if it can instantiate a predicate of the operation. This is because (1) Each of its attributes knows which variable instances of the predicate can be instantiated by itself, and such knowledge can be collected at the object predicate object; (2) The object knows the structure of the operation,

and therefore knows which variable instances are associated with, and in what order in, each predicate node of the operation.

2. For each constant a, note that a can be an attribute of more than one object, assume we know which variables of which predicates of an operation can be instantiated by a and group all such (*variable instantiated – variable*) pairs into a set $BVIS_a$, where $BVIS$ stands for *Binding Variable Instances Set*. For an object predicate $p(a_1,...,a_n)$, it can satisfy the operation $f = f_1 \wedge ... \wedge f_n$ through a predicate $p(x_1,...,x_n)$ if the following conditions are held:

   (a) $p(a_1,...,a_n)$ can instantiate $p(x_1,...,x_n)$;

   (b) For any variable V, if $x_j = V$, and there exists another predicate $g(y_1,...,y_r)$ where $y_k = V$, we say there is an *inter-predicate tupling condition* between x_j and y_k. Clearly, for any inter-predicate tupling condition between x_j and y_k, where $x_j = V$ and $y_k = V$, then both (a_j,x_j) and (a_j,y_k) should be contained in $BVIS_{aj}$. In other words, a_j should instantiate both x_j and y_k.

   The above implies that each object, which is connected to its attributes, can determine if each of its attributes can satisfy all the inter-predicate tupling conditions for an operation. (1) Each of its attributes knows which variable instances of the operation can be instantiated by itself, and such knowledge can be collected at the object predicate node; (2) The object predicate node knows the structure of the operation, and therefore knows which variable instances are associated with, and in what order in, each predicate node of the operation.

3. A *partial solution* for an operation B can be formed in an object predicate p, whose attributes are assumed to be $a_1,...,a_n$, through a predicate node $p(x_1,...,x_n)$, if each a_i can satisfy the inter-predicate tupling conditions. Note that if a partial solution can be formed for B, it is not guaranteed that a complete solution exists, where a *complete solution* is a partial solution in which all variables are instantiated. A partial solution of B can be expressed as $\alpha_1,...,\alpha_n$, where α_i, $1 \leq i \leq n$, is either a substitution (expressed in the form of a/V, meaning a constant a instantiates a variable V) or a variable. Two partial solutions $\alpha_1,...,\alpha_n$ and $\beta_1,...,\beta_n$ for B can be merged into another, yet more complete, partial solution $r_1,...,r_n$ such that

   (a) $r_i = \alpha_i$, if α_i is a substitution and β_i is a variable;

   (b) $r_i = \beta_i$, if β_i is a substitution and α_i is a variable;

   (c) $r_i = \alpha_i$ (or β_i), if both α_i and β_i are substitutions or variables (Note that in this case α_i and β_i have to be equal).

   Two partial solutions for B are *incompatible* if they cannot be combined.

4. Based on 2 and 3, each object predicate node can form a partial solution, if exists, for an operation. Consider an operation $B = p_1,...,p_n$, where (1) Each p_i has arguments $(x_{i1},...,x_{iu(i)})$; and (2) p_i and p_{i+1} share at least one variable. A complete solution for B can be formed as follows:

   (a) Each object predicate node whose predicate symbol is p_1 determines a partial solution, if exists, for B. If the partial solution is a complete solution, stop.

(b) Each object predicate node whose predicate symbol is p_1 which has a partial solution for B passes the partial solution to each object predicate node whose predicate symbol is p_2 which is connected to it through the common constant(s) and asks each connected object predicate object whose predicate symbol is p_2 to expand the partial solution. This can be done in each of the object predicate nodes whose predicate symbol is p_2 by combining the passed partial solution and the partial solution determined by itself. If they are not compatible, stop; if the combined partial solution is complete, stop.

(c) Each object predicate node whose predicate symbol is p_2 which has a partial solution for B passes the partial solution to each object predicate node whose predicate symbol is p_3 which is connected to it through the common constant(s) and asks each connected object predicate node whose predicate symbol is p_3 to expand the partial solution. This can be done in each of the object predicate nodes whose predicate symbol is p_3 by combining the passed partial solution and the partial solution determined by itself. If they are not compatible, stop; if the combined partial solution is complete, stop. This process is repeated until each of the object predicate nodes whose predicate symbol is p_n is processed.

4.7 Conclusion

In this chapter, we have presented the design of a parallel object-oriented simulation environment. The environment provides an object-oriented interface that allows the control and the communication aspects of active objects be easily specified. It also allows complex objects be composed easily. Representing the control of each active object as a production system, the environment considers the state space of each object in determining the impact of any change made to the object.

While the idea of rule-based simulation is not new (see, e.g., [282] [387] [608]), our approach converts a large number of active objects into a much smaller set of operations applying to sets of passive objects. As a consequence, common computations can be shared among different objects. Compared to a fully distributed, asynchronous approach, it avoids the problem of managing a large number of active objects when the available physical resources are limited. An alternative object-oriented rule evaluation approach has been also proposed for parallel environments which contain a large number of relatively simple processing elements.

Chapter 5

Transition From A Relation To Object Model Implementation*

Bharat Bhargava[†], Yin-he Jiang[‡], Jagannathan Srinivasan[§], Prasun Dewan[¶]

5.1 Modeling Complex Data

Emerging "next generation" applications such as multimedia, engineering(CAD/CAE), and geographic information systems require managing databases that are more complex than those in applications such as banks and airline reservation systems. Current RDBMS (Relational Database Management Systems) are limited in their support for these applications. The limitations typically include lack of support for complex data structures and operations, integrity checking, and triggers, etc. They are confined to simple, predefined alphanumeric types. On the other hand, OOPLs (Object-Oriented Programming Languages) and OODBMS (Object-Oriented Database Management Systems) allow developers to define their own object/data types. These user-defined types are indistinguishable from the pre-existing types from the system view point.

Modeling hierarchical data (composite object) in RDBMS is cumbersome. The developer has to create numerous tables to model the complex data. When being requested to retrieve data in the composite object, the system has to join many of these tables on the fly which is a time-consuming job.

In OODBMS, an object can consist of a collection of other objects by two ways. One way is through embedded objects. That is, an object is stored directly in another object as one of its fields. The other is through pointer, or object referencing. In this case, the address of an object is stored in another object as a field. Because there are object identifier to each object including composite object, explicit links between objects and information to cluster data, the performance is improved when such objects are stored in an OODBMS. The time-consuming join operations in RDBMS is saved by direct object referencing in OODBMS.

Aimed at supporting the new applications, researchers have adopted two different ways. One is to build OODBMS from ground up (revolutionary approach). Such systems include O_2 [151], *ObjectStore* [347] and *GemStone* [83]. The other is to extend an existing RDBMS to support abstract data types (evolutionary approach).

*This research is supported in part by a grant from AIRMICS and UNISYS.

[†]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907
[‡]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907
[§]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907
[¶]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907

Such systems include *POSTGRES* [557], *Starburst* [379] and *O – Raid* [153]. Both ways have been successful and have their advantages and disadvantages. While the revolutionary approach is focused on supporting abstract types from ground up and allows the system to be finely tuned, the development cost and time is relatively great. On the other hand evolutionary approach allows reuse of RDBMS software which is cost-effective and could shorten the developing time of the software significantly. The flexibility of the design and implementation in this approach is restricted and the performance is bounded by the underlying system. In the following discussion we will be focused on the evolutionary approach. We start on surveys of systems using this approach and focus on the O-Raid system we developed at Purdue University.

5.2 Survey of Extended Relational Systems

POSTGRES In addition to business applications, POSTGRES expands to supports object management and knowledge management. Object management entails storing and manipulating nontraditional data types such as bitmaps, icons, text and polygons. Knowledge management includes the ability to store and enforce a set of rules that are part of the semantics of a application. Such rules enforce integrity constraints and allow the derivation of data that is not directly stored in the database [557].

It also supports class, class inheritance and user-defined functions in the DBMS. In the POSTGRES approach, Classes are modeled by relations. A tuple within a relation represents a class instance. The attributes define the instance variables. A relation may inherit the attributes and the functions defined for another relation. POSTGRES defines a relation called *superclass* relation. This has three attributes, *class, superclass and seqnum.* The *class* and *superclass* attributes store the class name and the superclass name respectively. The *seqnum* attribute stores a number, used for resolving conflicts.

Object referencing in POSTGRES is realized through storing *procedure* valued attributes in a relation. Such attributes can contain a QUEL *query* to retrieve tuples from other relations. Accessing the procedure valued attribute automatically executes the *query,* retrieving the desired data. The procedure valued attribute thus serves as a pointer to another object.

Starburst Starburst supports user-defined types through extending a relational DBMS. Its goal is to provide the desirable features of object-oriented, logic, deductive and other DBMS technologies, while still retaining all the strength of a relational DBMS [379]. It supports user-defined production rules. These rules could form a forward chain that enforce data constraints through performing actions when predefined conditions are satisfied (triggered).

Starburst supports user-defined types and functions. The user-defined functions must be statically linked with the rest of the Starburst system. Dynamically linking a user-defined function with the system is underdevelopment [379]. There two ways of storing complex objects in Starburst. One is to store the entire object in a 'long field'. This means that the system has minimal ability to apply selection predicates based on the contents of this 'long field' in a query. The other is to store the components of an object as rows in a table. This allows objects to constructed by composing these rows using different relational views. Similar to POSTGRES, Starburst supports object referencing by embedding queries in data fields. These queries defined the data objects to be constructed.

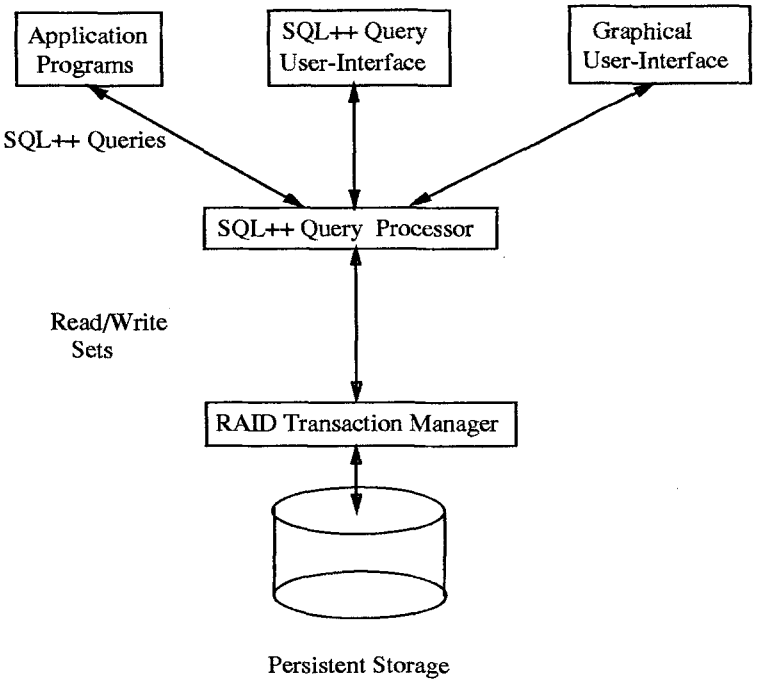


Figure 5.1: O-Raid System Layout

O-Raid O-Raid extends in layers the implementation of an existing distributed relational system called RAID [61] (See Figure 5.1). O-Raid integrates the relational and object models. As a result it retains the simplicity of the relational model while providing the functionality of the object model. Specifically, the relations in O-Raid can contain attributes of user-defined types, i.e., the tables can contain objects in their columns. The user-defined types are generally *classes*. A class is a set of objects having common features. In the class definition, the common features of objects are defined by instance variables (members) and methods (functions). The state of an object, i.e., the values of the instance variables in the object, can only be changed though the execution of methods defined in the class.

O-Raid supports both inter-object referencing (or pointer referencing) and intra-object referencing (or embedded object referencing, where an object is stored within another object). In queries persistent pointers to user defined types in attributes of a relation is allowed. A persistent pointer in O-Raid is represented by three integers, object identifier (OID), relation identifier (RID), and a offset (OFFSET) [420] that uniquely identify the object and its class. Objects, classes, and inheritance are supported together with a predicate-based relational query language. O-Raid objects are compatible with C++ objects and may be read and manipulated by a C++ program without any "impedance mismatch".

RAID	O-Raid
Language: SQL	Language: SQL++
Tables of simple types only e.g. integer, char	Tables of simple types and any user-defined types e.g. structures and functions
Predicates of simple comparisons e.g. =, >, <	Predicates of comparisons and user-defined fuctions e.g. contain_key()
Expressiveness of query: Sometimes may be non-intuitive and long	Expressiveness of query: Intuitive and terse

Figure 5.2: Query Language (RAID vs. O-Raid)

5.3 O-Raid System Design and Implementation

In O-Raid [153, 60] we are exploring the extensions to the relational system to implement an object-oriented system.

5.3.1 Expand Query Language

We have extended the standard SQL query language called SQL++, to support queries involving objects [420]. Figure 5.2 compares the features of the query languages supported in RAID and O-Raid.

In queries the objects are manipulated (retrieved or updated) using methods defined in the object class in C++. O-Raid [153] is built on top of the RAID [61] distributed relational database system. An O-Raid relation needs to be mapped to a RAID relation. Specifically, a relation involving user-defined types has to be transformed to a new relation with attributes of simple types. The object-relation data model poses new requirements for the schema specification facility, also referred to as *Data Definition Language (DDL)* facility:

- *Relation containing objects*: To support an extensible collection of data types, a facility for registering classes (user-defined types) is essential. The structure and methods for complex objects needs to be registered with the database.
- *Dynamic loading and execution of methods* In O-Raid the methods used for manipulating the objects are dynamically loaded and executed. The facility for

dynamic loading and execution of method needs method symbol name as opposed to method signature. Thus when a class is registered we need to generate a table mapping all its method signatures to corresponding method symbols.

- *Schema for databases spanning multiple sites* In O-Raid the relations can be replicated at all database sites (*full replication*), or the relation can be replicated at selected set of database sites (*partial replication*). The replication of selected fragments of a composite objects is also supported.

5.3.2 Extend Data Definition Facility

To create a database, users must specify the schema on which the structures of the database is built. A schema specification facility called *dbedit* is developed for specifying database schemas and to generate the corresponding meta information in the database. Classes are defined using C++ [560] programming language and they specify the structure as well as the methods that manipulate the data. The extensions to the traditional DDL include:

- *Register* class-name. This allows a user to store new user-defined types in to a database including the code that defines and implements the classes and functions (methods).
- *Create table* table-name ...

  This allows users to create a table with its column to be of user-defined types and to be able to invoke new functions in order to construct objects and filter data in the queries.
- *Distribute table* table-name. This could specify a table to be replicated over multiple sites. It could be used for selective replication of data.

We will illustrate the schema specification facility of O-Raid system through an example. Suppose we want to creat a two-site *document* database:

- The *Document* contains 4 sections, namely *introduction, indexing, replication* and *conclusion.*
- Since 2 users from 2 different sites (raid9 and raid11) have different access patterns, it requires that *introduction* and *conclusion* sections be fully replicated, section *indexing* is only created at raid9 site, and section *replication* only at raid11 site (See Figures 5.3).

The *Document* and *Section* classes could be defined in C++ as follows (see Figure 5.4 and 5.5):

The above user-defined classes can be registered to the system by the command:

REGISTER_CLASS Document;

The components of REGISTER_CLASS are shown in Figure 5.6, in more details:

- **C++_Parser**: *Input*: C++ programs; *Output*: Class information, including class name, class size, superclass, member types and their size in bytes.
- **Gen_Relation**: *Input*: the output of the last step; *Action*: generate metadata files to store the class relation and class attribute information.

  e.g. A tuple *4 0 1 "Document" "pintro_sec" 7 9 1 "Section"* in the CLASSATTRIBUTE means that attribute *pintro_sec* belongs to relation *Document*, is of

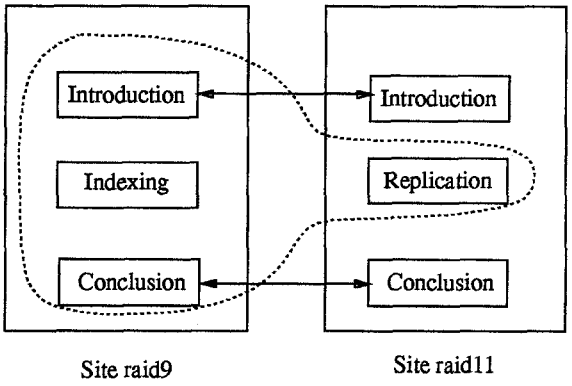


Figure 5.3: A 2-site Document Database. Identical copies of sections are connected by lines with arrows. Site raid9's view of the documents enclosed in the dotted boundary.

type *class Section* spanning column 7 to 9 and is a pointer to object (The 1 after 9; if 0, it means embedded object). *4 0 1* says that tuple id is 4, version is 0 and 1 means the tuple is used.

- **Gen_Method_Map**: *Action*: Compile the C++ file and generate the corresponding object file (`.o`). Generate a table mapping all its method signatures to corresponding method symbol names.

  e.g. *Document::Document(char *, char *)* is a method signature, and *__8DocumentPcT1* is a method symbol name, which dynamic loader can use directly to execute the constructor method *Document()*.

- **Update_Class_Method**: *Action*: The object files and method signature tables created in the previous step are stored in the database. The table is stored in a metadata file called CLASSMETHOD.

Support Dynamic Loading and Execution of Methods: When user invokes a function (method) in her query, the method signature is directly available from the query. But the dynamic loader needs the method symbol name of the precompiled method code for execution. Thus the *Method Signature Table* is consulted to convert method signature to method symbol name before being able to dynamically execute a method during query processing. The table is generated as follows:

- invoke the UNIX command *nm* on the class object file (`.o`) and select the mangled names for the methods.
- process them with a *demangler* program and build the desired table. The demangler program generates the method signature from a mangler method symbol.

A method signature table of our example *Document* class is shown in Figure 5.9.

Schema Specification The procedure of specifying the database schema of our example to the O-Raid database system is shown as follows:

```
// Document.h
class Section { // both raid9 and raid11
 public:
   char heading[TITLE_LEN];
   char text[MAX_TEXT];
   Section(char *t);
   void printobj();
};

class Section9 { // for raid9 site
   ...
};

class Section11 { // for raid11
   ...
};


class Document { // both raid9 and raid11
 public:
   char            name[MAX_NAME];
   char            title[TITLE_LEN];
   class Section   *pintro_sec;
   class Section9  *pindex_sec;
   class Section11 *preplic_sec;
   class Section   *pconcl_sec;
   Document(char *n, char *t);
    void printobj();
};
```

Figure 5.4: Header file Document.h for definition of class *Document*.

```
// Document.cc
// Constructor and Methods definitions
Section::Section(char *t) {
   strcpy(heading, t);
   strcpy(text,">>");
}
void Section::printobj() {
   if(this!=NULL) {
     printf("Heading: %s\n", heading);
   }
}
// Similarly for Section9 and Section11
// For class Document
Document::Document(char *n, char *t) {
   strcpy(name, n);
   strcpy(title, t);
   pintro_sec = NULL;
   pindex_sec = NULL;
   preplic_sec = NULL;
   pconcl_sec = NULL;
}
void Document::printobj() {
   printf("Name: %s\n", name);
   printf("Title: %s\n", title);
   pintro_sec->printobj();
   pindex_sec->printobj();
   preplic_sec->printobj();
   pconcl_sec->printobj();
}
```

Figure 5.5: Source file Document.cc for Constructor and Method code.

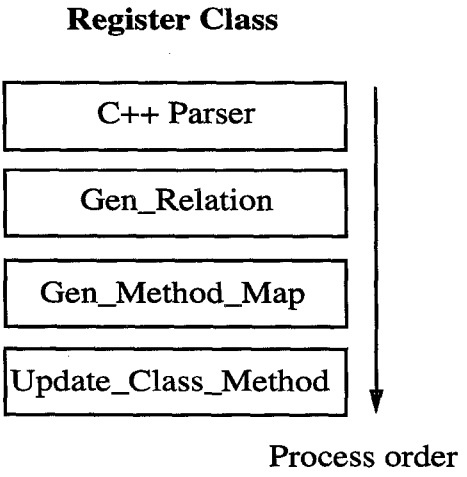


Figure 5.6: The components of REGISTER_CLASS command.

```
0 0 1 "Section" "s" 5 6 0 "Section"
  ...
3 0 1 "Document" "d" 5 18 0 "Document"
4 0 1 "Document" "pintro_sec" 7 9 1 "Section"
  ...
7 0 1 "Document" "pconcl_sec" 16 18 1 "Section"
8 0 1 "documents" "d" 3 16 0 "Document"
```

Note: From left to right, column by column it is "tuple id", "version", "used flag", "relation name", "attribute name", "start column", "end column", "level of object indirection", "class name".

Figure 5.7: Metadata file: CLASSATTRIBUTE

```
0 0 1 "Section" 10 640 "Document.o"
1 0 1 "Section9" 11 640 "Document.o"
2 0 1 "Section11" 12 640 "Document.o"
3 0 1 "Document" 13 208 "Document.o"
```

Note: In the last row "Document" is a class relation; 13 is the class id for that relation; 208 is the number of bytes for that class; "Document.o" is the name of the object file in which the class is implemented.

Figure 5.8: Metadata file: CLASSRELATION

```
0 0 1 "___7SectionPc" "Section::Section(char *)"
1 0 1 "___8DocumentPcT1" "Document::Document(char *,
                                              char *)"
...

4 0 1 "_printobj__7Section" "Section::printobj()"
5 0 1 "_printobj__8Document" "Document::printobj()"
...
```

Note: Columns left to right are "tuple id", "version", "used flag", "method symbol name", "method signature".

Figure 5.9: Method Signature Table: CLASSMETHOD

Invoke dbedit on a database name **(dbedit Comp)**
with the following DDL commands:
install; {* initialize database directory *}
config(raid9.cs.purdue.edu /uraid9/databases);
{* declare raid9 and its directory as one site
Similarly for raid11 *}

registerclass Document;
{* Register classes defined in Document.cc
into the database Comp *}
classread; {* Reread the existed classes *}

{* A table with an attribute of type Document *}
create table documents(Document d);

{* 2-site replicated tables *}
distribute table documents, Document, Section
(raid9.cs.purdue.edu /uraid9/databases);
distribute table documents, Document, Section
(raid11.cs.purdue.edu /uraid11/databases);

{* Single site tables *}
distribute table Section11(raid11.cs.purdue.edu
/uraid11/databases);
distribute table Section9(raid9.cs.purdue.edu
/uraid9/databases);

replicate; {* Action *}
quit;

Note: The comments of each command **(in bold face)** are put in between {* and *}.

5.3.3 Data Manipulation Language (DML)

Users interact with the database directly through DML facility by submitting queries. Extensions to the queries include:

- *Insert into* table-name: <attributes>. Here the attributes could be a simple type value, e.g. integer or a constructor method (function) call with arguments.
- *Select * from* table-names *where* predicate. In the "where" clause, the predicate can not only be simple attribute such as *name = "John"* but also involve filtering functions supplied by users such as *contain_key("database")*.
- Introduce variables that allow users to store, retrieve and update intermediate results by other queries. For example, *assign to* variable: *select query*; *update* variable *set* statement.

  Two kinds of variables are supported in O-Raid user interface. One is the temporary variable whose life time is the login session of the user. The other is the global variable whose life time is permanent. The global variable has the same

effect as a relation. Compared with the global variable, temporary variable has much less overheads in execution of queries.

Instantiate the Database We continue with our example. The database is instantiated by submitting queries to the user interface as follows:

- Invoke RAID servers and UI process on the database */uraid9/databases/Comp.*
- Input the following DML commands to UI:

```
insert into documents:<Document("ind_rep",
"Indexing and Replication Experiments")>;

create variable @s: &Section("Introduction");
update documents set d.pintro_sec = @s;

create variable @s: &Section9("Indexing");
update documents set d.pindex_sec = @s;

create variable @s: &Section11("Replication");
update documents set d.preplic_sec = @s;

create variable @s: &Section("Conclusion");
update documents set d.pconcl_sec = @s;
```

The **insert into** command constructs an in-memory object through constructor method *Document()* defined in Figure 5.5. The object is converted into tuple and inserted into relation *documents*.

The command **create variable** @s: &Section("Introduction") first construct an in-memory object through constructor method *Section()* defined in Figure 5.5. The address of the object is assigned to temporary variable *s* (denoted by prefixing character @).

The command **update** documents **set** d.pintro_sec = @s sets the attribute *d.pintro_sec* of the relation *documents* to point to what the variable *s* is pointing to, which is the object created in the previous command.

Query Execution in O-Raid After a user query is parsed, related relations are brought into the memory. The attributes of the relations involved in the query are checked to see if they are of user-defined types, for example, `Seminar` is a user-defined type. For attributes of user-defined types, e.g. the attribute `s` in the example `select` query, an object is constructed for each tuple in the relation `colloquia`. Here each tuple contains an entry for a seminar. The query predicate is then evaluated on the constructed object and depending upon the true/false value the object is selected or rejected.

If the user-defined methods are involved in the predicate, the methods signatures are mapped to the methods symbol names. For example, the method signature `Keys::contain_key(char *)` in the sample `select` query is mapped to the method symbol name `_contain_key_4KeysPc` via the methods symbol table created

when `register_class` command is invoked. Based on the symbol names, the precompiled method code is then loaded and linked dynamically with the process that is handling the user query and the method is executed. For update queries in the end of query processing, a translation from objects to tuples of the relation (reverse conversion) takes place before writing relations to the database. O-Raid interact with the underlying RAID system through *Read/Write* operations and *init_transaction*, *commit_transaction* primitives provided by the RAID system.

Graphical User Interface A graphical user interface called O-UI has been engineered using Suite system [152] to allow access and direct manipulation of O-Raid database relations and objects [434] (Figure 5.10). This interface facilitates the users in composing their queries to an extent that parts of the user queries are provided automatically.

Earlier, we had built a simple teletype interface for O-Raid called S-UI, which accepted a query typed by the user and displayed the query result (if any) in a tabular form. The interface was simple and portable but had several limitations. The display of relations with a large number of tuples or attributes was difficult to view. The tabular display of data was awkward for relations containing objects. A flexible mechanism for display of objects was desired. We wanted to display objects through a special *display* method defined for that class, which specifies how the objects of that class should be displayed[6]. Another limitation was that the query result could not be reused for a subsequent query. This increased the effort required to obtain desired information through a series of steps (query refinement). Also, the manipulation of relations could only be done by specifying update queries.

To overcome these limitations we embarked on building the O-UI graphical interface. We wanted to build the graphical interface based on the direct manipulation paradigm [521], with features such as mouse based interactions, pop-up menus, windows, icons, and graphical display of data. Another goal was to minimize the amount of information the user has to know (such as query language syntax, etc.) and reduce the data that needs to be typed.

5.4 Performance Studies

We have conducted experiments on O-Raid to identify the overheads involved in executing SQL++ queries [540]. This allows us to assess the effort required to extend a relational system with objects, evaluate the efficiency and usability of some of the design and implementation decisions we have taken, and gain experience in using database objects. We found that about 15% overheads incur on the response time of query processing. We consider this reasonable as O-Raid provides more expressive power and convenience to the users than its relational counterpart.

For replicated database, we build a two-site database, using Read-One-Write-All (ROWA) replication control algorithm. Table 5.1 and 5.2 show the comparison of the overheads under *local*, *remote* and full replication access for `select` and `insert` queries respectively. The full replication scheme has the benefit of lowest cost for local *read* access, but has the highest cost for *write*.

Based on this observation, a selective replication scheme for composite object is proposed in [541]. The composite object called `document` is created with four `section` subobjects. Two users are simulated to access the `document` object. The access probability for the two user sites is shown in Table 5.3.

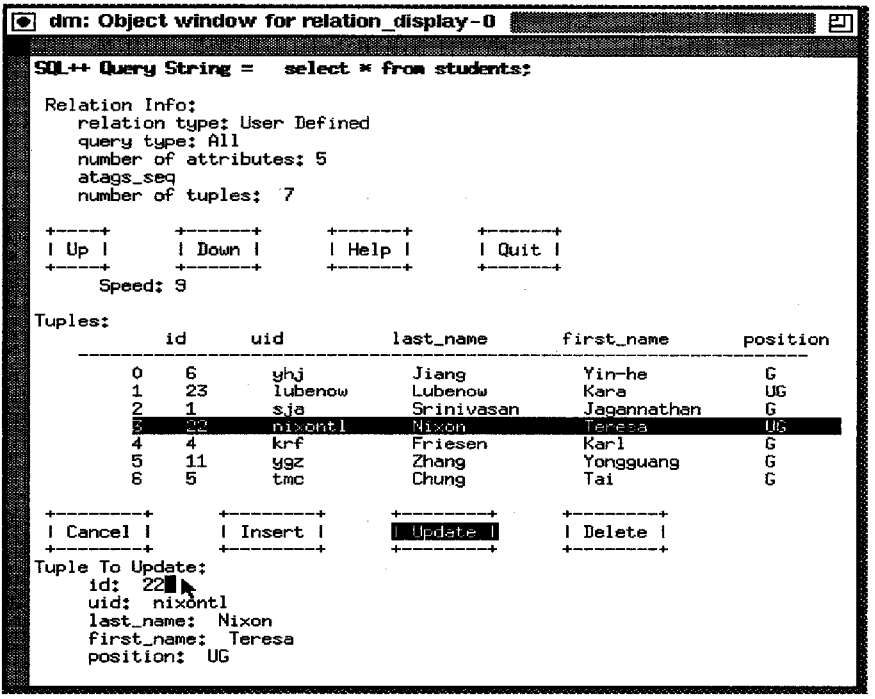


Figure 5.10: Query Interface and Relation Display Window

Query Type	Trans-lation	Read	Writes	TOTAL
local	132	318	121	589
	[131,132]	[317,319]	[118,123]	[585,592]
remote	133	354	167	673
	[131,135]	[352,357]	[164,171]	[668,677]
full replic	125	321	120	585
	[125,126]	[320,322]	[116,123]	[581,588]

Table 5.1: Comparison of processing time (in ms) for select queries with objects on 25 tuples

Query Type	SQL parsing	Trans-lation	Writes	TOTAL
Local Insert	10 [9,10]	7 [7,7]	172 [167,176]	193 [189,197]
Remote Insert	10 [10,11]	7 [7,7]	232 [226,239]	255 [248,261]
Two-site Insert	10 [9,10]	7 [7,7]	242 [234,250]	263 [255,271]

Table 5.2: Comparison of processing time (in ms) for insert queries with objects

Section Name	site1	site2
"introduction"	0.25	0.25
"indexing"	0.375	0.125
"replication"	0.125	0.375
"conclusion"	0.25	0.25

Table 5.3: Access probabilities by site1 and site2 users

Table 5.4 and 5.5 contain the response time for `select` query and `update` query under three replication schemes, namely *full replication*, *selective replication* and *no replication*. In the *selective replication* scheme, the sections "introduction" and "conclusion" are fully replicated on the two sites. However, only a single copy of the "indexing" and "replication" sections is maintained at site1 and site2 respectively. The two sites can both be on a LAN, or one at local and the other across the Internet.

From the above data we could see that the selective replication scheme shows strong benefits over the fully replicated and the single copy schemes in the WAN as well as in the LAN environment, considering the combination of retrieve and update operations [540]. The selective replication scheme allows users to fine tune the replication and achieve high performance.

Emulated Host	full replic		seletive replic		no replic	
	site1	site2	site1	site2	site1	site2
LAN	238	233	249	253	236	299
ecn.purdue	252	240	256	260	257	360
uiuc	256	256	278	299	248	552
uta	252	247	293	342	247	806
helsinki.fi	251	239	341	431	245	1534

Table 5.4: Select query response time (ms)

Emulated Host	full replic		seletive replic		no replic	
	site1	site2	site1	site2	site1	site2
LAN	331	335	321	322	279	353
ecn.purdue	371	375	348	350	295	420
uiuc	438	452	392	353	287	604
uta	527	526	440	476	288	861
helsinki.fi	812	814	703	707	301	1619

Table 5.5: Update query response time (ms)

5.5 Research Issues

We are currently investigating several issues including indexing of relations containing objects [295], selective replication of parts of composite objects [59], providing integrated access to a variety of data sources through the federated objects [585], and supporting transactions on fragmented composite objects [584].

Replication The traditional scheme of replicating an object in its entirety, for increasing the availability, would incur high performance and storage overheads when applied to composite objects such as multimedia documents and large pieces of software. Our extension allows a replication scheme, which allows replication of selected parts (subobjects) of the composite object. Such *composite object replication* can be used to tune the replication granularity to meet both the availability and performance requirements of distributed applications, and at the same time minimize the storage costs. For example, a "document" object may be a composite object consisting of pointers to four *section* objects, namely the "introduction", "indexing", "replication", and "conclusion". Each of these sections can be independently distributed and replicated by choice.

Indexing We are extending O-Raid to support *path indices* [58, 295]. Path indices allow efficient selection of composite objects based on the nested member attribute values. The goal is to identify the cost of creating and maintaining indices and the performance benefits resulting from path indices. The study will help us develop policies regarding use of path indices for different applications. We are considering support of indexing on methods that represent derived attributes. For methods with no arguments, indices will be built after the method values are precomputed. This technique of precomputation cannot be used for methods with arguments. In general, for these methods there may not be any efficient strategy for maintaining indices. However, we are looking for a partial solution. We plan to identify characteristics of the commonly occurring methods (with arguments) and develop suitable index structures for them.

Fragmentation We use *object fragmentation* as the basis for the transaction processing mechanism, which to a great extent avoids intersite communications and blocking delays [584]. In this approach much of the transaction processing is handled at the local participant database level. This makes the scheme especially appealing in the presence of local autonomy requirements. As opposed to the static fragmentation schemes, this approach is based on a demand-driven *dynamic reconfiguration* of object

fragments. The reconfiguration is supported by two mechanisms: *object reincarnations* and *negotiation protocols*. Object reincarnation models situations such as cyclic consumption of resources where the object fragment values have to be reset after a certain time interval. The negotiation protocol reconfigures the object fragments in a given time interval to meet the varying demand for resources. We have developed a formal model for different fragmentation schemes and designed constructs for specifying fragment distribution. We have also designed scheme for supporting object reincarnations and negotiation protocols in a distributed database system.

Composite/Federated Object We model existing related distributed data by a *composite* object, where each data source is treated as a fragment of the composite object. A collection of methods are defined to allow manipulation of the related distributed data in a controlled and consistent manner. The composite object so created, referred to as *federated object*, is a pragmatic approach to data integration. Unlike, heterogeneous database systems, in this approach each data source is not required to have full database system capabilities. Instead, the data sources can be simple files, or application generated binary files. Federated objects are especially suitable for computer-support-cooperative work (CSCW), where coordination and communication among a group of people is required, and the type of collaboration is dynamic in nature. A *dynamic data reconfiguration* mechanism is proposed to allow efficient access of data. Federated objects are constructed from existing data resources using a C++ class library, which provides the mechanisms needed for access and manipulation. The *toolkit* approach provides flexibility of adding application-specific mechanisms during the data integration process.

Chapter 6

An Object-Oriented Knowledge Model for KBMS-supported Evolutionary Prototyping of Software Systems

Stanley Y. W. Su*, Yuh-Ming Shyy†

6.1 Introduction

6.1.1 Motivation

The development of complex software systems is a costly endeavor. If prototypes can be rapidly constructed to test the structural and behavioral properties of these systems as the developers gain more knowledge about their requirements, then complex systems can evolve from a series of prototyping efforts [28, 29]. In this chapter, we take a knowledge base modeling approach to evolutionary prototyping of software systems by treating each prototype system as a high-level executable model of the target system, which defines the structural and behavioral properties of the target system at any level of abstraction (from a large program module to a single program statement) as desired by the prototyper. The executable model evolves gradually through a series of schema modifications and refinements to provide more and more details about the requirements and implementations of the target system. At each stage of evolution, the model (i.e., the prototype) can be executed to test its functionalities and performance. All the debugging, modification, and maintenance can therefore be performed directly against the executable model throughout the software lifecycle as shown in Figure 6.1.

As we all know, all software systems are computer programs and, based on Wirth [618] and Kowalski [332], we have the following formula: "**Program = Data Structure + Logic + Control**". If a knowledge model can uniformly model all types of software systems in terms of their (i) structural properties (corresponding to the data structure aspect of a program and the control structure among program segments), (ii) operations/methods (corresponding to the procedural semantics of program algorithms), and (iii) knowledge rules (corresponding to the declarative semantics of program logic and control), then any software system in the traditional concept can be evolutionary modeled by this knowledge model throughout its software lifecycle. Note that, although the semantics represented by rules can be implemented in methods, high- level declarative rules make it much easier for a prototyper to clearly specify the

*Department of Computer and Information Science, University of Florida, Gainesville, FL 32611, U.S.A.

†Versant Object Technology, Menlo Park, CA 94025, U.S.A.

semantics instead of burying the semantics in application codes and thus simplify the tasks of testing, modification, debugging, and maintenance. It is not necessary to make the traditional distinction among software systems (e.g., application systems, operating system, and data base management system) because all of them are executable models of the underlying object-oriented knowledge base as shown in Figure 6.2. The structural and behavioral properties of all object classes which model programs and application domain objects are stored in an object-oriented knowledge base under the management of a Knowledge Base Management System (KBMS) and can be shared and reused among the users of various application domains.

We have extended an object-oriented semantic association model OSAM* [564, 565, 634] with reflexivity and control associations as an extensible framework for KBMS-supported evolutionary prototyping described above. The advantages of this approach are three-fold. First, by using a single unified knowledge model and schema notation, we eliminate the mismatch between the traditional data-oriented models [108, 266] and the process-oriented models [142, 426, 452, 406] to support both structural and behavioral prototyping within an object-oriented framework. Secondly, all types of software systems, application domain objects that these systems deal with, and related meta information can be uniformly modeled by the knowledge model and managed and processed by an underlying KBMS, or the so-called "Next-Generation Database Management System" [196, 4], which uses this knowledge model as its underlying model. Thirdly, instead of serving as throw-aways or being limited to conceptual design, the model of a target system can evolve from specification to implementation throughout the software lifecycle as shown in Figure 6.1. We have also developed a knowledge base programming language called K as a high-level interface to define, query, and manipulate the knowledge base as well as to code methods [522, 523]. In this chapter, we shall concentrate on the knowledge model itself and its application in evolutionary software development.

6.1.2 Related Works

As an extension to relational, semantic, and object-oriented data models, knowledge rules have been incorporated into many research works in next-generation database systems such as HiPAC [92], ODE [7], OSAM* [564], Postgres [377], and Starburst [378]. However, these models do not provide facilities for explicitly modeling method implementations.

Object-Oriented data model provides a uniform framework by encapsulating both the structural properties and part of the behavioral properties (in terms of signature specifications of methods) of a target system into object classes. Nevertheless, the implementation part of each method is still left as a blackbox and cannot be further modeled. Because the specification of methods does not carry enough behavioral information, the implementation is often prone to errors. Several research works have been done in an effort to provide an integrated diagram notation for static and dynamic aspects of software systems. Both Kung [338] and Markowitz [389] tried to combine ER data model and data flow oriented process specification as a single graphic design tool for conceptual modeling. However, they do not explicitly model process implementations and therefore cannot support evolutionary prototyping of software systems throughout their entire lifecycles. Besides, as behavior properties (processes) are not incorporated into an object-oriented framework, they cannot take advantage of object-oriented paradigm such as inheritance and object-oriented database system support.

Brodie and Ridjanovic [80] proposed ACM/PCM (Active and Passive Component

Modeling) methodology for structural and behavioral modeling of database applications using an integrated object/behavior schema. Three types of control abstractions (sequence/parallel, choice, and repetition) are used to represent the behavioral relationships between an operation and its constituent operations. Since behavioral properties are explicitly modeled only at a gross level of detail by relating operations to form high-level, composite operations, there is not enough information for the behavioral schema to be executable and evolve into the target system at the implementation level.

Kappel and Schrefl [307] proposed object/behavior diagrams as a uniform graphic representation of object structure and behavior based on a semantic data model and petri-nets. Behavior diagrams are split into (i) life-cycle diagrams which identity possible update operations and their possible execution sequences with synchronization constraints, (ii) activity specification diagrams which represent method specifications, and (iii) activity realization diagrams which represent method implementations at any level of details. Though closely related to our work, object/behavioral diagram is more of a graphic design tool than a formal knowledge model. Because there is no kernel model to model object/behavior diagrams themselves, software systems represented by these diagrams cannot be uniformly modeled and managed by some underlying KBMS. For example, a user will not be able to inquire about the structural and behavioral properties of objects.

The rest of this chapter is organized as follows. In Section 6.2, we give an overview of the knowledge model. Structural and behavioral abstraction mechanisms of the knowledge model are described in Section 6.3 and 6.4, respectively. Section 6.5 summarizes this research work and gives our conclusions.

6.2 Knowledge Model Overview

6.2.1 Classes

We use *classes* as the knowledge definition facilities to classify objects by their common structural and behavioral properties in an integrated fashion. Classes are categorized as entity classes (E_Class) and domain classes (D_Class). The sole function of a domain class (e.g., integer, real, and string) is to form a domain of possible values from which descriptive attributes of objects draw their values. An entity class, on the other hand, forms a domain of objects which occur in an application's world and can be physical entities, abstract things, functions, events, processes, and relationships. The structural properties of each object class (called the *defining class*) and thus its instances are uniformly defined in terms of its structural associations (e.g., aggregation and generalization [527]) with other object classes (called the *constituent classes*). Each type of structural association represents a set of rules that govern the knowledge base manipulation operations on the instances of those classes that are defined by the association types. Functional associations between object classes can also be specified by such association types as "friend" [561] and "using" [71] to facilitate programming in the large as will be described in Section 6.3.1. Manipulation of the structural properties of an object instance is done through methods, and the execution of methods is automatically governed by rules to maintain the system in a consistent state or to trigger some pre-defined actions when certain conditions become true. In other words, the behavioral properties of each object class are defined as methods and rules applicable to the instances of this class. The procedural information (algorithm) of methods can be explicitly modeled using control associations as will be described in Section 6.4.1.

Since rules applicable to the instances of a class are defined with the class, rules relevant to these instances are naturally distributed and available for use when instances are processed. Structural associations, functional associations, and control associations are all called "class associations" as each of them specify an association between the defining class and the constituent classes. A schema is defined as a set of class associations, which corresponds to a real world application. A sample entity class definition of Student is given in Figure 6.3 to illustrate the skeleton of a class definition. A detailed description will be given in the latter sections.

6.2.2 Objects and Instances

Objects are categorized as domain class objects (D_Class_Object) and entity class objects (E_Class_Object). Domain class objects are self-named objects which are referred to by their values. Entity class objects are system-named objects each of which is given a unique object identifier (oid). We adopt a distributed view of objects to support generalization and inheritance as in [345, 634] by visualizing an instance of class 'X' as the representation (or view) of some object in class 'X'. Each object can be instantiated (as an instance) in different classes with different representations but with the same oid. Each instance is identified by a unique instance identifier (iid) which is the concatenation of cid and oid, where cid is a unique number assigned for each class in the system. Each entity class object (and therefore all its corresponding instances in different classes with the same oid) can be either persistent or transient. After a user session ends, all the transient objects created in this session are deleted and all the persistent objects are stored back into the database. A detailed discussion of persistence can be found in [522]. Each entity class is associated with an extension which is the set of all its instances.

6.3 Structural Abstraction

Structural properties of objects are modeled by using various structural association types. In Section 6.3.1, we give a brief description of the kernel association types "aggregation" and "generalization". A three-level information hiding mechanism is described in Section 6.3.2. As each class can be thought of as a reusable software module in object-oriented software development, two types of functional associations are provided to facilitate programming in the large. The introduction of the "friend" and "using" associations also illustrates the extensibility of the knowledge model. In Section 6.3.3, We illustrate the model reflexivity and the structural schema notation. A brief description of structural association patterns is given in Section 6.3.4.

6.3.1 Structural Association Definitions

Aggregation. For each object class, one can define a set of attributes (which are also expressed as data members or instance variables in other object-oriented programming languages) to describe the state of its instances in terms of their associations with other classes by using the aggregation (A) association type. Each attribute specification corresponds to an instance of class "Aggregation" and also a named aggregation association (A-link) from the defining class to the constituent class. The name of an attribute must be unique within the defining class. An aggregation association defines either (i) a value attribute if its constituent class is a domain class, or (ii) a reference

attribute if its constituent class is an entity class. Multi-valued attributes are specified using the constructors "set", "list", and "array [size]". Note that an aggregation association between two entity classes is interpreted as a bi-directional link to support bi-directional navigation and to maintain the referential integrity of the knowledge base. For example, before deleting a Department instance, the system can follow the "major" link of Student in the reverse direction to identify those students who major in this department and remove their "major" links to this particular Department instance.

Generalization. For each object class, one can use generalization (G) association to specify its immediate superclass or subclass. Class 'B' is said to be a subclass or specialization of class 'A' (i.e., there is a generalization association from 'A' to 'B') if for each object which has an instance in class 'B', it also has an instance in class 'A'. Both instances have the same oid and are conceptually connected by a generalization association (G-link).

6.3.2 Encapsulation and Inheritance

We adopt the C++ three-level information hiding mechanism [561] by classifying aggregation associations and methods as either "public", "private", or "protected". Note that all the rules are treated as "protected" by definition. At the class level, all the (i) public/protected aggregation associations and other types of associations, (ii) public/protected methods, and (iii) rules defined by a class are inherited by its subclasses. At the instance level, an instance of class 'A' stores only the attributes defined for 'A', and it inherits all the public/protected attributes from its corresponding instances (with the same oid) of all the superclasses of 'A'. Name conflict in multiple inheritance is resolved by requiring the user to explicitly specify from which superclass a particular property is inherited.

Friend: This association type is used to support the three-level information hiding mechanism described above. A "friend" (F) association specifies that all the constituent classes are "friends" of the defining class and thus authorizes them to access the private and protected properties of the defining class.

Using: Similar to the "#include" macro in C++, a "using" (U) association specifies that all the public interfaces defined by the constituent classes will be available to the defining class (client- server relationship). Note that though this information has been implicitly captured in parameter specifications and method invocations, we include it at the class level for better readability and maintainability of complex software systems. For example, a user can easily capture the overall structural and functional relationships among system modules by just reading the association definition or graphic display of the system schema rather than going into the detailed codes of each method. Besides, the compiler can make use of the semantic information provided by the "using" associations in a system schema to automatically include all the necessary classes for compilation. Note that the "using" association provides a modular mechanism at a larger granularity than ordinary classes as one can either (i) functionally compose many classes into a big module structure or (ii) functionally decompose a big module into smaller modules.

6.3.3 Extensible Kernel Model

Model extensibility is achieved via a reflexive kernel model shown in Figure 6.5 in which all the data model constructs described above such as classes, associations, methods,

and rules are modeled as first- class objects. One can extend the data model by modifying this set of meta classes. This kernel model also serves as the data dictionary as all the object classes in the system are mapped into this class structure. One can therefore browse and query any user-defined schema as well as the dictionary uniformly. Note that Figure 6.4 illustrates the overall generalization lattice, and Figure 6.5 shows the detailed structural relationships among those kernel object classes as we will describe in the following sections. In our structural schema notation, (i) entity classes and domain classes are represented as rectangular nodes and circular nodes, respectively, (ii) a generalization association is represented by a "G" link from a superclass to a subclass, and (iii) an aggregation association is represented by an "A" link from the defining class to a constituent class. Note that the root class "Object" is represented by a special notation because it is neither an entity class nor a domain class. The sole function of class "Object" is to serve as the collection of all the objects in the system. After compilation, any user-defined class (e.g., "Person" and "Student" in Figure 6.4 will be added to the class structure as an immediate or non-immediate subclass of either "E_Class_Object" or "D_Class_Object", while at the same time the objects corresponding to the class definition, associations, methods, and rules of the defining class will be created as instances of the system-defined entity classes named "Class", "Association", "Method", and "Rule", respectively. Note that this class structure is reflexive in the sense that we use the model to model itself. For example, while any user-defined or system-defined entity class is a subclass of "E_Class_Object", "E_Class_Object" itself is also an entity class (represented by a rectangular node). Similarly, "D_Class_Object" itself is also a domain class.

As any application domain (including the model itself) is uniformly modeled and mapped into the kernel model, the class structure can be further extended at any level of abstraction. For example, one can use the kernel model to incrementally extend the model itself by either (i) adding new structural association types or introducing subtypes of existing association types (e.g., "Interaction", "Composition", and "Crossproduct" [564]) by specifying their structural properties (in terms of existing structural association types) and behavioral properties (in terms of rules which govern the knowledge base manipulation operations on the instances of those classes defined by the association types) or (ii) extending the definition of existing association types (e.g., add new attributes "default_value", "null_value", "optional", "unchangeable", and "dependent" [523], as well as their corresponding rules for association type "Aggregation") so that more semantics can be captured in the schema and maintained by the KBMS instead of being buried in application codes. Once a new association type is defined, it becomes a semantic construct of the extended data model and can be used in the definition of any object classes (including any other new association type). In such a way, the data model itself can be incrementally extended to meet the requirements of various application domains.

6.3.4 Structural Association Patterns

Since the development and execution of all software systems using the evolutionary prototyping approach are supported by a KBMS, any program execution would generally involve the processing of a persistent knowledge base. For knowledge base retrieval and manipulation, a knowledge base programming language should include some knowledge manipulation constructs in addition to general programming constructs. In our work on K [522, 523], we use pattern-based querying constructs for this purpose. We modify the context expression of OQL [11, 233] as the primitive construct for specifying struc-

tural association patterns based on which the system can identify the corresponding contexts (sub-knowledge-bases) that satisfy the intentional patterns. In general, each association pattern can be defined as

```
class_1 '[' <intra-class selection condition> ']'
        <op> <direction> '['<association-name>']'
class_2 '['<intra-class selection condition>']'
        <op> <direction> '['<association-name>']' ...
```

where <op> could be either an "associate" ("*") or a "non-associate" ("!") operator and <direction> can be either ">" or "<" so that the defining class of <association-name> is always at the open side, i.e., the left-hand-side of ">" or the right-hand-side of "<". One can also explicitly specify a range variable over a class in the association pattern as "<var>:<class>". For example, "g:Grad[major.name=''CIS''] *>[advisor] p:Professor !<[instructor] Course" specifies a sub-knowledge-base that contains all the graduate students of CIS department who has an advisor (i.e., there is an "advisor" link connecting this student with a professor) who does not teach any course (i.e., this professor is not connected through the "instructor" association with any course instance), as well as their advisors and those courses which these advisors do not teach. Here, g and p are variables that represent the graduate students and professors satisfying the association pattern specification, respectively. Instead of using a class notation, one can also directly designate objects by replacing class name with any user- defined variable. For example, "this *>[advisor] Professor" specifies a context which consists of the particular student denoted by "this" and his/her advisor. A context can be thought of as a normalized relation whose columns are defined over the participating classes and each of its tuples represents an extensional pattern of iids that satisfy the intentional pattern. A detailed description of more complex association patterns and the use of association patterns to express universal and existential quantifiers can be found in [522].

After a context is identified, one can use the context looping statement provided by the system to manipulate objects over each extensional pattern. One can use range variables to implicitly project over only those columns which he/she is interested and eliminate the resulting redundant tuples. For example, the following statement will print the name of each student who takes any CIS course. Note that we use range variable s to do a projection over Student column and remove the redundant tuples so that each qualified student will appear only once even if this student takes more than one CIS courses.

```
context s:Student *>[enroll] Course[offered\_by.name=''CIS's]
  do s.name.display();
end\_context;
```

6.4 Behavioral Abstraction

Behavioral properties of objects are modeled by methods and rules. In the traditional object-oriented programming, a method consists of a signature which specifies the name of method, parameters, and the data type of a returned value (if a value is to be returned) and the actual program codes that implement the method. However, in the prototyping of a complex system, the prototyper may want to avoid the actual coding of a method at a particular point in time and use instead some simpler table

lookup codes to simulate the function of the method (i.e., given some legitimate input data, produce some legitimate output value by a table lookup). Or the prototyper may feel that the method is still too complicated to code directly and wants to decompose its implementation into program segments interconnected by a control structure. In this case, the method implementation can be represented by a control structure of its program segments which are modeled as object classes with their own methods to define their functionalities. In other words, each program segment (whose size could vary from thousands of statements to a single statement) can be modeled as an object class along with a method (the default method name is "main") to represent the functionality of this segment. To activate a program segment, one just sends a message to an instance of the corresponding object class to invoke the proper method. Through this decomposition process and, at each step, each method associated with an object class is either represented by an actual or simulated program, or by a control structure of program segments that model the method. Each control structure can be used by the prototyping system for automatically generating the corresponding codes. As each method is executable, the model of the entire software system is executable and can gradually evolve into the target system by modifying and refining the executable model. Thus, procedural abstraction and functional decomposition are also incorporated into the proposed object-oriented framework. For the above reason, the meta model of the "Method" class shown in Figure 6.5 consists of an execution mode and a signature of its method name, parameter declarations, and the return type. Based on the execution mode which is either "model" or "operational", the system can choose one of the following to execute: (i) a method_model object which is the prototype model (schema) of a method implementation, and (ii) a piece of simulated codes or actual implementation of the method in some programming language. In Sections 6.4.1 and 6.4.2, we describe the modeling of method implementation in terms of control associations and method_model objects, respectively. A description of rules is given in Section 6.4.3.

6.4.1 Method Model and Control Associations

It is shown in [137] that three forms of control structures (sequence/parallel, choice, and repetition) can be used to define all partial recursive (i.e., computable) functions. As mentioned in Section 6.3.3, one of the advantages of the extensible kernel model is that we can extend the model itself by introducing new association types to carry whatever information we need in association links. In order to explicitly model method implementations in an object-oriented framework, we define a class called "Control" as a subclass of "Association" to model the control relationships among program segments that implement the method. Control associations are categorized as "Sequential" (S), "Parallel" (P) , "Synchronization" (Y), "Testing" (T), and "Context_Looping" (L) as shown in Figure 6.4. A method model is defined as a class schema in which one uses object classes to model program segments and control associations among these object classes to model the control structure of these program segments that implement a particular method.

Figure 6.6 represents some program segments with basic control constructs using control associations. Each rectangular node shown in Figure 6.6 is an entity class which models a program segment that constitutes a method implementation, and each control association in our model represents a possible control flow in terms of message passing between these object classes. A Context-Looping association is used to model the context looping statement in which the system (i) first establishes a relation representing a sub-knowledge-base satisfying the intentional association pattern which is modeled by

the first class in Figure 6.6(8), and (ii) performs certain operation for each tuple of the relation as described in Section 6.3.4. To sum up, each program segment in a method model can be described by a triplet (C, M, P) where 'C' is an object class, 'M' is the "main" method of 'C' that performs the functionality of the program segment being modeled, and 'P' is a message passed to a specific instance of 'C' to invoke method 'M'. In other words, 'C' and 'M' can be thought of as the object-oriented procedural abstraction of the program segment being modeled, and 'P' represents the activation of this particular program segment. Through an iterative process, any complex software system can be modeled to any level of details at which point the prototyper can begin to write actual codes in the target language.

The advantages of using method models are four-fold. First, instead of visualizing each method as a black box, a method model provides a graphic representation of method implementation to capture the behavior properties of a method. Secondly, the method associated with a class that models a program segment can be further modeled by another method model. The process can be repeated to any level of abstraction as desired by the prototyper, and the lowest level of abstraction is each individual program statement. Thirdly, a KBMS can use method models for an automatic generation of codes in the target language where each program segment modeled by (C,M,P) will be replaced by the actual codes of 'M' or the actual codes recursively generated from the method model of 'M'. The resulting codes can then be compiled by the compiler of the target language for execution. Fourthly, a KBMS can directly execute a method model by using an interpreter to dynamically activate each program segment in a control structure following the control association links. Since all the structural and behavioral information needed for execution are stored in the control association links, the execution of a method model can be thought of as the processing of the set of control association links which constitute the method model.

Structurally, each control association link can carry different behavior information as defined by the following attributes where (1) "context_branch" and "sub_kb" are defined by "Context_Looping", and (2) "test_branch" is defined by "Testing" as shown in Figure 6.5.

(1) context_branch and sub_kb: a Context_Looping association can be specified by a context_branch attribute whose value could be either "next" or "exit" to represent the iteration or exit of the looping, respectively). Note that the defining class of a Context_Looping association corresponds to the program segment which, when activated, will generate a relation representing the sub-knowledge-base satisfying an intentional association pattern. During the execution of a Context_Looping association, the system will also keep a pointer to the relation (the value of "sub_kb") over which the context looping is performed.

(2) test_branch: a Testing association can be specified by a test_branch attribute whose value could be either "true", "false" (for modeling the "if-then-else" statement), "otherwise", or any other value (for modeling the "case" statement) as shown in Figure 6.6. The defining class of a Testing association corresponds to the program segment which can be activated to generate the proper value of "test_branch" based on which the system can choose one of the possible control flows to follow during the execution time.

The behavioral properties of each control association type are described by the following algorithm of execution. We assume that for each process (in the case of concurrent system) created by a user session, there is a "wait_set" for recording those control association links which are waiting for synchronization. We also assume that each entity class which models a program segment defines a method called "main" to

represent the functionality of this program segment. To activate a program segment modeled by class 'X', we create a transient instance of class 'X' and apply the "main" method of class 'X' to this instance.

Case 1. There is a Sequential (S) association link L1 between class 'A' and class 'B'. We activate the program segment modeled by class 'A'. Then, if there is no control association link starting from class 'B', then we activate class 'B' and terminate. Otherwise, we continue to process the next control association link(s) from class 'B' (i.e., those control association objects whose defining class is class 'B').

Case 2. There is a list of Parallel (P) association links between class 'A' and class 'B1', 'B2',..., and 'Bn'. We first activate the program segment modeled by class 'A'. Then, we fork n new processes in parallel, one for each class 'Bi'. For each class 'Bi', if there is no control association link starting from 'Bi', then we activate 'Bi' and terminate the process. Otherwise, we continue to process the control association link(s) starting from class 'Bi'.

Case 3. There is a Synchronization (Y) association link between class 'B' and class 'A1'. There is also a set of Synchronization association links from class B to classes 'A2', 'A3',...,'An'. Let L1, L2,...,Ln represent these Synchronization association links, respectively. We first activate the program segment modeled by class 'A'. Then, if "wait_set" already contains L2 to Ln, then the synchronization condition is met and we do the following: (1) remove L2 to Ln from the "wait_set", (2) if there is no control association link starting from class 'B', then activate the program segment modeled by class 'B'; otherwise, continue to process the control association link(s) starting from class 'B'. Otherwise ("wait_set" does not contain all L2 to Ln), we terminate the process which currently executes L1, and add L1 into the "wait_set".

Case 4. There is a list of Testing (T) association links between class 'A' and class 'B1', 'B2',..., and 'Bn'. We first activate the program segment modeled by class 'A' and, based on the returned value, the system will choose one Testing association link whose "test_branch" attribute value is equal to either (i) the returned value, or (ii) "otherwise" if none of the test_branch values matches the returned value. Assume this chosen association link is defined from class 'A' to class 'Bi'. If there is no control association link starting from class 'Bi', then we activate the program segment modeled by class 'Bi' and terminate. Otherwise, we continue to process the next control association link(s) starting from class 'Bi'.

Case 5. There are two Context_Looping (L) association links between class 'A' and class 'B1' and 'B2'. Let L1 represent the association link whose "context_branch" attribute value is "next" (and assume which is defined from 'A' to 'B1'), and L2 represent the association link whose "context_branch" attribute value is "exit" (and which is defined from 'A' to 'B2'). If "L1.sub_kb" is null, then we activate the program segment modeled by class 'A' and return a pointer to a relation representing the sub-knowledge-base over which the looping will be performed. The "sub_kb" attribute of L1 will be set to this pointer. If "L1.sub_kb" points to an empty relation or all the tuples have been processed, then we do the following: (1) delete the relation, (2) set "L1.sub_kb" to null, (3) if there is no control association link starting from 'B2', we activate the program segment modeled by 'B2' and terminate; otherwise, we process the next control association link(s) starting from 'B2'. Otherwise (i.e., there are more tuples to be processed), we get the next tuple and continue to process the next control association link(s) starting from 'B1'.

6.4.2 Method_model Object and Evolutionary Prototyping

As shown in Figure 6.4, "Method_model" is a subclass of the class "Schema". Each method_model object represents the executable model of a method and is described by (i) a set of class association objects (inherited from class "Schema"), (ii) a starting point which is a control association object in (i), and (iii) a set of local variable declarations. Note that in order to unambiguously preserve the semantics of the order of execution (control flow) when a method model is mapped into a set of association objects, each class which appears in more than one places in the method model must be recorded by using alias names internally. An alias name is formed by appending an underscore and an integer to the class name, e.g., Sort_1 and Sort_2. Note that without using alias names, the system will not be able to restore the model correctly. For example, the control structure restored from three consecutive "Sequential" associations A-B, B-C, and C-A will form an infinite loop instead of a sequence if no distinction is made between these two appearances of class 'A'. Besides, each method_model object must know which association object is the "starting point" of execution. From the starting point, the method model can be restored and processed by following the control associations.

In the following, we illustrate the concept of evolutionary prototyping and the use of all types of control associations by developing "eval_GPA()" which is a method of Student as shown in Figure 2.1. Note that although "eval_GPA()" is a rather simple method which normally could have been directly coded, the technique illustrated by this example can be applied to the modeling of complex methods of a large software system to any level of details. Assume we have defined Transcript as an entity class whose each instance represents the grade point of a particular student for a particular course, and we need a program to compute the GPA of a given student. For this example, it is obvious to model this program as a method of class Student with the signature "eval_GPA() : GPA_Value". In the beginning, one might just write a simple piece of simulated codes to generate some legitimate GPA_Value from some given legitimate student instance as the receiver of this method by either performing some table lookup or inquiring the user interactively so that this method can be executable (in operational mode).

Later on, one may decide to model the detail of this method by decomposing its functionality into five consecutive program segments: (1) compute the total grade points of this student and assign this value to a local variable, (2) compute the total credit hours of this student and assign this value to a local variable, (3) get the GPA by dividing results from (1) and (2), (4) print a message if the GPA is below 2.0, and (5) return the GPA. Each program segment can be modeled as an entity class with a "main" method to represent its functionality, and each "main" method can be given simulated or actual codes or recursively modeled as described in Section 6.4.1. Note that for this example, both segment (1) and segment (2) can be further decomposed as Context_Looping control structures, and the computations can be performed over the same context concurrently. Therefore, we combine them together to illustrate the Context_Looping, Parallel, and Synchronization associations. A method model at a particular stage of decomposition is shown in Figure 6.7. We first declare local variables s1, s2, and GPA to hold the accumulated grade points, accumulated credit hours, and GPA value, respectively. The receiver of this method is denoted by the pseudo variable "this". We first use a Context_Looping association to iterate over the context specified by "`this *<[student] t:Transcript *>[course] c:Course`" to

evaluate the accumulated grade points and credit hours of this student in parallel. Note that the updates of s1 and s2 are performed in parallel and must be synchronized before the execution can be continued. After the looping is finished, we get the GPA value by dividing s1 by s2. A message is printed if this student has a GPA lower than 2.0. Finally, we return the GPA value. Note that in some cases it is necessary to introduce entity classes which model null program segments in a control structure. For example, the classes "null_1" and "null_2" in Figure 6.7 model the null program segments which serve as the "fork" and "join" points of control flows, respectively. This example shows that it is possible to model a method recursively to such a detailed level that each program segment contains only a single statement. Naturally, the segment size (i.e., the level of detail) in a model will be determined by the prototyper. By using a graphic user interface as part of a prototyping environment, a prototyper can click the mouse button to select any class in a method model and view the program segment it represents as shown in Figure 6.7. A control structure of the kind shown in Figure 6.7 can be used by a KBMS to dynamically execute a method model or automatically generate the proper executable code that implements the method as shown in Figure 6.3.

6.4.3 Rule Definition

Rules serve as a high-level mechanism for specifying declarative knowledge that governs the behavior of methods. We modified and extended the rule language of [12, 566] so that it can be seamlessly incorporated into the knowledge base programming language K [522]. Each rule is specified by a set of trigger conditions and a rule body. Each trigger condition consists of two parts: (1) timing specification or coupling mode, which can be either "`before`", "`after`", "`immediate_after`", or "`in_parallel`", and (2) event specification, which can be a KBMS operation or user-defined method. The rule body consists of (i) "condition" clause which is a guard expression, and (ii) "action", and "otherwise" clauses, both of which can be simple or compound statement. Each guard expression is in the form "`(guard1, guard2,...,guardN | target)`" and the evaluation of a guard expression can return either (i) true: if all the guards and the target (all of which are boolean expressions by themselves) are true, (ii) skip: if any of the guards is false when they are evaluated from left to right, (iii) false: if all the guards are true but the target is false. All the rules are assumed to be active when a user session begins. However, during the execution of a user program, one can explicitly activate or deactivate any particular rule by sending the "activate()" or "deactivate()" messages to a specific rule object, respectively.

Each active rule of class X will be checked (i.e., the evaluation of the rule body) according to the coupling mode at either (i) before the triggering event, (ii) immediately after the triggering event, (iii) in parallel with the triggering event, or (iv) not immediately after the triggering event, but at the end of the parent event which causes the triggering event. The rule body of each rule is evaluated as follows: (i) if the condition-clause returns true, then the action-clause (if provided) is executed, (ii) if the condition-clause returns skip, then do- nothing, and (iii) if the condition-clause returns false, then the otherwise-clause (if provided) is executed. For example, the rule CIS_rule1 specified in Figure 6.3 will be executed at the end of those methods which are applied to a student instance and update the major of this particular student. The otherwise-clause will be executed if this particular student is a CIS major (guard is true) and his/her GPA is not greater than 3.0 (target condition is false). Similarly, General_rule1 will be checked after the method "suspend".

6.5 Conclusions

In this chapter, we take a knowledge base modeling approach to evolutionary prototyping of software systems by introducing a unified and reflexive object-oriented knowledge model as an extensible framework for (i) capturing both the structural and behavioral properties of any target system at any level of details, and (ii) reflexively modeling the knowledge model itself as a kernel model so that all the meta information (structural and behavioral properties of objects) can also be modeled as object classes. Five types of control associations (sequential, parallel, synchronization, testing, and context_looping) are introduced for explicitly modeling the behavior properties of methods in terms of control flow and message passing relationships between object classes. We have developed a prototype of the knowledge base programming language K [522, 523] on top of ONTOS 2.1 [278] as the first step toward a full-fledged KBMS-supported software development environment for supporting evolutionary prototyping. We are currently extending the graphic user interface of OSAM*.KBMS [346] to support the definition and processing of control associations. Any user-defined structural/behavior schema will be translated into K code for execution and also mapped into the kernel model for storage. A prototype processor which can dynamically interpret K code and control association objects is also under development at the Database Systems Research and Development Center of the University of Florida.

Acknowledgement - This research is supported by National Science Foundation under grant #CCR-9200756. The development of an executable KBMS and the translator of K is supported by the Florida High Technology and Industry Council under grant #UPN900978.

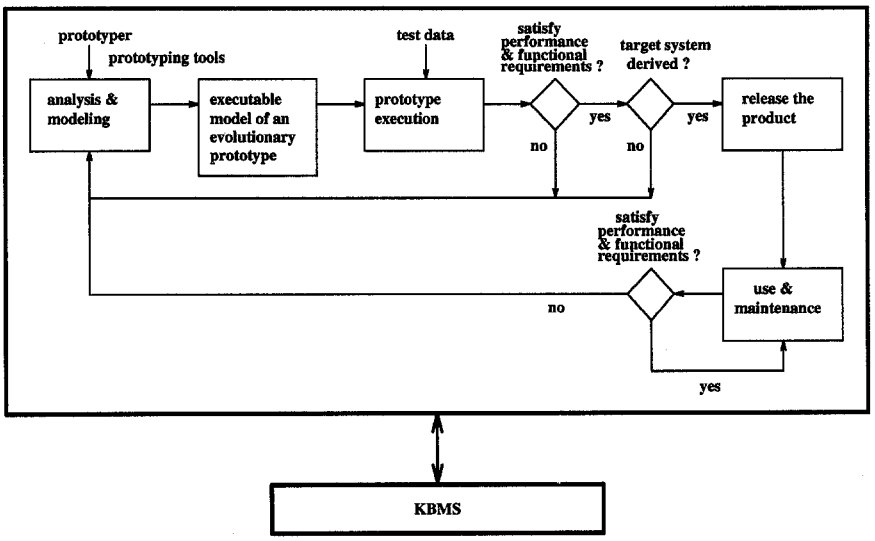


Figure 6.1: An Overview of a KBMS-supported Evolutionary Prototyping Process

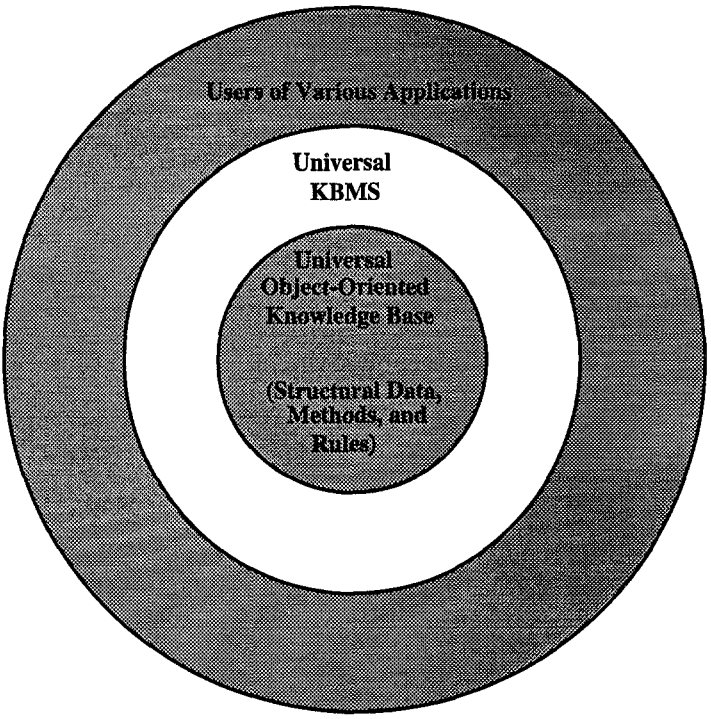


Figure 6.2: A Universal KBMS-Supported Software Development System

```
entity_class Student is
associations:
specialization of Person;/* Student is a subclass of Person */
friend of Faculty; /* authorize Faculty to access the private and
protected properties */
aggregation of
public:  /* definition of public attributes */
enroll:  set of Course; /* a student can enroll in a set of courses
*/
college_report:  array [4] of GPA_Value; /* annual report of every
college year */
major:  Department;
protected:  /* definition of protected attributes */
S#:  S#_Value;
methods:  /* the signature of methods */
public:
method eval_GPA() :  GPA_Value;
private:
method suspend() :  void; /* no return value */
method inform_all_instructor() :  void;
rules:
rule CS_rule1 is
/* after updating the major of a student, if the new major is "CIS"
then the GPA of this student must be greater than 3.0,
otherwise we suspend this student */
triggered after update major
condition (this.major.name = "CIS" | this.eval_GPA() > 3.0) /*
guarded condition */
otherwise this.suspend()
end CS_rule1;
rule Student::General_rule1 is
/* after suspending a student, if this student enrolls in any
course,
then inform all the instructors of this student */
triggered after suspend()
condition this.enroll != NULL
action this.inform_all_instructor()
end General_rule1;
implementations:  /* actual coding of methods */
method eval_GPA() :  GPA_Value is
local s1, s2 :  real := 0; GPA : GPA_Value; /* local variable
declarations */
begin
context this *<[student] t:Transcript *>[course] c:Course
/* looping over a context */
```

```
do parbegin /* for each <this,t,c> tuple, do the following */
s1 := s1 + c.credits * t.grade_point;
/* calculate the accumulated grade points */
s2 := s2 + c.credits; /* calculate the accumulated credit hours */
parend;
end_context; /* end of context looping */
GPA := s1/s2;
if GPA < 2.0
then "GPA Below 2.0".display();
end_if;
return GPA;
end eval_GPA;
end Student;
```

Figure 6.3: The Class Definition of Entity Class Student in K

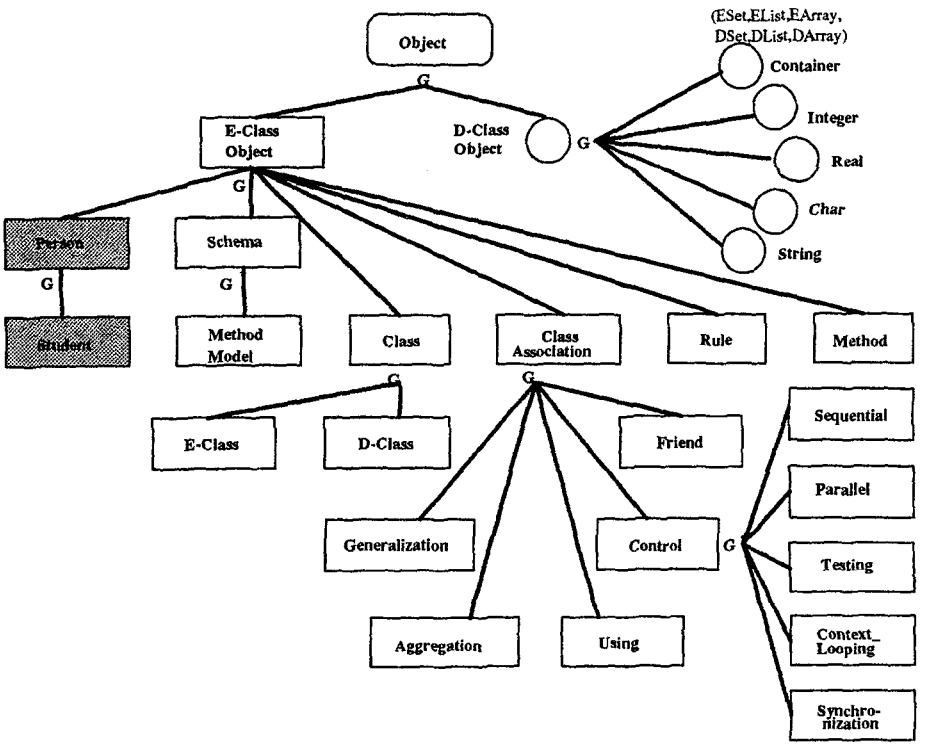


Figure 6.4: Class Generalization Lattice of the Extensible Kernel Model

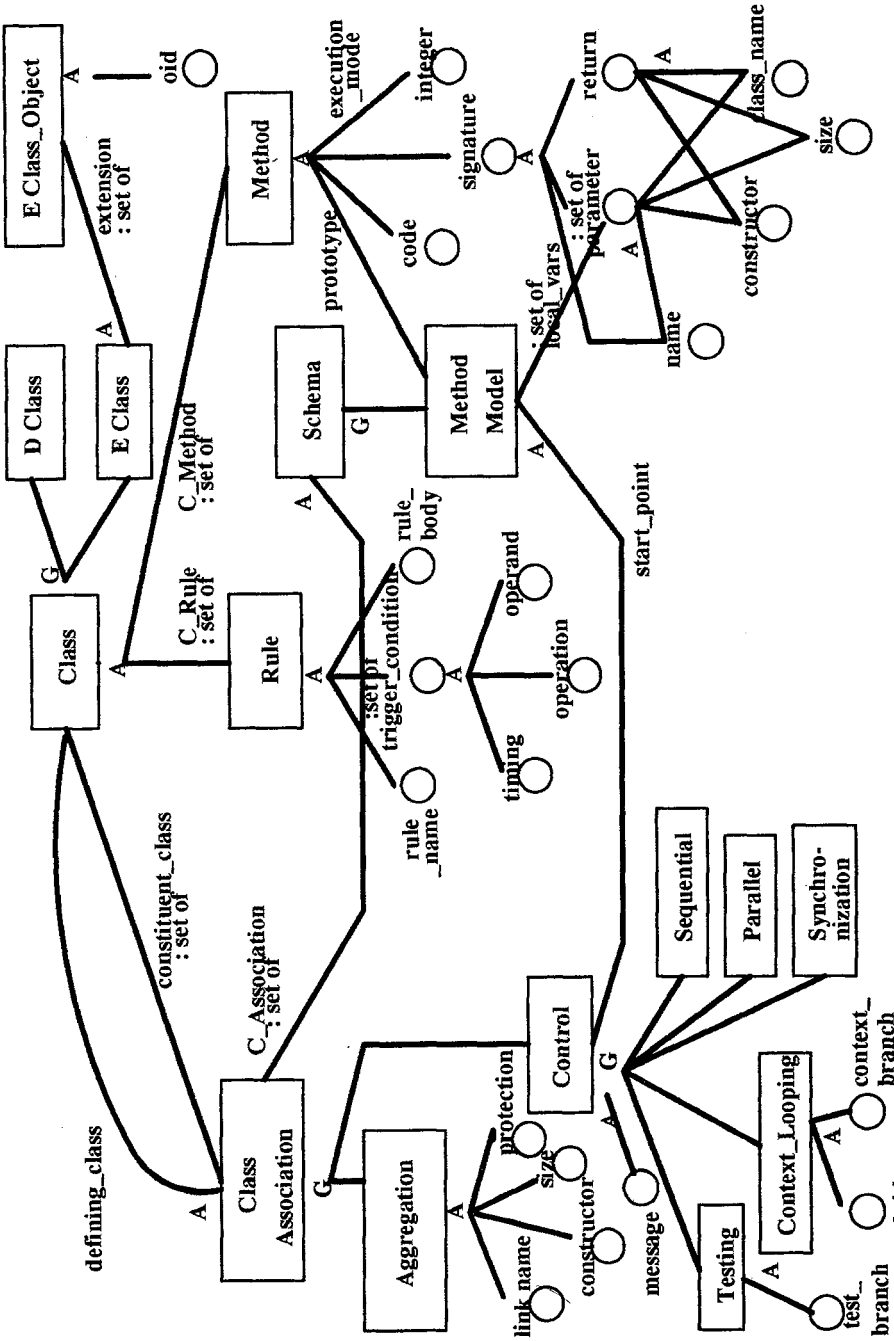


Figure 6.5: Aggregate of the Extensible Kernal Model

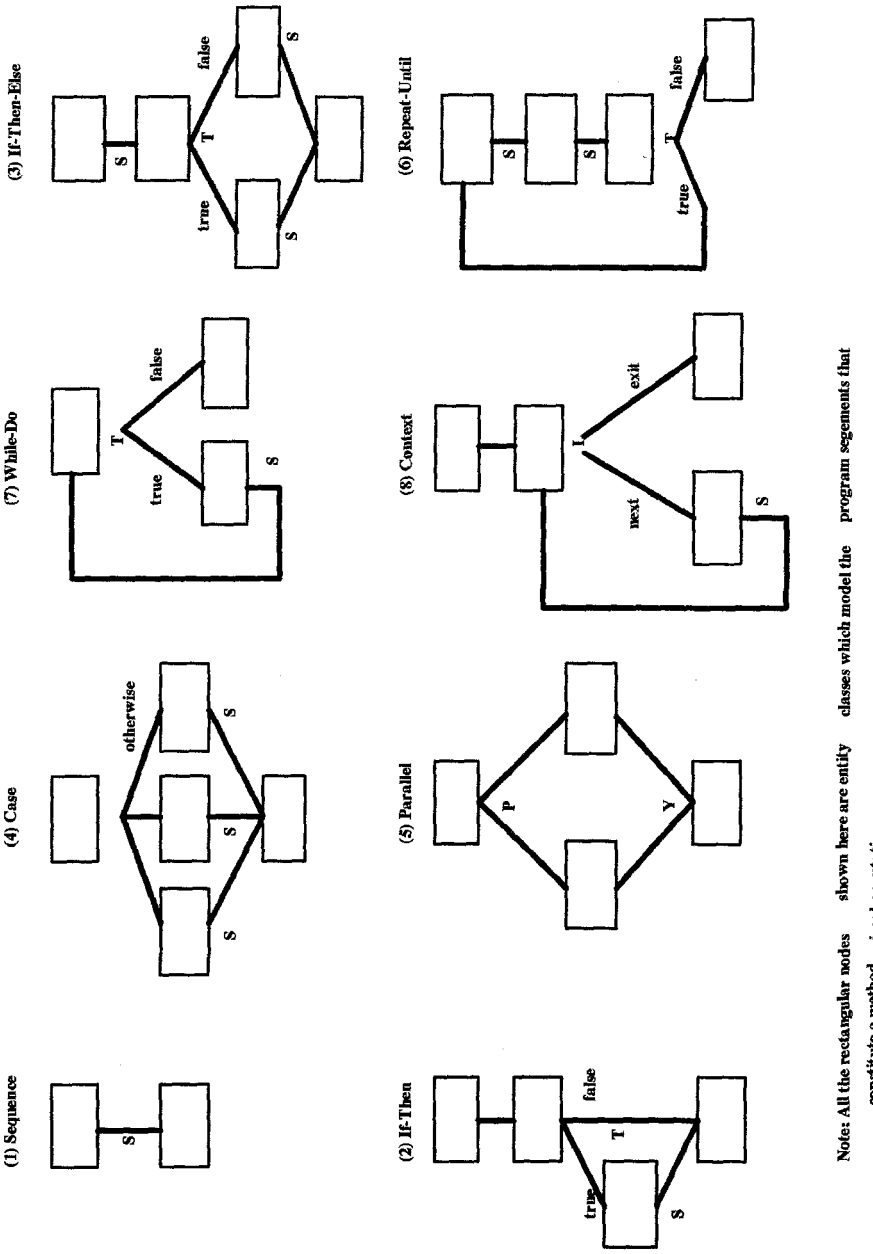


Figure 6.6: The Modeling of Basic Control Constructs

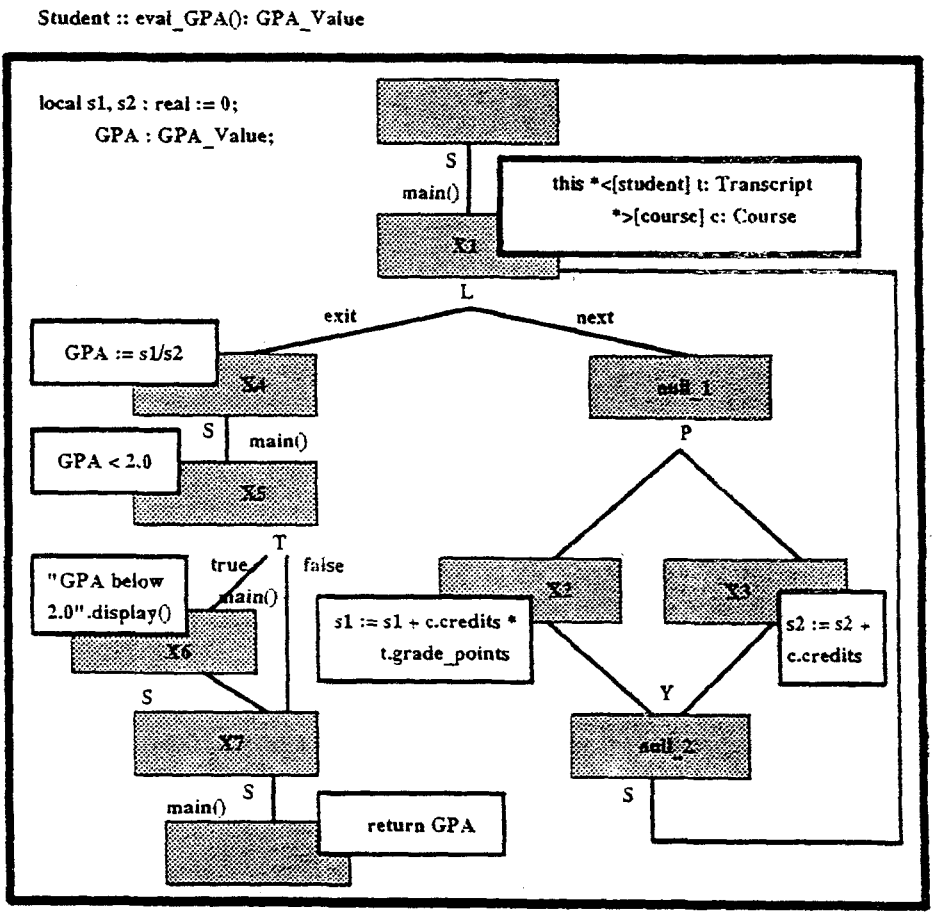


Figure 6.7: The Method Model of "eval-GPA()

Chapter 7

Applying OOAD in the Design and Implementation of an Intelligent Geographic Information System

Ramesh Subramanian*, Nabil R. Adam†

7.1 Introduction

Object-oriented Analysis and Design (OOAD) is currently an active area of research, and several OOAD techniques have been suggested for the Object-oriented design of computer-based applications (e.g. [124, 141, 147, 148, 184, 296, 356, 357, 358, 355, 359, 400, 405, 435, 468, 532, 615]). Notable among the suggested techniques are: the method of Coad and Yourdon [124], the approach by Meyer [400], the *Responsibility-Driven Approach* by Rebecca Wirfs-Brock from Tektronix [616, 617, 615], *Ensembles*, which is ongoing work at Hewlett Packard led by Dennis de Champeaux [141], the *Object-oriented Role Analysis, Synthesis and Structuring method* (OORASS) by Trygve Reenskaug at the Senter for Industriforskning in Oslo, Norway [468], *Frameworks* by Johnson at Urbana-Champaign [296] and the *Demeter method* developed by Karl Lieberherr at Northeastern University [356, 357, 358, 355, 359].

Recently, some studies that survey, compare and contrast the various OOAD approaches and techniques have appeared in the literature (e.g., see [617, 413, 187]). These studies seek to categorize the various approaches by identifying their *capabilities*, and their focus is towards identifying the particular effectiveness of one approach over the other. In one particular study, Wirfs-Brock and Johnson [617] view the above approaches as complementing each other. To our knowledge, there has been no study that tests the efficacy of the approaches by applying them to actual, complex and real-life modeling problems. This situation places the practitioners in a difficult position, since nobody wants to commit scarce time and resources towards adopting a particular approach that has not been adequately tested on industry-sized problems.

Adopting approaches that have been inadequately tested could also cause other problems, since in practice, the analysis and design stages have important ramifications on the implementation stage. For instance, (as explained in [248, page 147]), a "disjoint mapping" would result, if an OO design is implemented in a non-object-oriented language. One of the strengths of the OO approach is the use of the *same* language environment for the analysis, design, and implementation phases [26, 74]. That is, the analysis is done in terms of the *objects* that make up the actual system.

*CIOS Department, School of Business, University of Alaska-Anchorage, Anchorage, AK 99508, Email: afrs@acad2.alaska.edu

†MS/CIS Department, GSM, Rutgers University, Newark, NJ 07102, adam@adam.rutgers.edu

The results of the analysis are transformed into a software design that is expressed in terms of *objects*. Finally, the design is implemented also in terms of *objects*. This consistency throughout the analysis, design, and implementation phases avoids the potential *mismatch* that could result during the transition from one phase to the other. Maintaining this consistency also requires that the OOAD used to carry out first steps, namely analysis and design, be adequately tested for its real life application.

In this chapter we detail a modeling study that seeks to address the above issue. Namely, we detail a real-life, complex modeling problem, and illustrate, step by step, how to analyze the problem domain and design a model using one of the OOAD approaches, namely the *Responsibility-Driven Approach*. We consider this study to be a first step towards a practical analysis of the suitability of the various OOAD approaches for modeling specific problems domains.

The chapter is organized as follows: in Section 7.2 we detail the characteristics of the geographic database application that we model. In Sections 7.3 and 7.4 we discuss the methodology that we adopt to develop our model, and describe the Responsibility-Driven Approach. In Section 7.5 we detail, step by step, our approach to developing the geographic data model. In that section we also detail how our model acquires and handles knowledge about spatial contexts and user perspective. In Section 7.6 we briefly touch upon the implementation of our Model, and show how the various features of our Model and Query Processing System are used to process certain imprecise queries by using an actual query processing scenario. We present our conclusions in Section 11.6.

7.2 Geographic Databases: Modeling & Query Processing

There exists a large number of potential applications related to Geographic Information Systems (GISs). Examples include: applications concerned with storing and manipulating the characteristics of *political* spatial entities such as states, counties and towns; those that are concerned with storing and manipulating information pertaining to certain naturally occurring geographical features such as lakes, rivers, mountains and oceans; others that deal with the mapping of population distributions, land resource utilization, the spread of vegetation and certain *special districts* such as electoral districts, water pumping districts, and school districts.

Geographic data modeling requires the definition of high-level objects such as counties, towns, and districts from low level data elements such as points and lines [5]. Furthermore, spatial objects possess certain unique characteristics — for instance, even though the state of a spatial object may remain unchanged, its representation and relationships with other objects may be perceived differently in different *contexts* or *user perspectives*. Thus, if a *country level* perspective is assumed then the representation of the cities can be considered as *points*. If a *city level* perspective is assumed, then a city and its components can be considered as *polygons*. In the same fashion, at a particular perspective level, a bridge can be considered to be a *line*, but at a higher level of perspective, it could be assumed to have a *point* representation.

Thus, depending upon *user perspective*, the assumed *representation* of spatial objects could change. This can impact the results of a query such as *"What is the distance between X and Y?"* — depending upon the perspective of the user, the representation of the objects under consideration may change, thereby impacting the

results produced. This example shows that implementing even a *semantically well-defined* operator such as "distance" could become a complex exercise, if we try to incorporate the notion of user perspective and object representations in relation to the perspective. Examples of other semantically well-defined operators include "intersect" and "area". While several spatial database researchers have defined and implemented such *semantically well-defined* spatial operators, very few of the implementations incorporate the notion of *dynamically* deriving different user perspectives and object representations. For details of previous work in spatial databases see [223, 424, 101, 102, 36, 474, 249, 438, 237, 485, 486, 229].

In addition to semantically well-defined operators, we were interested in another class of operators that are actively being studied in the literature. These are the *ill-defined* or *relativistic* spatial operators. One prominent example of this is the study of the operator "near" by Robinson [472], in which he developed a C program that uses fuzzy logic and human machine interaction to derive an approximation of "near".

"Near" (or "close-to"), "between" and "adjacent-to" are examples of *semantically ill-defined* spatial operators. Such operators do not have a precise definition in the literature. Also, their *interpretations* change with user perspective. Thus, two towns, for example, may be considered to be close to each other at one perspective, but not so at another perspective. Similarly, two streets may be considered to be adjacent to each other even though there may be a building block between them.

From the above discussion, it can be seen that spatial (geographic) databases differ in character from conventional databases in the representation of *inter-object relationships*. In conventional database applications, relationships among data objects are *all* known *beforehand* and can therefore be represented explicitly in the database. In spatial databases, on the other hand, *numerous* implicit relationships may exist among spatial objects, and it is impractical to represent *all* of the relationships explicitly. Therefore, several such relationships need to be *derived* dynamically. This necessitates the development of new models and query processing strategies for efficient representation and manipulation of spatial data.

In our study we develop a model that supports the following:

- Defining high-level spatial objects from low-level data elements.
- Facilitating the explicit representation of certain naturally occurring relationships among spatial objects (e. g. the containment hierarchy that is exhibited by most spatial objects) and at the same time providing adequate facilities for deriving, on a dynamic basis, the many and complex relationships that exist among objects.
- Incorporating the notion of *user perspective* and different object *representations* in relation to perspective.
- Facilitating the implementation of a class of *semantically ill-defined* (hereafter will be referred to as imprecise) spatial operators such as "close-to", "between" and "adjacent-to". Here, the user must have the flexibility to specify his/her own operational parameters for processing the operators.
- Facilitating application flexibility i. e. , the ability to extend the model and its implementation to other similar applications.
- Encapsulating deductive reasoning that facilitates spatial query processing pertaining to the imprecise query operators.

We are interested in applying our model to such application areas as *districting* (electoral and water), and *public utilities maintenance management*. Districting is the

process by which a certain area of land is divided into several pieces, in order to satisfy certain requirements and needs. For example, an *electoral* district may be drawn based on the population size, composition of the population and other constraints such as *compactness*[‡] and *contiguity*[§] of the district. A pumping district may be allocated based on the current population and expected growth of the proposed district as well as the availability and ease of access to a water supply facility such as storage tank or pumping station.

A system such as the one developed in this study would provide the tools required for making the districting process easier. That is, the new geographic data modeling and query processing techniques would provide the means for "interactive districting". We are particularly interested in using our model and implementation for answering the following types of queries:

- Is polling-center Y *close-to* every sub-area of voting district X?
- What are the towns and counties which lie *between* districts X and Y?
- What are the counties *adjacent-to* pumping district X?
- What are the districts that are *close-to* the water source X?
- Retrieve all the towns whose populations (each) exceeds 75,000 and which lie *between* the pumping district X and town Y.

Given the above general requirements of our geographic application, we illustrate in the next few sections how we proceeded to design and implement the model.

7.3 Spatial Data Modeling

7.3.1 The Design Methodology

In [616], Wirfs-Brock classifies OO design methodologies into *DataDriven Approaches* and *Responsibility-Driven Approaches* and argues in favor of the latter approach. The latter approach intuitively seems to have certain merits over the former approach since it provides a convenient way of designing objects and the methods encapsulated within them. This is achieved by focusing on the objects' responsibilities rather than the detailed description of the objects, at the start of the design process.

Our application domain deals with queries about geographic data and their topological inter-relationships. Geographic data are complex in nature, and the possible inter-object relationships among them are *numerous*. In such a problem domain, the model design process could be made efficient by focusing on the *functions* that the various objects must perform (i.e., their *responsibilities*) rather than trying to define all the attributes of each individual object. This corresponds to the Responsibility-Driven Approach. We use the Responsibility-Driven Approach for designing our model. First, we provide below, brief presentation of the Responsibility-Driven Approach.

[‡]Compactness can be defined (on a scale from 0 to 1) as the ratio of the area of a district to the area of the smallest circumscribing circle [422]

[§]A district must be connected in some sense, so that it cannot be defined as an arbitrary collection of smaller areas [422].

7.4 The Responsibility-Driven Approach

The Responsibility-Driven Approach seeks to improve encapsulation by viewing objects in terms of the *client/server* model. According to this model, a client makes a request to a server, who provides the service requested. Both the client and the server can be classes or instances of classes. The exact way in which a client and a server can interact are described by a *contract*. A *contract* is a list of requests that a client can make to a server. The client is not concerned with the exact details of the actions taken by the server to provide the service requested. Thus, the focus in this approach is not on *how* a server performs an action requested from it, but *what* are the services that it can provide.

When applying this design approach, the designer is concerned more with the behavior and the *responsibilities* of the objects rather than their structural details. The structural specification of an object is focused on, only at the implementation stage. The responsibilities of an object are: *the knowledge that an object maintains*, and *the actions an object can perform*. The actions that are performed by an object are either performed individually, or by *collaborating* with other objects.

The Responsibility-Driven Design process is divided into two phases — *the Exploratory phase* and *the Analysis phase*. (This discussion is adapted from [617].)

The **exploratory** phase consists of the following tasks:

1. Identify Classes: The classes required to model the application are identified. Classes identified may be of two types: Abstract Classes that are designed to be inherited, and Concrete Classes that are designed to be instantiated. Abstract Classes are similar to type definitions.
2. Identify Responsibilities: Here the overall responsibilities of the system are identified, then the responsibilities of the individual classes are derived.
3. Identify Collaborations: If a class collaborates with another class in performing its responsibilities, such collaborations are identified during this task.

The **analysis** phase consists of the following tasks:

1. Analyze Hierarchies: The classes are organized in the form of a hierarchy. Such a hierarchy will be useful in finding classes that are reusable. An *is-a* hierarchy specializes the classes from top-down, and generalizes the classes from bottom-up. Wirfs-Brock and Johnson suggest that inheritance hierarchies must model the *is-kind-of* relationship. That is, "every class should be a specific kind of its superclasses" ([617, p110]). The *is-kind-of* relationship is similar to the *is-a* relationship.
2. Analyze Subsystems: A subsystem is a set of classes that fulfills some particular purpose in conjunction with other classes in the system. Thus, acting as a group, a subsystem of classes cooperate to fulfill a role. For instance, a graphical application may have a printing subsystem which may consist of a printer, which may be further specialized into line printer and laser printer [617, p111]. One way to determine if a group of classes form a subsystem is to try to name the group [617]. If we succeed in naming the group, then we automatically determine the *responsibility* or *role* of the subsystem.
3. Create Protocols: This final task in the design phase involves the implementation of the abstract classes and their behavior, by specifying actual protocols between the classes in the form of methods. Methods are used to describe contracts among

classes. A *base method* is one that is described in a superclass and is inherited down the hierarchy by the subclass. Another method which is inherited down the hierarchy, is known as *abstract method.* Such a method is *reimplemented* in the subclass in order to reflect a different (as compared to its superclass) and more specialized processing action between the subclass and its collaborator. In C^{++} such a method is also called a *virtual function.* A third type of method is called *template method* which serves the purpose of providing an abstract definition of an algorithm. Here the algorithm is made up of steps and each step is implemented as an abstract or base method. For more details see [617, p112]. A fourth type of method, which is not explicitly discussed in [617], is one where different responsibilities of the *same* class can be carried out, *depending upon* the particular parameters passed to it. For example, consider the transactions that have to be processed in a bank on a typical day. The transactions occur in serial order, and may be a *deposit, withdrawal,* or *transfer.* If the bank has a method *process,* then different actions could be defined within the method, depending upon the type of transaction processed. Such a type of method definition is termed *function overloading,* and is useful in describing a class of actions that can be performed by asking the same request, but with different parameters.

In the next section we detail the development of our Spatial Model, using the Responsibility-Driven Approach.

7.5 Developing the Data Model

7.5.1 The Exploratory Phase

In this phase, we identify the *main* classes that are required to model our geographic application, their *responsibilities* and their *collaborations.* We detail our work on the different tasks below.

Classes

We identify the following classes.

- **Spatial_Feature.** This is an abstract class that represents a template for a spatial feature. The responsibilities specified in this class are inherited by abstract as well as concrete subclasses that may be identified and added to the model at a later stage in the design process. Concrete subclasses that may be added through this process are instantiated with such high-level spatial features as "New York" state, or I-95.
- **Line_Segment.** This is a concrete class that represents a low-level spatial object. This class can be instantiated with the line segments from a geographic line segment file.

It should be noted that in addition to the above Classes, we identify more Classes at a later stage in the Exploratory phase. This is detailed in later sections.

Our next task in the design process is to identify the overall *responsibilities* of the system, as well as the responsibilities of individuals classes.

Responsibilities

- **The responsibilities of the system as a whole** are:

  1. Populate the geographic data model with data pertaining to the United States. This requires the system to have certain features whereby the raw data (line segments) are first loaded into the model. These would then be processed to build other high-level spatial features such as States and Counties.
  2. Derive dynamically the user perspective and the object representations of the spatial features in the model in relation to user perspective.
  3. Process queries pertaining to the spatial features in the model which involve imprecise operators such as "close-to", "between" and "adjacent-to".

Analysis of these responsibilities reveals that most queries pertaining to our application would be asked *with reference to* certain well understood spatial features in our model, such as Country, State, County, and Town. Examples of such queries include: Is County X *close_to* County Y?; What are the Counties *adjacent_to* State X?; What are the highways that run *between* Town X and Town Y?. Country, State, County, and Town are *container* objects that *contain* other *container* as well as *non-container* objects in the model. Highways and Rivers are examples of non-container objects.

We thus notice that the geographic features in our application can be broadly divided into *container* features which contain other features and *non-container* features which do not contain any other feature. Queries will be posed with reference to the *container* features. Due to this fact, we define two additional abstract classes, to represent the *container* features and the *non-container* features: **Reference_Spatial_Feature** and **Non_Reference_Spatial_Feature**, respectively.

We notice here that, **Reference_Spatial_Feature** and **Non_Reference_Spatial_Feature** are specializations of spatial features. Therefore, they are subclasses of Spatial_Feature, and inherit any responsibility that may be specified in the Spatial_Feature class. This leads to an inheritance hierarchy that is analyzed in detail in Section 7.5.2. As noted earlier, Reference_Spatial_Features are container features that are used as *reference features* in most spatial queries.

Reference_Spatial_Feature and **Non_Reference_Spatial_Feature** are Abstract Classes whose characteristics are inherited down the hierarchy by its subclasses. In order to add real life spatial objects such as States, Towns and Highways into the model, we define Concrete Subclasses of **Reference_Spatial_Feature** and **Non_Reference_Spatial_Feature**. Examples of the Concrete Subclasses of Reference_Spatial_Features are Country, State, County and Town. Interstate Highways, Railroads, Large Rivers, State Highways, County Roads, and Small Rivers are examples of the Concrete Subclasses of Non_Reference_Spatial_Features.

These spatial features are arranged in a *containment hierarchy*, with Country containing States, Interstate Highways and Large Rivers, etc., and States containing Counties, State Highways, Small Rivers, etc., and Counties containing Towns, County Roads, etc. More details of the containment hierarchy are discussed in Section 7.5.2.

At this stage in the design process we notice that even though the Reference Feature subclass such as Country *contain* Non Reference Feature subclasses such as Interstate Highways, Railroads and Rivers, some of these Non Reference Features can be divided into composite *segments*. The segments are *part-of* such Non Reference Features. Thus,

- Non References Features such as Interstate Highways and Rivers are *composed of*:
  - State-Segments which are contained entirely by States.
  - County-Segments which are contained entirely by Counties.
  - Town-Segments which are contained entirely by Towns.
- Non References Features such as State Highways *composed of*:
  - County-Segments which are contained entirely by Counties.
  - Town-Segments which are contained entirely by Towns.

This composite arrangement pertaining to the real world requires that we define additional Concrete subclasses of Non_Reference_Spatial_Feature. These are **State_I-Hwy_Segment**, **County_I-Hwy_Segment**, **Town_I-Hwy_Segment**, **State_River_Segment**, **County_River_Segment**, **Town_River_Segment**, etc. The spatial features pertaining to these subclasses are further subdivided into a composite arrangement, using the *part-of* relationship. The *part-of* hierarchy is given below.

Each State_I-Hwy_Segment is a *part-of* subclass Interstate Highway. Each County_I-Hwy_Segment is a *part-of* subclass State_I-Hwy_Segment. Each Town_I-Hwy_Segment is a *part-of* subclass County_I-Hwy_Segment. In the same way, each State_River_Segment is a *part-of* subclass River, and each County_River_Segment is a *part-of* subclass State_River_Segment and each Town_River_Segment is a *part-of* subclass County_River_Segment.

Other subclasses similar to the above, pertaining to the different levels of the containment hierarchy are also identified and represented.

- **The Rule Class: a result of identifying responsibilities of the system**

  Another result that is derived from identifying the responsibilities of the system is that, in order to process imprecise queries, it is necessary to incorporate into the system a body of *knowledge about contexts or user perspectives.* In addition, depending upon the problem domain, it may be required to store in the system other problem-specific knowledge. For example, in an *electoral districting* application, it may be required to store rules such as: All districts must have the same population, and must have the same racial mix; the component regions of an electoral district must be contiguous. We therefore, define another abstract class: **Rule.** In this class, we instantiate and maintain rules that are required for the problem domain in question. Examples of such rules are "ancestors-rule" (for reasoning about all the ancestors of a given object) and "perspective-rule" (for reasoning about the correct context of a given query). We elaborate more on these rules as well as other such rules later in this section, where we discuss the *responsibilities of individual classes.*

  The function and use of the **Rule** class in our model is worthy of further discussion. The incorporation of rules in a spatial data model to reason about the objects and their relationships within the context of a database is, to our knowledge, not available in currently existing spatial models.

Rules, in addition to *methods* offer a powerful construct for introducing *behavior* into the data model. In our model, the function of a rule is different from the function of a method. A *method* is a piece of procedural code that is embedded *within* an object in order to perform certain actions on the database. The method may be invoked by passing a message to the object. A *rule*, however, is created separately and is not attached to any particular spatial object in the database. Rules may be added or removed without interfering with any spatial object.

Rules play an important role in processing imprecise spatial operators. Rules for determining the *user's perspective* and the appropriate *object representation* in relation to the perspective while processing ill-defined spatial operators are incorporated into the model. Furthermore, the Rule class also provides support to incorporate specific constraints that must be satisfied in certain problem domains.

To illustrate, consider a districting problem, where it is required to interactively create districts from a geographic database of the United States. In addition, these districts must confirm to certain conditions. Assume that the geographic database consists of States, Counties and Towns. Districts are to be created by combining certain counties together. The database contains information pertaining to each county's population, number of schools available, number of fire stations available, number of community buses available, number of drivers available and number of physicians available.

Let us assume that the following are the conditions that must be met while creating districts:

- If the population of the district is between 100,000 and 120,000, then the number of schools required is 3.
- If the population of the district is between 180,000 and 220,000, then the number of schools required is 5.
- If the number of schools is 5, then the number of buses required is 15.
- If the number of schools is < 5 , then the number of buses required is 10.
- If the number of school buses is between 5 and 10, then the number of drivers required is 8.
- If the population is between 150,000 and 200,000, then the number of fire stations required is 4.
- For each school, the number of physicians required is 2.
- If the population is between 150,000 and 200,000, then the number of physicians required is 10.
- The district must be made up of contiguous counties.

There may be several more conditions that are required for this districting process. An important fact to be noted here is that the conditions may *change* periodically.

The query that pertains to this application is: Is the combination of County X, Y and Z an acceptable district?

In order to process this query, first the data pertaining to counties X, Y and Z are retrieved from the database. The combination of their characteristics is then tested against each of the conditions. If the conditions are satisfied, the district is acceptable.

There are two possible ways of representing the numerous conditions in the model. One way is to write procedural code in which all the conditions are

This poses a problem, because a particular combination of conditions may require other particular combinations of conditions to be satisfied. Thus, in the above example, the population affects the number of schools required, and the specific number of schools affects the number of buses required and the number of drivers required. The number of schools as well as the population of the district affect the number of physicians required.

Thus, the specification of such procedural code will have to be very detailed. All the possible rules and conditions, along with the procedures to actually construct the districts that meet the conditions will have to be incorporated. A change in one condition would affect several conditions, thus requiring changes to be made at several parts of the code. Furthermore, if the same type of query is *repeatedly* issued, the same set of procedural operations will have to be *repeatedly* performed, by the methods, which raises the cost of query processing.

An alternative to this approach is to store much of the conditions in declarative form as a set of *rules* that is *not* part of a particular spatial object but is incorporated within the model as instances of *Rule*. It is less complicated to use logic rules to specify recursive dependencies and other complex conditions, and they are also easier to understand and maintain [88, p70]. If a condition changes, only that specific rule will have to be changed, without worrying about the rest of the rules. The addition of a new rule or the deletion of an existing rule can be accomplished without disrupting the rest of the model.

Another advantage of using rules to specify and store conditions, pertains to queries which are applied repeatedly. In a rule based approach, some of the conditions and the intermediate results of prior queries may be saved and used by a subsequent query thus, improving the efficiency of the query processing [88, p84].

Given the above advantages, we incorporate *Rule* as an abstract class within our model.

We continue with the design process, by identifying the responsibilities of the individual classes, below.

- **The responsibilities of the individual classes** are:

  1. Every *Spatial_Feature* class has the following responsibilities:
     (a) Identifies and stores boundaries of itself.
     (b) Provides(requests) static characteristics, such as name and boundaries of itself(another spatial feature).
     (c) Provides distance information between itself and another spatial feature.
     (d) Provides(requests) point representation of itself(another spatial feature).
     (e) Provides(requests) the minimum and maximum x and y coordinates of itself(another spatial feature).
     (f) Provides the perspective when the query object is itself, or when the query objects are itself and another spatial feature.
     (g) Provides(requests) ancestors of itself(another spatial feature).
     (h) Provides(requests) spatial features *close_to* itself(another spatial feature).

(i) Provides spatial features *between* itself and another spatial feature.

(j) Provides(requests) spatial features *adjacent_to* itself(another spatial feature).

2. The responsibilities of class Line Segment are as follows:

   (a) Provides start-point and end-point of itself in terms of latitude and longitude.

   (b) Provides a line-id which is a unique identification of itself.

   (c) Constructs high-level Reference_Spatial_Feature such as State. (This responsibility is required in order to build the high-level object State from the low-level line segments. Once the State is created, then it is possible to construct objects that are *contained* by the State).

3. The responsibilities of class Reference_Spatial_Feature are as follows:

   (a) Provides(requests) all the classes of Reference Spatial Features it-(another spatial feature) contains.

   (b) Provides(requests) all the classes of Non Reference Spatial Features it(another spatial feature) contains.

   (c) Provides(requests) the Reference_Spatial_Feature(s) containing itself(another spatial feature).

   The responsibilities of class Reference_Spatial_Feature are inherited by abstract as well as concrete subclasses that may be identified and added to the model at a later stage in the design process. Concrete subclasses that may be added through this process are instantiated with Reference_Spatial-_Features.

   We notice at this stage in the design process that the possible subclasses of Reference_Spatial_Feature are Country, State, County and Town. These are concrete subclasses which can be instantiated with spatial features such as "USA" and "New Jersey" state. Thus, these above subclasses are identified and added to the model.

   Reference_Spatial_Feature's responsibilities are inherited by its subclasses Country, State, County and Town. However, the *implementation* of the methods corresponding to the responsibilities may differ depending on the specific subclass. Consider for example, the method *Make_Reference_Spatial_Features_within*, that corresponds to the responsibility: "Construct Reference_Spatial_Features it contains". The Country implementation of this method constructs the States that are contained within that country. Whereas, the same method implemented for a County constructs the Towns within the County.

   We also notice that in addition to inherited responsibilities and thus methods, depending on the problem domain, each subclass may also contain specific methods which are unique to that particular subclass. For example, in an electoral districting problem, the subclass State may have a specific method *Get_districts_contained*, which retrieves all the electoral districts that are defined within that state. On the other hand, the subclass Town may have a specific method *Get_district_containing*, which retrieves those districts that the town is part of.

4. The responsibilities of class Non_Reference_Spatial_Feature are as follows:

(a) Provides(requests) the Spatial Feature(s) containing it(another spatial feature).

(b) Provides(requests) the line segments that it(another spatial feature) is composed of.

Similar to Reference_Spatial_Feature, the responsibilities of Non_Reference_Spatial_Feature are inherited by its abstract and concrete subclasses, which may be identified later in the design process. Examples of concrete subclasses are: Interstate Highways, State Highways, County Roads and Rivers. These subclasses are instantiated with actual Road features and Highway features.

In addition to methods that correspond to the inherited responsibilities, each subclass may contain methods that apply to that specific subclass only, depending upon the specific problem domain.

5. The responsibilities of the class Rule are as follows:

   (a) Provides static characteristics, such as its name and its description.

   (b) Performs rule processing on the (spatial feature) arguments provided, and returns the results.

   The responsibilities of the *Rule* class are eventually inherited by all its concrete subclasses, which are then instantiated with rules.

   We notice at this stage of the design process that possible subclasses of the Rule class, given our problem domain, are: *Container_Rule*, *Ancestor_Rule* and *Object_Representation_Rule*. These concrete subclasses can be instantiated with rules that fit into the category of each subclass. The above subclasses are identified and added to the model.

   Within our problem domain, the subclasses of Rule have the following additional responsibilities:

   (a) The *Container_Rule*: reasons and provides the *container* of a given Reference *or* Non Reference Spatial feature by reasoning over the appropriate instance of *Container_Rule*.

   (b) The *Ancestor_Rule*: reasons and provides the *ancestors* of a given Reference **or** Non Reference Spatial feature.

   (c) The *Object_Representation_Rule*: reasons and provides the *object representation* of a given Reference or Non Reference Spatial Feature.

   (d) The *Adjacency_Rule*: reasons and determines if two given Reference or Non Reference Spatial Feature(s) satisfy the conditions for adjacency.

   Clearly, more subclasses of the *Rule* superclass can be added, depending upon the requirements of the problem domain.

Collaborations

In our model the following categories of collaborations are identified.

- Collaborations between Reference_Spatial_Features and Line Segments - to establish boundaries of the reference spatial features, and to determine the lines within the spatial features.
- Collaborations between Non_Reference_Spatial_Features and Line Segments - to establish boundaries of the Non Reference Spatial Features, and to determine the lines within the spatial features.

- Collaborations between Reference_Spatial_Features and Rules - to reason about notions such as contexts and object representations.
- Collaborations between Non_Reference_Spatial_Features and Rules - to reason about notions such as contexts and object representations.
- Collaborations between Reference and Non Reference Spatial Features - to determine the Reference Spatial Feature that contains the Non Reference Spatial Feature, and to determine imprecise spatial relationships between them by implementing operators such as "close-to", "between" and "adjacent-to".

7.5.2 The Analysis Phase

Analyze Hierarchies

In Section 7.5.1, we observe that Reference_Spatial_Features *contain* other Reference as well as Non Reference Features, whereas Non_Reference_Spatial_Features do not contain any other features. Analysis of the "Identify Responsibilities" and "Identify Collaborations" phases reveals that the classes Reference_Spatial_Feature and Non-_Reference_Spatial_Feature are specializations of the more *general* Spatial_Feature class, and inherit its responsibilities.

We also identified concrete subclasses of Reference_Spatial_Feature and Non_Reference_Spatial_Feature. Subclasses of the former are Country, State, County and Town. Subclasses of the latter are Interstate Highway, River, Lake, Railroad, State I-Hwy Segment, County I-Hwy Segment, Town I-Hwy Segment, State River Segment, County River Segment, Town River Segment, State Railroad Segment, County Railroad Segment, Town Railroad Segment, State Highway, County St-hwy Segment, Town St-hwy Segment, County Road, Town Cty-Rd Segment and Town Street.

The hierarchy that exists at this stage of the design process is given in Figure 7.1.

We notice, however, that the hierarchy in Figure 7.1 does not reveal any information on how the geographic objects are organized in the real world, and the relationships among them. That is, there is no notion of the different *perspective levels* that are inherent among geographic objects. All the concrete subclasses of Reference_Spatial-_Feature are organized into one group, and all the concrete subclasses of Non_Reference-_Spatial_Feature are organized into another separate group. The inter-relationships between these two groups are not apparent.

An analysis of the organization of geographic objects in the real world reveals that objects exhibit a *natural hierarchical ordering.* Thus, each Country contains Reference_Spatial_Features such as States, and Non_Reference_Spatial_Features such as Interstate Highways, and large Hydrographic Features (e.g. Large Lakes and Rivers); each State, in turn, contains Reference_Spatial_Features such as Counties and Non-_Reference_Spatial_Features such as State Highways. Each County contains Reference Features such as Towns, and Non Reference Features such as County Roads; each Town contains Reference Features such as Neighborhoods, and Non Reference Features such as Town Streets.

Further, at any level in the hierarchy, there exists *only one* particular *type* of subclass of Reference_Spatial_Feature. This subclass contains not more than *one* other type of subclass of Reference_Spatial_Feature, and *one or more* types of subclasses of Non_Reference_Spatial_Feature. For example, the class Country (a Reference_Spatial-_Feature) contains *one* type of subclass of Reference_Spatial_Feature, the class State, and one or more types of subclasses of Non_Reference_Spatial_Feature, such as class

Abstract class Spatial_Feature.

Abstract class Reference_Spatial_Feature *superclass* Spatial_Feature.
Concrete class Country *superclass* Reference_Spatial_Feature.
Concrete class State *superclass* Reference_Spatial_Feature.
Concrete class County *superclass* Reference_Spatial_Feature.
Concrete class Town *superclass* Reference_Spatial_Feature.

Abstract class Non_Reference_Spatial_Feature *superclass* Spatial_Feature.
Concrete class Interstate Highway *superclass* Non_Reference_Spatial_Feature.
Concrete class River *superclass* Non_Reference_Spatial_Feature.
Concrete class Railroad *superclass* Non_Reference_Spatial_Feature.
Concrete class Lake *superclass* Non_Reference_Spatial_Feature.
Concrete class State Highway *superclass* Non_Reference_Spatial_Feature.
Concrete class State I-Hwy Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class State River Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class State Railroad Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class County Road *superclass* Non_Reference_Spatial_Feature.
Concrete class County I-Hwy Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class County River Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class County Railroad Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class County St-hwy Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class Town Street *superclass* Non_Reference_Spatial_Feature.
Concrete class Town I-Hwy Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class Town River Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class Town Railroad Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class Town St-hwy Segment *superclass* Non_Reference_Spatial_Feature.
Concrete class Town Cty-Rd Segment *superclass* Non_Reference_Spatial_Feature.

Figure 7.1: Preliminary geographic-objects hierarchy

Interstate Highway, class River and class Railroad. In the same way, the class State (a Reference_Spatial_Feature) contains *one* type of subclass of Reference_Spatial_Feature, the class County, and one or more types of subclasses of Non_Reference_Spatial_Feature, such as class State Highway, State I-Hwy Segment and State River Segment. This ordering proves to be very useful in describing *perspective levels* — each level in the containment hierarchy represents a single perspective level, such as Country level, State level, County level and Town level.

We incorporate this view of the real world containment hierarchy explicitly in our model by specifying that:

- A Reference_Spatial_Feature at a particular level in the hierarchy can contain not more than *one* other Reference_Spatial_Feature, and the contained Reference-_Spatial_Feature must be of a different *type* (subclass) than the container Reference-_Spatial_Feature.
- A Reference_Spatial_Feature at a particular level in the hierarchy can contain one or more Non_Reference_Spatial_Features, and the contained features may belong to different types of subclasses as long as those subclasses are not also contained by any *other* Reference_Spatial_Feature.

Analyze Subsystems

The following subsystems are identified.

The collaborations between Reference_Spatial_Feature, Non_Reference_Spatial_Feature, Line_Segments and Rules indicate the existence of a *subsystem* which is comprised of: Spatial_Feature, its subclasses Reference_Spatial_Feature and Non_Reference_Spatial-_Feature, and their respective subclasses such as Country and State. Thus, if at a later stage a new subclass, such as "Neighborhood" is to be added to the model, this could be accomplished by extending this subsystem, without disrupting the rest of the application.

In addition to the *Spatial_Feature* subsystem, we also identified another subsystem that results from the collaboration between class Reference_Spatial_Feature and class Rule, and class Non_Reference_Spatial_Feature and class Rule. This subsystem is comprised of: Rule and its subclasses such as "Ancestor_Rule" and "Object-Representation_Rule". The advantage of having such a subsystem is that the addition of a new class of rules at a later stage could be accomplished without affecting any of the other classes in the model.

Create Protocols

This task involves specifying the implementation of the abstract classes and their behavior which is specified in the form of methods. The base classes as identified above are: *Spatial Feature* and *Line Segment.*

The base and abstract methods for each of the classes so far identified are discussed below.

- **Base Methods.**
  The following *base methods* are specified for the *base classes*:

  1. Base methods of class Spatial_Feature:
     (a) *Get_boundary* - provides the boundaries of the spatial feature by collaborating with the class *Line Segment.*

(b) *Get_distance_between* - provides the distance between itself and another spatial feature.

(c) *Find_perspective* - determines the perspective of a query by identifying the spatial features that are addressed in the query, and then collaborating with the *Rule* classes.

(d) *Get_max-x_and_min-x* - computes the maximum and minimum x coordinate values of spatial feature by collaborating with *Line Segment.*

(e) *Get_max-y_and_min-y* - computes the maximum and minimum y coordinate values of spatial feature by collaborating with *Line Segment.*

2. Base methods of class Line_Segment:

(a) *Make_Reference_Spatial_Features* - constructs spatial features by grouping the line segments belonging to a a given spatial feature.

- **Abstract Methods.**

These methods are defined when it is required to have a particular method be inherited down a hierarchy with each subclass having a *different implementation* of the method.

In our application, an abstract method *Make_boundary* is described in the superclass *Spatial_Feature.* This method is inherited down the hierarchy by the subclasses *Reference_Spatial_Feature* and *Non_Reference_Spatial_Feature*, and further down, by *their* subclasses *Country, State, County and Town*, and *Interstate_Highway, State_Highway, County_Road and Town_Street.* Here, *Country, State,* etc., are each subclass of *Reference_Spatial_Feature* existing at the same level of the hierarchy. As noted earlier in Section 7.5.1, the *implementation* of the methods corresponding to the responsibilities may differ depending on the specific subclass. Thus, the State implementation of the method *Make_boundary* determines those line segments that form the boundaries of States (i.e., those line segments which have two *different* States lying on each of their sides), and then builds the boundaries of the States. Whereas, the same method implemented for a County determines the line segments that form the boundaries of Counties, and then builds the boundaries of the Counties.

We emphasize this point by giving another example: Consider, the method *Make_Reference_Spatial_Features_within*, that corresponds to the responsibility: "Construct Reference_Spatial_Features it contains". The Country implementation of this method constructs the States that are contained within that country. Whereas, the same method implemented for a County constructs the Towns within the County.

We list the abstract methods in our application, in Appendix 7.7.

- **Overloaded methods.**

In addition to the *Base Method*, which is inherited down a hierarchy without any change in its implementation, and the *Abstract Method*, in which the implementation may change down the hierarchy, we use another type of method, referred to as *Overloaded Method.*

An Overloaded method is inherited from the Base Class down the hierarchy. In an Overloaded Method, *different* implementations of the *same* method can be described for the *same* class, by providing the method with different kinds of information, in the form of *parameters.* This functionality is useful in our application. For example, a spatial feature such as County may have a method

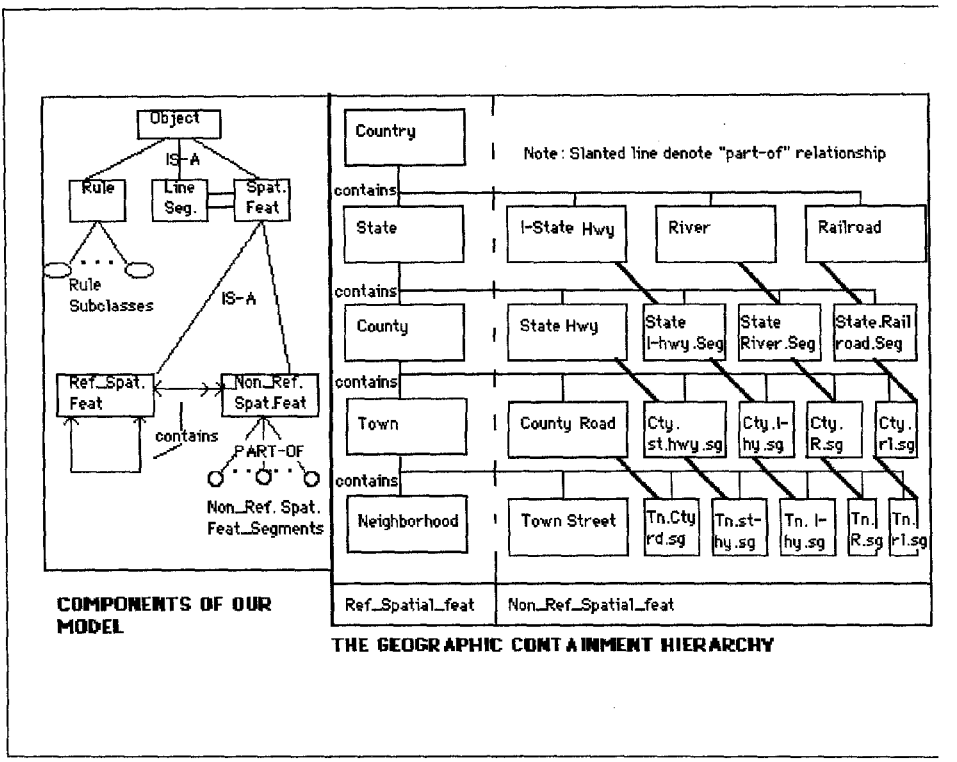


Figure 7.2: The Spatial Data Model

Find_object_close_to. We mentioned in Section 7.2 that *close_to* is a notion that may have different interpretations depending upon the user perspective or context. Now if we invoke the method *Find_object_close_to* on a particular instance of County without giving any further information about the user perspective, the method may carry out a default implementation. If, on the other hand, we provide the method with additional information about the perspective, such as *perspective = Country level,* or *perspective = State level,* the method will carry out different implementations, depending upon the perspective provided.

Overloaded methods help us achieve better *encapsulation.* Had the Overloaded Method functionality not been available, we would have had to provide the different perspective-varying implementations by using a series of *"if-then-else" Statements* which would have limited the program reusability and understanding. The Overloaded Methods identified in our model are listed in Appendix 7.7.

This completes our discussion of the Spatial Data Model. The components of the model, and the geographic containment hierarchy that we model, are summarized in Figure 7.2.

7.6 Implementation

The model was implemented using LAURE which is an object-oriented language developed at Bellcore by Caseau [88]. LAURE is a powerful knowledge representation language based on sets. It is both an interpreted and a compiled language, and can be used easily in both modes. This feature is especially useful during the system development and testing stages by providing a fast prototyping functionality. LAURE supports large programs, provides efficiency similar to C^{++}, and offers a clean interface with C. In addition, LAURE provides a deductive system with sound, efficient and complete resolution of deductive rules, a feature that is very useful in our application to reason about user perspectives and object representation in relation to user perspectives, and to process imprecise queries.

We give below, only brief details of our LAURE implementation, since a comprehensive discussion is beyond the scope of this chapter. For a comprehensive discussion of our implementation, please see [567]. A description of the class Spatial_Feature, along with some sample queries and answers in LAURE syntax, is presented in Appendix 7.6. The class Spatial_Feature described in the appendix contains method-templates (the actual code is not included here, since it is beyond the scope of this discussion) for determining imprecise relationships such as "close_to", "between", and "adjacent_to".

For the purposes of our implementation, we used "raw" geographic line segment data from the U.S. Bureau of Census' TIGER file [595]. The line segments that we used pertained to the state of Rhode Island, U.S.A. Approximately 1500 low-level line segments that covered a cross section of all the counties, towns, state highways, county roads and town streets were loaded into the model, and then queries were posed to the geographic database.

The queries that we posed consisted of both precise as well as imprecise topological operators. Some query examples (in English) are given below. The actual query syntax can be found in Appendix 7.6.

Q_1. Add the boundary of "Bristol" county to State boundary.
Comment: Evaluates to "true" implying that the request was accomplished successfully.

Q_2. Get the names of the counties contained by Rhode Island.
result: returns Bristol, Kent, Newport, Providence, Washington.

Q_3. Get all the Non-reference-feature-contained of State Rhode Island.
Comment: returns Summit Greene Road, Plainfield Pike, Wallum Lake Road.

Q_4. Is Providence town *Close_to?* Bristol town?
Comment: Here *close_to?* is an imprecise operator. Since an explicit perspective is not provided, the system determines the possible perspective, using the embedded rules, and then affirms that perspective with the user. Based on the user's response, additional information is assumed or computed. Then the result to the query is computed and returned.

Q_5. Get all the Towns that are *Close_to!* Providence town.
Comment: The processing is similar to query Q_4.

In addition to the above queries, it is also possible to combine query operators to form conjunctive queries such as: "Get all the Towns that are *close_to* Providence **AND** lie *between* Cumberland and Cranston **OR** are *adjacent_to* Bristol". In all of these cases, our implementation proved to successful and thus robust.

7.7 Conclusion

In this chapter we presented our experience in applying the Responsibility-Driven Approach to the design and implementation of a complex, real-life geographic application. The approach's emphasis on identifying and designing object-classes based on *responsibilities* rather than their *characteristics* proved to be very effective in achieving a robust design and implementation that is close to user requirement and at the same time enhancing the application's extensibility and maintainability.

Future work includes investigating the applicability of our geographic model to a varied set of problem domains such as electoral districting and public utilities management.

APPENDIX

A.1 List of Abstract Methods

1. Abstract Methods of class Spatial_Feature.
   (a) *Get_point_representation* with null implementation.
   ("Null implementation" means that the method defined for the object in question does not have any implementation. It exists as a *template* which is inherited by the subclasses of the object. A method inherited in such a manner by a subclass may have a real (non-null) implementation, corresponding to the responsibilities of the subclass).
   (b) *Make_boundary* with null implementation.
   (c) *Make_point_representation* with null implementation.
   (d) *Make_min-max-x* with null implementation.
   (e) *Make_min-max-y* with null implementation.
2. Abstract Methods of class Reference_Spatial_Feature (superclass Spatial_Feature).
   (a) *Get_point_representation* with implementation to get point representation of Reference_Spatial_Feature.
   (b) *Make_boundary* with null implementation.
   (c) *Make_point_representation* with null implementation.
   (d) *Make_min-max-x* with null implementation.
   (e) *Make_min-max-y* with null implementation.
   (f) *Make_Reference_Spatial_Features_within* with null implementation.
   (g) *Make_Non_Reference_Spatial_Features_within* with null implementation.
3. Abstract Methods of class Non_Reference_Spatial_Feature (superclass Spatial_Feature).
   (a) *Get_point_representation* with implementation to get point representation of Non_Reference_Spatial_Feature.
   (b) *Make_boundary* with null implementation.
   (c) *Make_point_representation* with null implementation.
   (d) *Make_min-max-x* with null implementation.
   (e) *Make_min-max-y* with null implementation.

It is worth noting that we make use of the Abstract Methods for such implementation details as loading the raw data which is originally at a low-level (line segments), and then constructing high-level objects such as States, Counties, Towns and Roads from the low-level objects. This is a very useful and important functionality of our model because it proved to us that the model was capable of **reflective** capabilities i.e., the model's components *themselves* could be used for loading the raw data, and deriving high-level data and populating the various classes that are described in the model.

We continue our discussion of Abstract Methods by listing those which pertain to the high level objects Country, State, County and Town. The Abstract Methods listed below are used to get the reflective functionality discussed above.

4. Abstract Methods of class Country (superclass Reference_Spatial_Feature). It should be noted that the Abstract Methods encapsulated by subclasses such as Country have specific implementations corresponding to their responsibilities).
   (a) *Make_boundary* with implementation to construct Country from Line_Segment.
   (b) *Make_point_representation* with implementation to compute point representation of Country.
   (c) *Make_min-max-x* with implementation to compute the maximum and minimum x coordinate values of Country.
   (d) *Make_min-max-y* with implementation to compute the maximum and minimum y coordinate values of Country.
   (e) *Make_Reference_Spatial_Features_within* with implementation to construct States contained by Country, in collaboration with Line_Segment.
   (f) *Make_Non_Reference_Spatial_Features_within* with implementation to construct Non_Reference_Spatial_Features at the State level that is contained by Country, in collaboration with Line_Segment.
5. Abstract Methods of class State (superclass Reference_Spatial_Feature).
   (a) *Make_boundary* with implementation to construct State from Line_Segment.
   (b) *Make_point_representation* with implementation to compute point representation of State.
   (c) *Make_min-max-x* with implementation to compute the maximum and minimum x coordinate values of State.
   (d) *Make_min-max-y* with implementation to compute the maximum and minimum y coordinate values of State.
   (e) *Make_Reference_Spatial_Features_within* with implementation to construct Counties contained by State, in collaboration with Line_Segment.
   (f) *Make_Non_Reference_Spatial_Features_within* with implementation to construct Non_Reference_Spatial_Features at the County level that is contained by State, in collaboration with Line_Segment.
6. Abstract Methods of class County (superclass Reference_Spatial_Feature).
   (a) *Make_boundary* with implementation to construct County from Line_Segment.
   (b) *Make_point_representation* with implementation to compute point representation of County.
   (c) *Make_min-max-x* with implementation to compute the maximum and minimum x coordinate values of County.
   (d) *Make_min-max-y* with implementation to compute the maximum and minimum y coordinate values of County.
   (e) *Make_Reference_Spatial_Features_within* with implementation to construct Towns contained by County, in collaboration with Line_Segment.

(f) *Make_Non_Reference_Spatial_Features_within* with implementation to construct Non_Reference_Spatial_Features at the Town level that is contained by County, in collaboration with Line_Segment.

7. Abstract Methods of class Town (superclass Reference_Spatial_Feature).

   (a) *Make_boundary* with implementation to construct Town from Line_Segment.

   (b) *Make_point_representation* with implementation to compute point representation of Town.

   (c) *Make_min-max-x* with implementation to compute the maximum and minimum x coordinate values of Town.

   (d) *Make_min-max-y* with implementation to compute the maximum and minimum y coordinate values of Town.

   (e) *Make_Reference_Spatial_Features_within* with null implementation since in our application the Town class does not contain any Reference_Spatial_Feature.

   (f) *Make_Non_Reference_Spatial_Features_within* with implementation to construct Non_Reference_Spatial_Features *below* the Town level that is contained by Town, in collaboration with Line_Segment.

8. Abstract Methods of *subclasses* of Non_Reference_Spatial_Feature.

   The classes Interstate_Highways, State_Highways, Rivers, Railroads, County_Roads and Town_Streets fall under this category. These are linear objects which do not *contain* any other object. In order to avoid the repetition of ideas, we provide specifications of the Abstract Methods for only *one* representative class of such objects here.

   (a) *Make_boundary* with implementation to group the component line segments which make up the concerned object.

   (b) *Make_point_representation* with null implementation.

   (c) *Make_min-max-x* with implementation to compute the values of the maximum and minimum x coordinate values of the concerned object.

   (d) *Make_min-max-y* with implementation to compute the values of the maximum and minimum y coordinate values of the concerned object.

A.2 List of Overloaded Methods

1. Overloaded Methods of Spatial_Feature:
   In the following, "X", "Y" and "Z" represent spatial features.

   (a) X **close_to?** Y.
   Determines if X is close to Y.

   (b) **close_to!** X.
   Determines the spatial features that are close to X.

   (c) Z **between?** X, Y.
   Determines if Z lies between X and Y.

(d) X **between!** Y.
Determines the spatial features between X and Y.

(e) X **adjacent_to?** Y.
Determines if X and Y are adjacent to each other.

(f) **adjacent_to!** X.
Determines the spatial features adjacent to X.

A.3 Implementation notes in LAURE syntax

A.3.1 The Spatial_feature class

```
[ spatial_feat :: union superset (named_object)
comment "represents high level spatial objects, e.g. State"
with
    (slot name -> string)      ; i.d. code
    (multi_slot boundary -> line_seg)
    (slot elevation -> number)
    (multi_slot lines_within -> line_seg)
    (method point_reprn -> point)
          => [the implementation code goes here] )
    (multi_method adjacent_to -> spatial_feat
          comment "finds features adjacent to itself"
          => [the implementation code goes here] )
    (method adjacent_to? (x:spatial_feat) -> boolean
          comment "finds if x is adjacent to oself"
          => [the implementation code goes here] )
    (multi_method  between (b:spatial_feat) -> spatial_feat
         comment "finds objects between oself and b"
          => [pers as nil [ pers <- [[[[ancestors(oself) find]
             intersect [ancestors(b) find]] nth ] owner] ]]
             [if [and [rep(oself) = "poly"] [rep(b) = "poly"]]
                  [the implementation code goes here]
             else_if [and [rep(oself) ="point"] [rep(b)="point"]]
                  [the implementation code goes here]
             else [the implementation code goes here]
             ]  )
    (method between? (x:spatial_feat y:spatial_feat
              perspective:string) -> boolean
          comment "finds if y is between oself and x"
          => [the implementation code goes here] )
    (multi_method close_to (perspective:string) -> spatial_feat
          comment "finds features close to itself"
          => [the implementation code goes here] )
    (method close_to?(x:spatial_feat perspective:string)->boolean
          comment "finds if x is close to it"
          => [the implementation code goes here] ) ]
```

NOTE: ''pers'' denotes the perspective, and ''rep'' denotes the object representation.

A.3.2 Sample Queries — (in LAURE syntax)

```
Q1. Add boundary of "Bristol" county to State boundary

laure> [y all [x select State [[x name] = "RI"]] [[y boundary]
        add ["Bristol" get_boundary "RI"]]]

eval> t    ;;;returns 'true' indicating successful completion of
           query

Q2. Get the names of the counties contained by a State RI

laure> [y all [[x get_all State [[x R_feat_contained] ?]] nth 1]
       ["~A~%" printf [y name]]]

eval> Bristol
      Kent
      Newport
      Providence
      Washington

Q3. Get all NR_feat_contained of State RI

laure> [y all [x all State [[x NR_feat_contained] return]]
       ["~A~%" printf [y name]]]

eval> Summit Greene Road
      Plainfield Pike
      Wallum Lake Road

Q4. Is Providence Town close to Narragansett?

laure> [[[x select Town [[x name] = "Providence"]] nth 1]
       close_to?
       [[x select Town [[x name] = "Narragansett"]] nth 1]
       {} geometric]

eval> (determines user perspective, computes and returns true or
      false)

Q5. What are the towns close to Providence Town?

laure> [[[x select Town [[x name] = "Providence"]] nth 1]
       close_to]

eval> (determines user perspective, computes and returns all towns
      close to Providence)
```

Chapter 8

Indexical Databases

James Clifford*

8.1 Motivation

The "three great data models" [594] were created in response to a set of needs arising in certain traditional data management and processing environments, and to a large extent they have been successful in meeting those needs. Today, however, the expanded use of. and familiarity with, a variety of computers and software systems are generating more sophisticated data management and processing needs. Today's sophisticated users are running into a kind of "brick wall" in the current generation of Database Management Systems (DBMS), nearly all of which are based on one of these three data models. This phenomenon is, to be sure, not unique to the database arena; by and large *all* areas of computer usage are experiencing the limits of today's software. The growing revolution in expert systems, decision support systems, etc., all of which are attempts to tackle ever more sophisticated problems for which traditional programming languages and software development methodologies are largely inadequate, attests to this fact.

Simply put, today's data models and DBMS's were largely designed for storing and retrieving *facts.* (Recent work in the area of "object-oriented database systems" does not seem to have digressed from this overall perspective.) While this functionality is sufficient for many, if not most, of the applications and functions which the corporate DBMS is intended to service, a growing segment of the DBMS user community, having become comfortable, in many cases proficient, with the *"just the facts, please..."* mode of use is now interested in having the system support much more of their needs. Among the kinds of information needed are the following: opinions, expectations, judgments, personal observations, histories, predictions, expert advice, hypothesized scenarios, design versions, locations in space/time, simulations, sources of data, model or software used to compute the data, etc.

Certainly most, if not all, of these functions *can* be met by some combination of the DBMS and a host programming language. But by and large the advantage of a DBMS has been the accessibility of the information and the functionality of the system to end users through an *interactive query language.* The "Host language + DML" approach has, of course, always supported the transaction processing and report production component of data processing, but the interactive query language mode is what has made a DBMS so attractive, because, by abstracting the *general* functions of

*Department of Information Systems, Leonard N. Stern School of Business, New York University, New York, New York 10012-1126.

database querying into a general purpose set of querying functions, it allows access to the contents of the database *without the need for programming.*

Today there are many research efforts underway to expand the data structuring and data processing functionality of a DBMS to meet these expanding needs. The economic realities of today's world would seem to doom most of these to failure, however, for two major reasons. One is that each such effort has targeted, by and large, only one facet of the problem, one potential user community, if you will, and the other is the growing commercial success of today's relational systems and the SQL language in particular.

This chapter is an attempt to generalize the notion of an extended relational model. Building upon the author's experience in one particular type of relational extension, namely an extension to capture the semantics of the temporal dimension of data at the model level ([116], [117], [123], [122],[119], [120]), this chapter investigates a model to encapsulate a wide class of such extensions into the notion of a generalized functional data type.

The basic model for the semantics of this expanded model, called the **Indexical Data Model (IDM)**, is borrowed from intensional logic, an attempt to formalize the *pragmatic* component of linguistic theory. An intensional logic looks at the phenomenon of *context* as a major contributing component to defining the interpretation of a language. As described by Richard Montague [414]:

> In interpreting a pragmatic language L we shall have to take into account the possible contexts of use. It is not necessary to consider them in their full complexity; we may instead confine our attention to those among their features which are relevant to the discourse in question. Thus it will suffice to specify the set of all complexes of relevant aspects of intended possible contexts of use. We may call such complexes *indices*, or to borrow Dana Scott's term, *points of reference.* For instance, if the only indexical feature of L were the occurrence of tense operators,[†] then the points of reference might naturally be chosen as moments of time, regarded as possible moments of utterance. On the other hand, if L contained in addition the first person pronoun 'I' ... two aspects of the context of use would become relevant, the speaker as well as the moment of utterance; and a point of reference might naturally be chosen as an ordered pair consisting of a person and a moment of time.

The **Indexical Data Model**, therefore, applies the same notion of indexical semantics to the realm of relational databases, in recognition of the need for potentially many *points of reference* in increasingly complex database applications. Moreover, following the guidelines proposed in [118], the model to be proposed will be a consistent extension, not only of the Historical Relational Data Model (HRDM) [119], but more importantly of the underlying relational data model itself. In this way we believe that there is some hope that commercial systems, built upon the model, have a chance of success – because the model is a consistent relational extension, a DBMS built upon the system has some chance of being truly "upwardly compatible" with existing DBMS such as *DB2* or *Ingres.*

[†]It was precisely consideration of such a situation that informed the development of the Historical Relational Data Model (HRDM) [119].

8.2 The Indexical Database Model

In this section we present an overview of the Indexical Database Model. We begin with a presentation of the structures of the model – indexical relations – which will be seen as a general extension to ordinary relations. We then provide an overview of a relational algebra for the model, focusing on those operators which take advantage of the expanded representational capability of the indexical relations.

8.2.1 The Structures

Let UVD = $\{VD_1, VD_2 \ldots, VD_{n_{vd}}\}$ be a (universal) set of **value domains** where for each i, $VD_i \neq \emptyset$. Each value domain VD_i is analogous to the traditional notion of a domain in that it is a set of atomic (non-decomposable) values.

(For example, an example application might have $VD_1 = \{Management, Finance, Accounting\}$.)

Let UA = $\{A_1, A_2, \ldots, A_{n_a}\}$ be a (universal) set of **attributes**. Each attribute names some property of interest in the application area.

(For example, we might have $A_1 = DEPARTMENT$.)

Let UID = $\{ID_1, ID_2, \ldots, ID_{n_{id}}\}$ be a (universal) set of **index domains**, where for each i, $\mathrm{ID}_i \neq \emptyset$, and the cardinality of each ID_i is at most countably infinite.

(For example, we might have $ID_1 = \{Halderman, Ehrlichman, Mitchell\}$, and $ID_2 = \{\ldots, t_0, t_1, \ldots\}$.)

Let UI = $\{I_1, I_2, \ldots, I_{n_i}\}$ be a (universal) set of **indices**. Each index I_i represents a "contextual coordinate"[354] which contributes to the context in which a particular fact is to be interpreted.

(For example, we might have $I_1 = VALIDTIME$, and $I_2 = OBSERVERS$.)

The sets UVD, UA, UID and UI are all pairwise disjoint. $IDOM : UI \rightarrow UID$ is a function which associates with each index I in UI its **index domain** ID in UID. We denote the index domain of index I by $IDOM(I)$. In order to give a uniform meaning to each index I the function $IDOM$ is defined at the database scheme level, and *not* at the relation scheme level.

(For example, we might have $IDOM(OBSERVERS) = ID_1$, and $IDOM(VALIDTIME) = ID_2$.)

Let VC = $\{\Theta_{VD_1}, \ldots, \Theta_{VD_{n_d}}\}$ be a set of value comparator sets, where each $\Theta_{VD_i} = \{\theta_{VD_{i_1}}, \ldots, \theta_{VD_{i_n}}\}$ is a set of **value comparators** over VD_i. More precisely, each $\theta_{VD_{i_j}}$ in Θ_{VD_i} is a set of ordered pairs $\{< v_j, v_k > | v_j, v_k \in VD_i$ and $v_j\ \theta_{VD_{i_j}}\ v_k\}$. Minimally, we require for each value domain VD_i that $\{=, \neq\} \subseteq \theta_{VD_{i_j}}$, i.e., that all value domains support equality and inequality comparisons.

Let IC = $\{\Theta_{ID_1}, \ldots, \Theta_{ID_{n_i}}\}$ be a set of index comparator sets, where each $\Theta_{ID_i} = \{\theta_{ID_{i_1}}, \ldots, \theta_{ID_{i_n}}\}$ is a set of **index comparators** over ID_j. More precisely, each $\theta_{ID_{i_j}}$ in Θ_{ID_i} is a set of ordered pairs $\{< id_p, id_q > | id_p, id_q \in ID_j$ and $id_p\ \theta_{ID_{ID_j}}\ id_q\}$. Minimally, we require for each index domain ID_j that $\{=, \neq\} \subseteq \theta_{ID_{i_j}}$, i.e., that all index domains support equality and inequality comparisons.

These preliminary definition serve, in effect, to define the basic vocabulary of our model. Armed with them, we can proceed to define an **Indexical Relation scheme R** as a 5-tuple $R =< A, K, VDOM, IDOM, DOM >$ where:

1. $A \subseteq UA$ is the set of **attributes** of scheme R
2. $K \subseteq A$ is the **designated key** of scheme R.

3. $VDOM : A \rightarrow UVD$ is a function which gives the **value domain** of each attribute of scheme R; we denote the value domain of attribute A_i in scheme R by $VDOM(A_i, R)$.
4. $IND : A \rightarrow 2^{UI}$ is a function which gives the set of **indices** of each attribute of scheme R; we denote the indices of attribute A_i in scheme R by $IND(A_i, R)$. [‡]
5. The **domain** of an attribute A_i in scheme R, denoted $DOM(A_i, R)$, is a function $DOM : A \rightarrow VDOM(A_i, R)^{2^{IDOM(IND(A_i,R))}}$ which gives the domain of each attribute of scheme R. Note that the domain of an attribute is the set of all possible functions from its $IDOM$ to its $VDOM$.

A **tuple t on scheme R** is an ordered pair $t =< i, l >$ where

1. $t.i : A \rightarrow I \rightarrow 2^{IDOM(IND(A_i,R))}$ the **index lifespan of attribute A in tuple t**, is a function that gives the **index lifespan** of each attribute $A_i \in A$. We denote the **index lifespan of attribute** A_i **in** t as $t.l.A_i$.
2. $t.v : A \rightarrow T^{UD}$ is a function that associates with each attribute $A_i \in A$ a temporal-based function from the **attribute index lifespan** $t.l(A_i)$, to the domain assigned to attribute A_i. That is, $t(A_i) : t.l(A_i) \rightarrow DOM(A_i)$ We denote the **value of attribute** A_i **in** t as $t.v.A_i$. Note that $t.v.A_i$ is subject to two constraints:
   (a) the **domain constraint**, namely, $\forall i[t(A_i) \in DOM(A_i)]$, and
   (b) the **key constraint**, namely, for any two *distinct* tuples t_i, t_j, $t_i(K) \neq t_j(K)$.

In general, we would like to allow a tuple to be only partially defined; i.e., if the domain of an attribute A_i in relation R $DOM(A_i, R)$ is the set of all functions in $VDOM(A_i, R)^{IDOM(A_i,R)}$, at any given time the tuple instance will be only a *partial function* in this space. We omit the details of this point, but point out that it would be analogous to our treatment of *lifespans* in [119]. In other words, the following points would have to be addressed:

1. A notion of *index span*, similar to that of a *lifespan* in historical databases, would need to be defined for each index
2. The issue of the *homogeneity* ([202], [119]) of the tuple in each of the index dimensions would need to be addressed. In other words, proceeding from the most to the least general treatment, each attribute-value pair could have its own *index span* for each index, or each tuple could be homogeneous in all of its index dimensions, or each relation could be homogeneous in all of its index dimensions.

Finally, we can define an **indexical relation r on scheme R** as a finite set of tuples, $r = \{t_1, t_2, \ldots t_n\}$, on scheme R.

8.2.2 Discussion of the Structures

As discussed in [120]), there have been two different strategies for incorporating a temporal dimension into the relational model in the literature. In one, the schema of the relation is expanded to include one or more *distinguished* temporal attributes (e.g.,

[‡]Note that it is by allowing a different $IDOM$ for each attribute of R that our model is inhomogeneous.

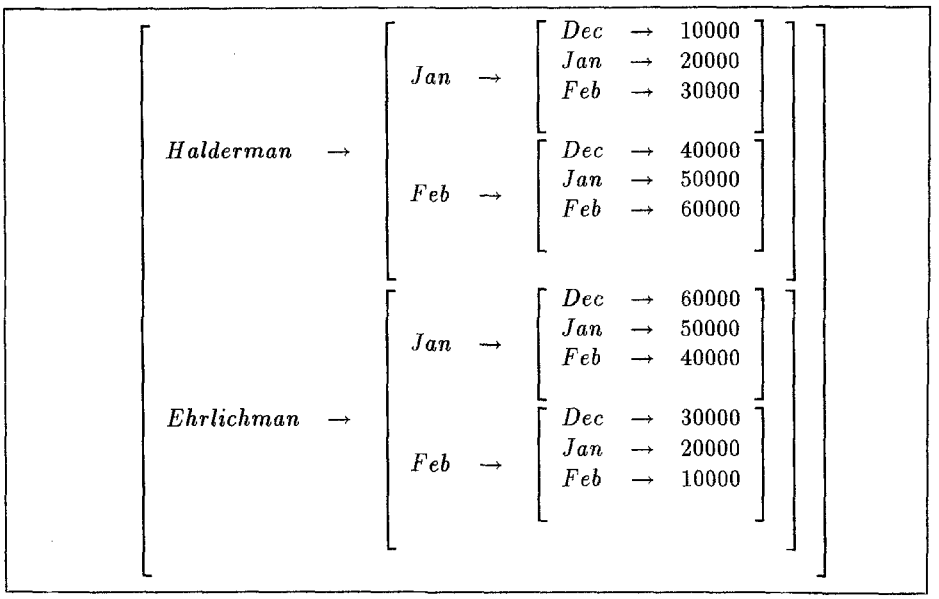


Figure 8.1: A Complex Function as an Attribute Value

time, or *START-TIME* and *END-TIME*) to represent the period of time over which the *fact* represented by the tuple is to be considered valid. This approach has been referred to in the literature as *tuple time-stamping* or as a *first-normal form (1NF)* model; in [120]) the term *ungrouped* is introduced for this type of approach. In the other approach, referred to as *attribute time-stamping* or as a *non-first-normal form (N1NF)* model, instead of adding additional attributes to the schema, the domain of each attribute is extended from simple values to complex values (functions, e.g.) which incorporate the temporal dimension. [120]) introduces the term *grouped* for this latter approach.

In the ungrouped approach an "object's" entire history is represented within a single tuple, within which the time stamps are embedded as *components* of the values of each attribute. In the grouped models, by contrast, all of the information about an object is represented in a single tuple. [120] contrasts these two approaches and shows that *temporally grouped* models are more expressive than *temporally ungrouped* models. The Indexical Data Model, then, is a *grouped* model in precisely the same sense or for precisely the same reasons. Thus, a value in the Indexical Database Model is not atomic, but rather a complex function, like the one in Figure 8.1.

It is well known (see discussion in [112]) that any function of n arguments can be represented by an equivalent function of $n-1$ arguments. We can therefore choose to represent n-place functions like:

$$< e_1, < e_2, < \ldots < e_n, v > \ldots >>>$$

by their equivalent 1-place function:

$$< \{e_1, e_2, \ldots, e_n\}, v >$$

$< Halderman, Jan, Dec >$	$\rightarrow$	10000
$< Halderman, Jan, Jan >$	$\rightarrow$	20000
$< Halderman, Jan, Feb >$	$\rightarrow$	30000
$< Halderman, Feb, Dec >$	$\rightarrow$	40000
$< Halderman, Feb, Jan >$	$\rightarrow$	50000
$< Halderman, Feb, Feb >$	$\rightarrow$	60000
$< Ehrlichman, Jan, Dec >$	$\rightarrow$	60000
$< Ehrlichman, Jan, Jan >$	$\rightarrow$	50000
$< Ehrlichman, Jan, Feb >$	$\rightarrow$	40000
$< Ehrlichman, Feb, Dec >$	$\rightarrow$	30000
$< Ehrlichman, Feb, Jan >$	$\rightarrow$	20000
$< Ehrlichman, Feb, Feb >$	$\rightarrow$	10000

Figure 8.2: Example of Attribute Value as 1-Place Function

Thus, the value in Figure 8.1 could equivalently be represented as in Figure 8.2. **IDM** can make extensive use of this ability — at the definitional, operational, or user levels — to view values in either of these two fashions. In addition, since the order of the indices is irrelevant in these functions, they can be viewed in whatever order the user deems appropriate to the task at hand.

8.2.3 An Indexical Example: The Watergate Database

In this section we give an example from the familiar world of politics, of an Indexical Database with three index sets. These are used to represent answers to the familiar questions asked during the televised *Senate Watergate Hearings*, i.e. "What did the President know and when did he know it?". We generalize slightly, and have an index set for each of these three points of reference: (i) "Who knew it", (ii) "When was it known?" and (iii) "When was it believed it to have occurred?"

For our *Watergate* example, we might choose to define the following three index sets:

$I_1 = Observers =$
$\{Halderman, Ehrlichman, Dean, Nixon, Mitchell, Colson, Liddy\}$
$I_2 = Data\_Time = \{July, August, September, October\}$
$I_3 = Rec\_Time = \{July, August, September, October, November\}$

For simplicity, we will consider only a single relation on the following scheme:
$PROJECTS =< A_{PROJECTS}, K_{PROJECTS}, VDOM_{PROJECTS}, IDOM_{PROJECTS}$
where:

1. $A_{PROJECTS} = \{PNAME, APPROPRIATION, APPROVER\}$
2. $K_{PROJECTS} = \{PNAME\}$
3. $VDOM_{PROJECTS}$ is as follows:

$VDOM_{PROJECTS}(PNAME) = \{Watergate\ BreakIn, Watergate\ Coverup,$
$Ellsberg\ BreakIn\}$
$VDOM_{PROJECTS}(APPROPRIATION) =$ positive integers
$VDOM_{PROJECTS}(APPROVER) =$

$\{Nixon, Mitchell, Halderman, Ehrlichman\}$

4. $IDOM_{PROJECTS}$ is as follows:

$IDOM_{PROJECTS}(PNAME) = \emptyset$
$IDOM_{PROJECTS}(APPROPRIATION) =$
$\{Observers, Data\_Time, Rec\_Time\}$
$IDOM_{PROJECTS}(APPROVER) = \{Observers, Data\_Time, Rec\_Time\}$

We can more simply view the structure of this relation as follows:
PROJECTS(P_NAME,
APPROPRIATION: <Observer, Rec_Time, Data_Time>,
APPROVER: <Observer, Rec_Time, Data_Time>)
where each attribute is followed, where necessary, by the ordered list of its *indices*. Note that we have indicated that the attribute P_NAME is the *key* of this relation, and have decided that it is not *indexical*, i.e., it's value is not a function from any of the indices in the model.

In the rest of the chapter, in discussing examples of algebraic operators for the indexical data model, we will refer to the instance on this schema shown in Figure 8.3.

Note that other applications would have an entirely different set of indices appropriate to the application. Examples could include the following:

- Sources of data, e.g. *Harris, Gallup, TRW*, etc.
- Models, e.g. *Lotus, Quattro, Excel*, etc.
- Versions of certain "objects", e.g. *V1, V2, ...*
- Points in Time
- Points in space
- Coordinates in space and time
- etc.

8.2.4 The Operations

In [122] we discussed some of the considerations that underlay the way in which we set about to define an algebra for historical relations. Chief among these was the notion of "dimensional purity" for reduction operators. By this we mean that each of the dimensions of a multi-dimensional object should be accessible through its own reduction operator. The same consideration informs the shape of the algebra of **IDM**; specifically, there will be operators to access the attribute dimension, the base value dimension, and the index dimension. In the rest of this section, we will outline the basic operators in an extended algebra for the Indexical Data Model as follows:

1. the set theoretical operators: union ($\cup$), intersection ($\cap$), difference ($-$), and Cartesian Product ($\times$),
2. extensions to the traditional relational operators: project (Π), select (σ), and join ($\bowtie$), and
3. new operators: function restriction(ι), and drop index (δ)

This presentation is based upon the algebra of **HRDM** [119].

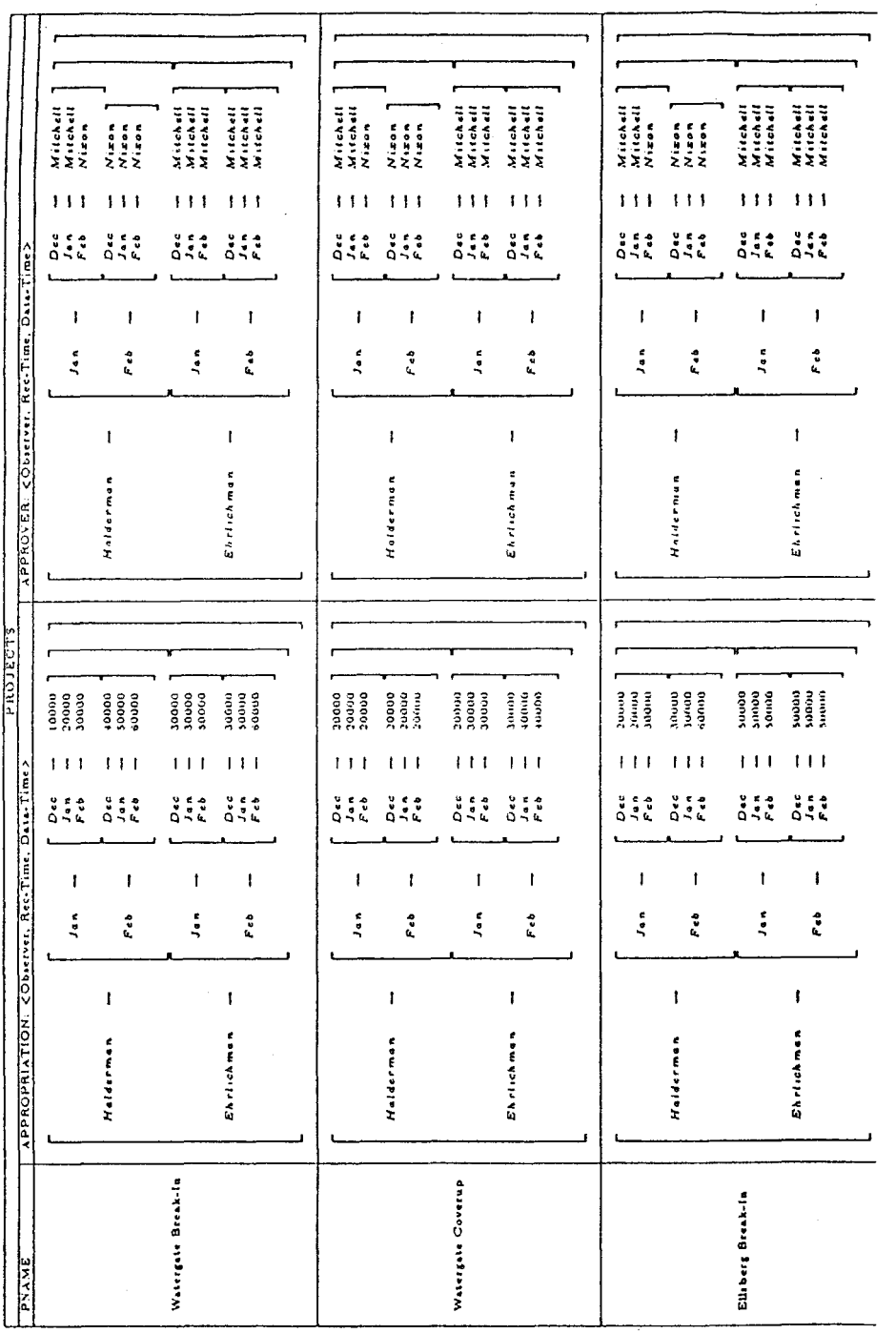


Figure 8.3: Example Indexical Database

Set Theoretic Operators

Indexical relations, like regular relations, are sets of tuples, and so the standard set-theoretic operations – $\cup$, $\cap$, $-$, and $\times$ are defined over them. However, as in the relational model, we restrict the application of these operators to **union-compatible** relations. The notion of union-compatibility must be expanded to encompass having the same number of columns over the same complex domains.

If $r1$ on $R1$ and $r2$ on $R2$ are union-compatible, then

1. $r1 \cup r2 = \{t \text{ on scheme } R3 | t \in r1 \text{ or } t \in r2\}$
   where $R3 =< A_1, K_1, VDOM_1, IDOM_1 >$
2. $r1 \cap r2 = \{t \text{ on scheme } R3 | t \in r1 \text{ and } t \in r2\}$
   where $R3 =< A_1, K_1, VDOM_1, IDOM_1 >$
3. $r1 - r2 = \{t \text{ on scheme } R1 | t \in r1 \wedge t \notin r2\}$

As pointed out in [119], the result of these operations is, unfortunately, often counterintuitive. The result, for example, of the union of two relations $r1$ and $r2$ will not automatically "merge" tuples which refer to the same object during different, but possibly overlapping, indices. Following [119], we can define three *object-based* versions of union, intersection, and difference, all of which rely on the concept of **mergeable tuples.**

Two relations r_1 and r_2 on schemes $R_1 =< A_1, K_1, VDOM_1, IDOM_1 >$ and $R_2 =< A_2, K_2, VDOM_2, IDOM_2 >$ are **merge-compatible** if and only if $A_1 = A_2, K_1 = K_2$, , $VDOM_1 = VDOM_2$, and $IDOM_1 = IDOM_2$.

Note that merge-compatibility requires that the two referenced relations have the same key, and is therefore a stronger constraint than union-compatibility.

Two tuples t_1 and t_2 on schemes $R_1 =< A_1, K_1, VDOM_1, IDOM_1 >$ and $R_2 =< A_2, K_2, VDOM_2, IDOM_2 >$ are **mergeable** if and only if

1. R_1 and R_2 are merge-compatible
2. their key value is the same over all indices, and
3. if the tuples are defined over the same *index spans*, their values must agree on these indices

Condition 2 states that the tuples have the same key value, and thus are assumed to denote the same *object.* Condition 3 states that at all indices in the intersection of the *index spans* of the two tuples, each pair of corresponding attributes have the same value. In other words, Condition 3 states that the two tuples do not contradict one another.

The merge of t_1 and t_2, $(t_1 + t_2)$ is then defined as the tuple t_3 where:

$$t_3(A) = t_1(A) \cup t_2(A) \text{ for all } A \in A_1$$

Given a tuple t and a set of tuples S, t is **matched in** S if there is some tuple $t' \in S$ such that t is mergeable with t'; otherwise t is **not matched** in S.

With these preliminary definitions we can define more semantically-based set-theoretic operations, denoted $\cup_o$, $\cap_o$, and $-_o$:

For example, if relations $r1$ on $R1$ and $r2$ on $R2$ are merge-compatible, then:

$$r1 \cup_o r2 = \{t | t \in r1 \text{ and } t \text{ is not matched in } r2 \vee$$
$$t \in r2 \text{ and } t \text{ is not matched in } r2 \vee$$
$$\exists t_1 \in r_1\ \exists t_2 \in r_2[t = t_1 + t_2]\}$$

CARTESIAN PRODUCT The Cartesian Product of two relations r and s on Schemes R and S can be defined in the usual way as:

$$r1 \times r2 = \{t \text{ on scheme } R3 | \exists t1 \in r1, \exists t2 \in r2[$$
$$\forall A \in R1[t.v(A) = t1.v(A)] \wedge$$
$$\forall A \in R2[t.v(A) = t2.v(A)]\}$$
$$\text{where} \quad R3 =< A_1 \cup A_2, K_1 \cup K_2, VDOM_1 \cup VDOM_2, IDOM_1 \cup IDOM_2 >$$

As pointed out in [119], however, we note that this definition can result in inhomogeneous tuples, because the *index spans* of the resulting tuple is not guaranteed to be uniform across the attributes. Various alternative definitions could certainly be given wherein the *index spans* of the resulting of the resulting tuple is adjusted in some way. Further research is needed to determine how best to handle the question of *homogeneity* or *inhomogeneity* in indexical databases. This and other problems with the Cartesian Product operator in temporal databases are well-known ([396]).

Extended Relational Operators

PROJECT The project operator π when applied to a relation r removes from r all but a specified set of attributes; as such it reduces a relation along the attribute dimension. It does not change the values of any of the remaining attributes, or the combinations of attribute values in the tuples of the resulting relation. Let r be a relation over the set of attributes R and $X \subset R$. Then the **projection of r onto X** is given by: $\pi_X(r) = \{t(X) | t \in r\}$

Projection Example

The query $\pi_{PROJECT}(funding)$
would yield the relation in Figure 8.4 showing which Projects were currently recorded in the database. Similarly, the query $\pi_{PROJECT,STATUS}(funding)$ would yield the relation in Figure 8.5 showing what was thought by various observers and at various times about the approval of these projects.

Note that because π does not specifically refer to the *values* in a database, it is virtually unaffected by the fact that **IDM** relations are "grouped" (see [120]), (except that complex values might "collapse").

SELECT In [119], we defined two versions of SELECT for historical databases, which can be viewed as Indexical Databases with a single index, representing the so-called "valid time" dimension of the data. We called these two operations SELECT-IF and SELECT-WHEN, and defined them as follows:

$\sigma\text{-}IF_{(A\Theta a,Q,L)}(r) = \{t \in r | Q(s \in (L \cap t.l))[t(A)s\Theta a]\}$
$\sigma\text{-}WHEN_{A\Theta a}(r) = \{t | \exists t' \in r[t.l = \{s | t'(A)(s)\Theta a\} \wedge t.v = t'.v|_{t.l}]\}$

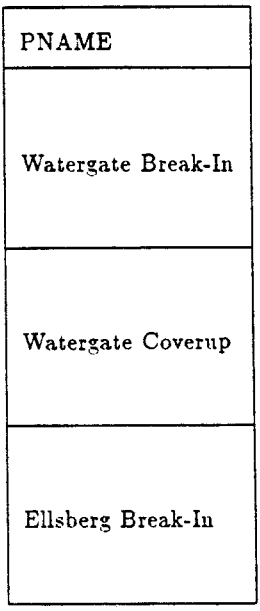


Figure 8.4: Projection Example 1

PNAME	APPROVER: <Observer, Rec-Time, Data-Time>
Watergate Break-In	[Halderman → [Jan → [Dec → Mitchell, Jan → Mitchell, Feb → Nixon], Feb → [Dec → Nixon, Jan → Nixon, Feb → Nixon]], Ehrlichman → [Jan → [Dec → Mitchell, Jan → Mitchell, Feb → Mitchell], Feb → [Dec → Mitchell, Jan → Mitchell, Feb → Mitchell]]]
Watergate Coverup	[Halderman → [Jan → [Dec → Mitchell, Jan → Mitchell, Feb → Nixon], Feb → [Dec → Nixon, Jan → Nixon, Feb → Nixon]], Ehrlichman → [Jan → [Dec → Mitchell, Jan → Mitchell, Feb → Mitchell], Feb → [Dec → Mitchell, Jan → Mitchell, Feb → Mitchell]]]
Ellsberg Break-In	[Halderman → [Jan → [Dec → Mitchell, Jan → Mitchell, Feb → Nixon], Feb → [Dec → Nixon, Jan → Nixon, Feb → Nixon]], Ehrlichman → [Jan → [Dec → Mitchell, Jan → Mitchell, Feb → Mitchell], Feb → [Dec → Mitchell, Jan → Mitchell, Feb → Mitchell]]]

Figure 8.5: Projection Example 2

We omit the definition of the analogous operators for the generalized indexical database model. Suffice it to say that analogs of both of these operators could be defined, and that they would take the following parameters:

- an attribute A
- a set of $< index, index\_value >$ pairs
- a value (i.e., a traditional, atomic value)

For example, the following query makes use of the σ-IF operator:

σ-$IF_{(APPROPRIATION=60000, \exists\ Observer, \exists\ Rec\_Time, \exists\ Data\_time\ \in\ \{Feb\})}$
(PROJECTS)

would yield a relation that shows which projects were thought, by anybody (the *Observer*), at any time (the *Rec_Time*), to have had an appropriation of 500000 sometime in February (the *Data_time*). For the example data the Watergate Breakin satisfies this query (both Halderman and Ehrlichman believed this in February) as does the Ellsberg Breakin (Halderman believed this in February). The result of this query is therefore the relation in Figure 8.6.

The σ-$WHEN$ operator is illustrated by the following query:

σ-$WHEN_{APPROVER=Nixon}(PROJECTS)$

which results in a relation showing PROJECTS ever recorded, by any observer, to have ever been approved by Nixon, and it will only show those "perceptions" about the PROJECTS.

New Operators

FUNCTION RESTRICTION This operator is intended to generalize the Time-Slice operator that has been defined in temporal database models. Function Restriction, symbolized by ι, evaluates the i-th index of an attribute A in relation r at a specified value i.

$$\iota : \mathrm{r} \times \mathrm{A} \times \mathrm{I} \times \mathrm{i}$$

In general, if r is a relation on scheme R, A is an attribute in R and I is an index of A in R, then

$$\iota_{A.I|_i}(r) = \{t | \exists\ t\prime \in r[t(R-A) = t\prime(R-A) \wedge t(A) = t\prime(A)(I)|_i]\}$$

Function Restriction Example

The following query illustrates the use of function restriction:

$$\iota_{APPROPRIATION.Rec-Time|_{Jan}}(funding)$$

This query would yield the relation in Figure 8.7, showing what was recorded about the appropriations for the projects as of January.

As is usually done with σ, we can generalize this operator to apply to multiple attributes and indices, as well as to sets of index values rather than a single value. Thus if r is a relation on scheme R, A is an attribute in R and $I_1 \ldots I_n$ are indices of A in R, then

$$\iota_{A.I_1|_{\{i_{1_1},\ldots,i_{1_m}\}},\ldots,A.I_n|_{\{i_{n_1},\ldots,i_{n_m}\}}}(r) =$$
$$\{t | \exists\ t\prime \in r[t(R-A) = t\prime(R-A) \wedge t(A) = t\prime(A)(I_1)|_{i_1} \wedge$$
$$\ldots \wedge t(A) = t\prime(A)(I_n)|_{i_n}]\}$$

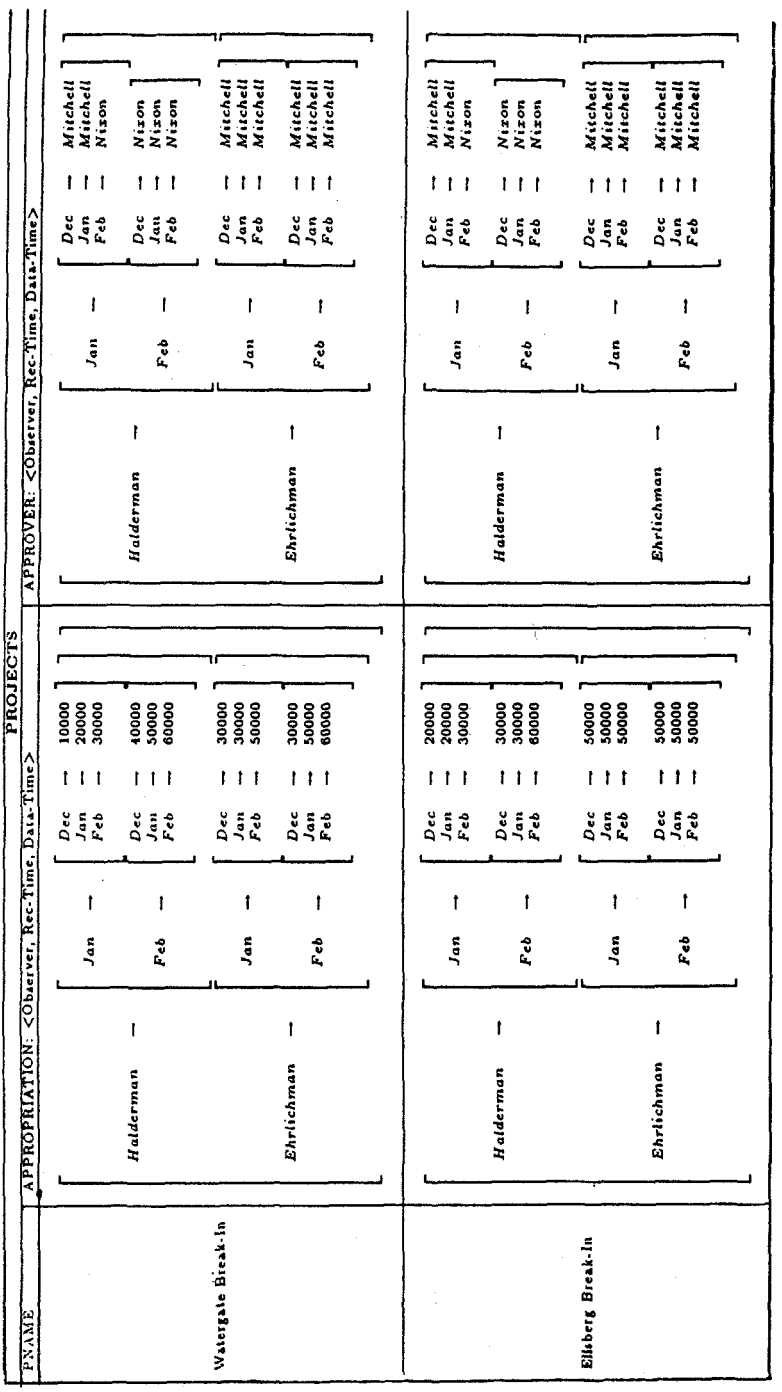


Figure 8.6: Select Example 1

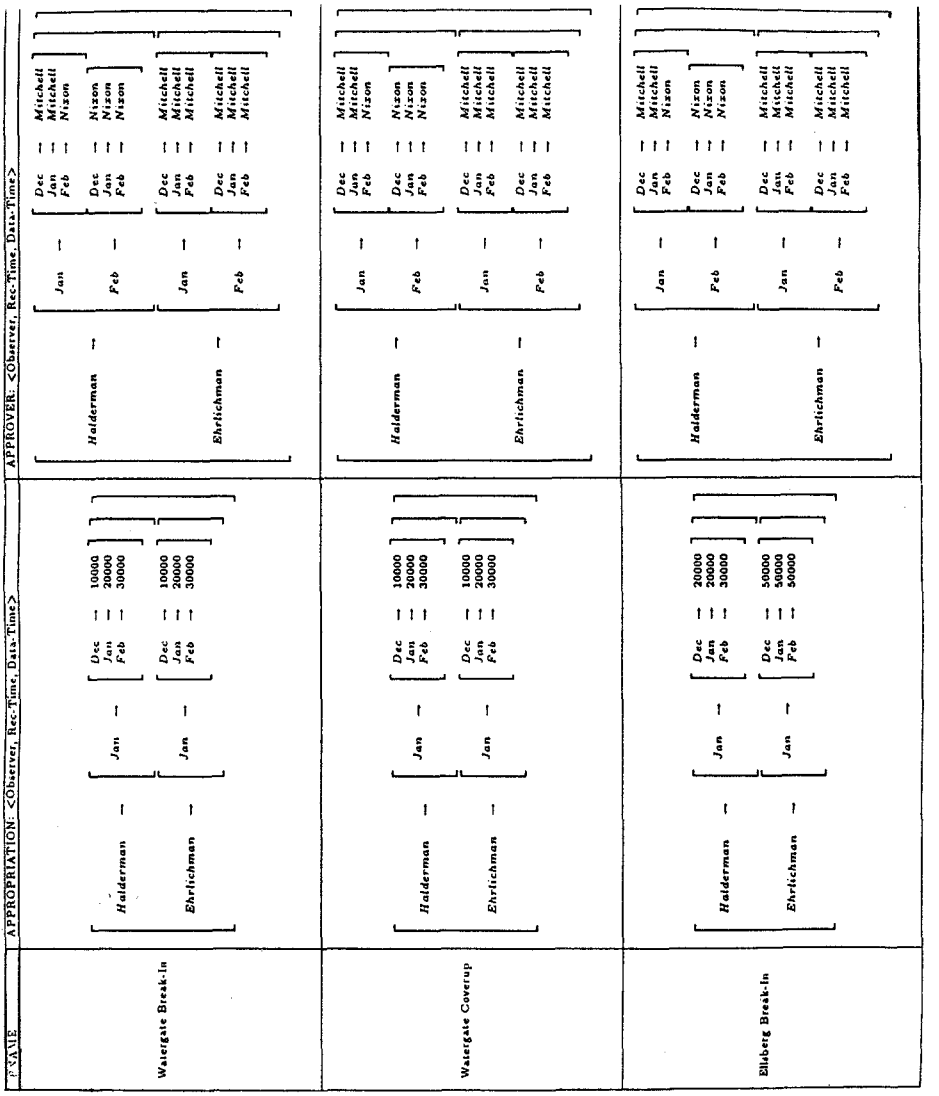


Figure 8.7: Iota Example 1

For example, Figure 8.8 show the result of the following query:

$$\iota_{APPROPRIATION.Observer|_{Halderman}}($$
$$\iota_{APPROPRIATION.Data\_Time|_{\{Jan,Feb\}}}($$
$$\iota_{<APPROVER.Observer|_{Halderman}}($$
$$\iota_{APPROVER.Data\_Time|_{\{Jan,Feb\}}}(funding))))$$

which restricts the relation to Halderman's view of things, and further only to his view of the Appropriations in January and February, and the Approvers in January and February.

DROP-INDEX Drop-Index, symbolized by δ, drops an index from a specified attribute or attributes in a relation. In effect, this operation evaluates an index at some specified value i, and therefore transforms an n-value into an n-1-value. Its analog in temporal databases is to evaluate the database as of some specified value, e.g., *now*. Note that δ, like π, changes the *scheme* of the resulting relation.

If r is a relation on scheme R, A is an attribute in R and I is an index of A in R, then

$$\delta_{A.I=i}(r) = \{t | \exists\ t\prime \in r[t(R-A) = t\prime(R-A) \wedge t\prime(A)(I) = i]\}$$

Drop-Index Example

$$\delta_{APPROPRIATION.Rec-Time=Feb}(funding)$$

would yield the relation in Figure 8.9 showing the Appropriations as they were recorded in the database as of February.

Again, extending the operator to multiple attributes and indices, Thus if r is a relation on scheme R, A is an attribute in R and $I_1 \ldots I_n$ are indices of A in R, then

$$\delta_{A.I_1=i_1,\ldots,A.I_n=i_n}(r) = \{t | \exists\ t\prime \in r[t(R-A) = t\prime(R-A) \wedge t\prime(A)(I_1) = i_1 \wedge \ldots \wedge t\prime(A)(I_n) = i_n]\}$$

The query:

$\delta_{APPROPRIATION.Rec-Time=Feb,\ APPROVER.Observer=Ehrlichman}(funding)$

would yield the relation in Figure 8.10 showing the Appropriations as they were recorded in the database as of February, and the Approvers as seen by Ehrlichman.

8.2.5 Partial Functions

In an ideal world our information is perfect, i.e., for each index (and combination of indices) associated with an attribute the database records a corresponding value. In this case there is no problem with any of the operations we would like to perform. In a database with a complex indexical structure, however, it is likely that not all of the data will be known or even existent. For example, data for certain moments in time, or representing the point of view of some particular observer, or derived using a particular model, may in certain cases be unavailable. This situation – which should be understood as essentially different from the issue of "null values" which has a long history of study in the context of the relational data model – has been recognized as a problem in the special case of historical databases. In the Historical Relational Data Model of [119] the solution of tuple "lifespans" was adopted to handle this problem. Other researchers have employed similar techniques.

Within the context of the Indexical Data Model, the problem arises when attempting to evaluate a function at a point where it is undefined. Analogous to the notion

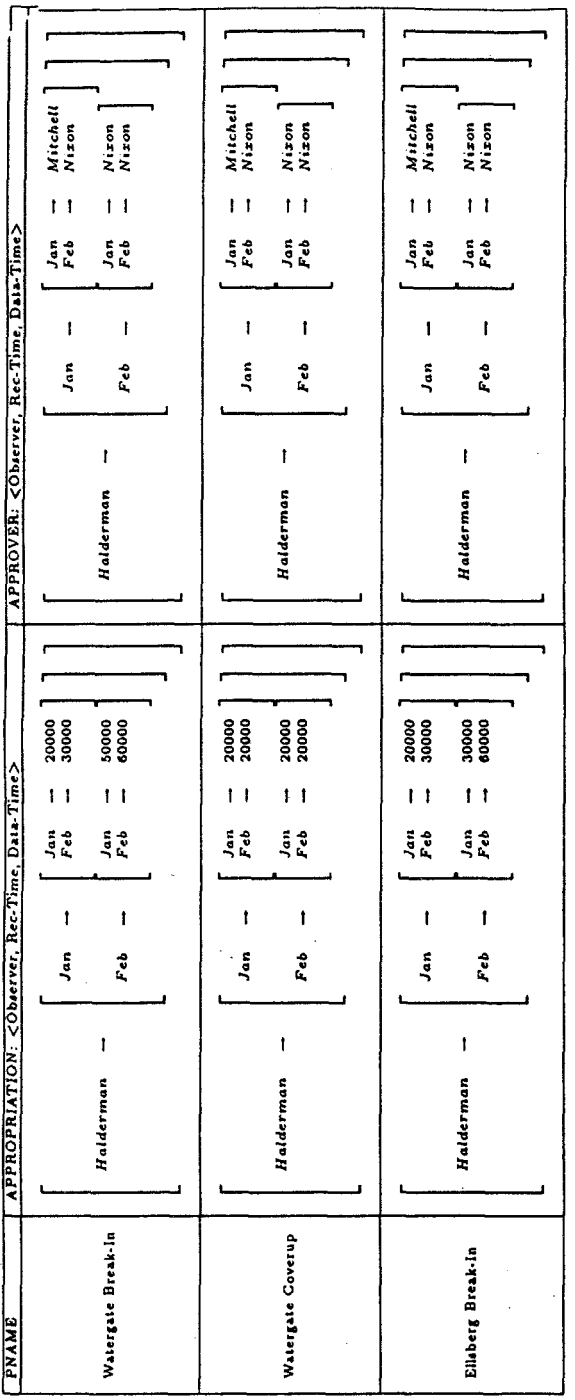


Figure 8.8: Iota Example 2

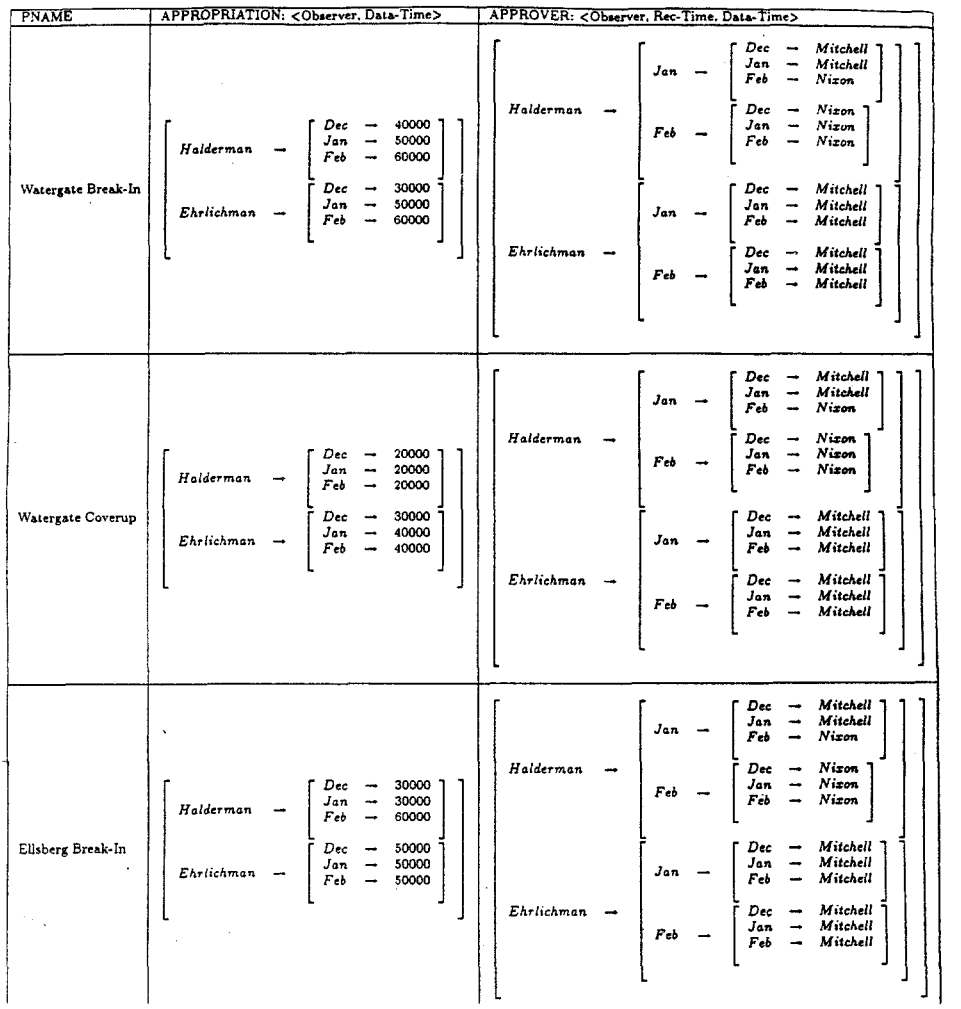


Figure 8.9: Drop-Index Example

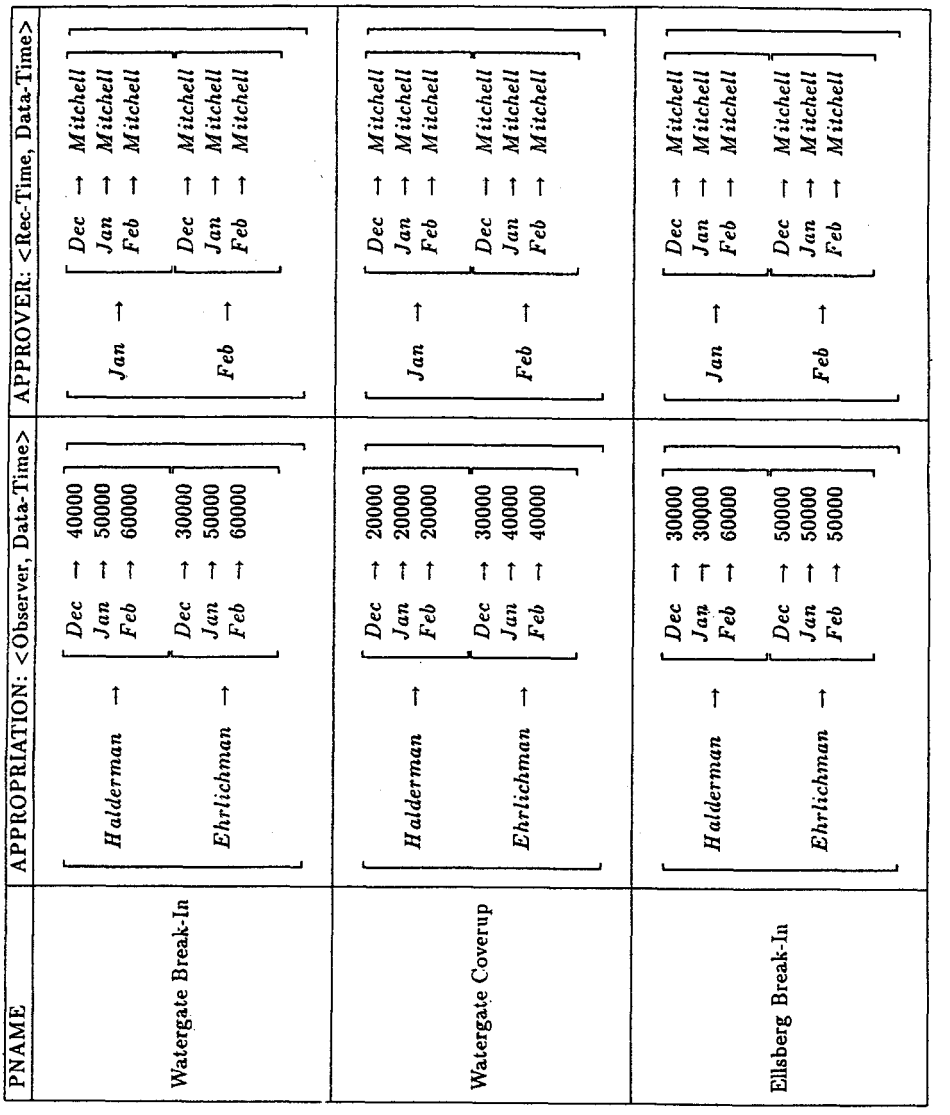


Figure 8.10: Drop-Index Example 2

of "lifespans" in **HRDM** ([119]), we can define for each Index, at either the attribute level, the tuple level, or even the relation level, the concept of *index spans* to handle this problem.

8.3 Instances of the Indexical Database Model

The Indexical Database Model is a general model which, we claim, generalizes many of the notions found in a number of other proposals which have appeared in the literature. For example, the Historical Relational Data Model (HRDM) [119] is an extension of the relational data model with a single time index to represent the so-called "valid time" of the data it manages. In IDM this would be a homogeneous indexical model with a single index. In this section we recast the model of HRDM as an indexical database model.

8.3.1 HRDM

In this section we recast the Historical Relational Data Model **HRDM** ([119]) as an instance of an IDM:

- $UD_H = \{D_1, D_2..., D_{n_d}\}$ is the (universal) set of **value domains**.
- $UA_H = \{A_1, A_2, ..., A_{n_a}\}$ is the (universal) set of **attributes**.
- $UI_H = \{I_1\} = T = \{..., t_0, t_1, ...\}$, i.e., there is only one index, the set of **times**.
- UD_H, UA_H, and UI_H are all pairwise disjoint.
- $VC = \{\Theta_{D_1}, ..., \Theta_{D_{n_d}}\}$, the set of value comparator sets, is application-dependent. (Again, minimally each value domain must support equality and inequality comparisons.
- $IC = \{\Theta_{I_1}\} = \{\Theta_T\} = \{=, \neq, <, >, \leq, \geq\}$.

A
Historical Relation Scheme R_H is a 4-tuple $R_H =< A_H, K_H, VDOM_H, IDOM_H >$ where:

1. $A_H \subset UA_H$ is the set of **attributes** of scheme R_H
2. $K_H \subset A_H$ is the **key** of scheme R_H
3. $VDOM_H : A_H \rightarrow UD_H$ gives the **value domain** of each attribute of scheme R_H.
4. $IDOM_H : A_H \rightarrow 2^{UI}$ specifies the **index domain** of each attribute A_H; in this case, for all attributes A_H of scheme R_H, $IDOM(A_H) = T$.

A **tuple t on scheme R** is an ordered pair, $t =< v, l >$, where

1. $t.l$, the **lifespan of tuple t**, is any subset of UI_H
2. $t.v$, the **value of the tuple** is a mapping such that for all attributes $A \in R$, $t.v(A)$ is a mapping of the type $t.l \rightarrow DOM(A)$.

Note that it is condition 2 in **HRDM** which stipulates that the values of each attribute in a tuple t are *partial functions*, and that the domain of these functions is the *lifespan* which is defined in condition 1.

A **relation r on R** is a finite set of tuples t on scheme R such that if t_1 and t_2 are in r, for all indices in the index spans of the two tuples, t_1 and t_2 disagree for those indices on the value of the key.

8.3.2 Bitemporal Database Models

The notion of a bitemporal database – a database which records not only the *valid time* of the data but also the *transaction time* when the data was stored in the database – is a case of an IDM with two indices. Snodgrass' Temporal Database TQuel [528], and Ariav's model [21] could equally well be cast as instances of an IDM database. We believe that the various models for version control ([310] provides a good overview) could also be expressed as specific instances of an IDM database. In [121] we discuss the issue of a uniform treatment of the valid and transaction times in bitemporal data models.

8.4 Summary and Conclusions

The modeling capability of today's DBMS's will need to be expanded if they are to serve the needs of tomorrow computing problems. Numerous research proposals have appeared to extend this functionality for a wider array of application areas. Among these proposals have been a plethora of suggestions for *historical* databases, *rollback databases*, and *bitemporal* databases.

We have described the **Indexical Database Model (IDM)** as a generalization of the work done in these and related areas. In this chapter we have provided an overview of the structures of this model and the operators in its algebra, and shown how one of these models – the historical relational data model (HRDM) – can be seen as a variety of **IDM**.

We have illustrated the power of the model by means of a few example queries expressed in an relational algebra extended to handle indexical relations. The algebra was chosen because its gives a better "flavor" of how you can *cut and paste* tables together. In fact, we can also define a multi-sorted calculus, similar to the language L_h discussed in [120], with variables over ordinary domains as well as over each type of *index*. Since L_h is shown in [120] to be more powerful than any *ungrouped* language for historical databases, it is reasonable to base our *indexical calculus* on the same framework. However, as [120] also points out, since there is as yet no known *historical algebra* equivalent in power to L_h the issue of the completeness of an *indexical algebra* remains an open one as well.

Chapter 9

A TEMPORAL QUERY LANGUAGE FOR A CONCEPTUAL MODEL

Ramez Elmasri*, Vram Kouramajian†

9.1 Introduction

This chapter is a summary of our work in temporal conceptual models and query languages [173, 178, 174, 179]. Most previous work in temporal database models and query languages has been mainly in the context of the relational model of data [528, 201, 429, 293]; and to a lesser extent, in conceptual data models [508, 178, 632]. However, these approaches have a fundamental pitfall, in that they fail to consider the semantics associated with time. In this chapter, we describe a Semantic Temporal model based on the Extended Entity-Relationship model (*STEER*), which distinguishes between conceptual and temporal objects. A *conceptual object*, once it is created, can always be referenced at any future time, whereas a *temporal object*, which we call an *entity role*, has a specific existence lifespan. For example, information concerning a *STUDENT* conceptual object can be referenced even after the student has completed his studies. However, the role of that entity as an *ENROLLED-STUDENT* has specific start and end times that define its lifespan. (*STUDENT* is the owner entity of *ENROLLED-STUDENT* role.)

The *STEER* model characterizes the properties of entities (conceptual objects), entity roles (temporal objects), and (temporal and non-temporal) attributes. It also defines temporal constraints among entity roles, differentiates between temporal and conceptual relationships, and provides rules for preserving temporal integrity constraints.

We complement our model by providing temporal query language constructs. The query language is a temporal extension of *GORDAS* [177, 176], which is a formal, high-level and user-friendly query language for the Extended Entity-Relationship model. The temporal query language distinguishes between temporal and conceptual objects/relationships. It allows selection conditions to retrieve attributes and relationships of a role or an entity type, since attributes and relationships of a role type and its owner entity type are public to each other and can be inherited. It also provides natural and high level temporal element constructor operators that simplify temporal query expressions. Finally, it supports temporal version restriction operators and allows multiple temporal scopes in a temporal projection.

*Computer Science Engineering Department, The University of Texas at Arlington, Arlington, Texas 76019–0015, U.S.A., elmasri@cse.uta.edu

†Computer Science Engineering Department, The University of Texas at Arlington, Arlington, Texas 76019–0015, U.S.A., kouramaj@cse.uta.edu

The remainder of this chapter is organized as follows. Section 9.2 describes the representation of time we use. Section 9.3 discusses the *STEER* data model. Section 9.4 presents constructs for temporal boolean expressions, temporal selection and temporal projection. Section 9.5 describes the temporal query language. Finally, section 9.6 contains a conclusion and planned future work.

9.2 Representing Time

Let T be a countably infinite set of totally ordered discrete chronons (time points), where T is denoted as $T = \{t_0, t_1, \ldots, t_{now}, t_{now+1}, \ldots\}$. We use *NULL* to represent unknown chronons, and t_{now} to represent the current chronon which is continuously increasing. We define a *time interval*, denoted by $[t_l, t_u]$, to be a set of consecutive equidistant time instants; that is, the totally ordered set $\{t_l, t_{l+1}, \ldots, t_{u-1}, t_u\} \subset T$, where t_l is the first element or start point of the time interval and t_u is the last element or end point.

The distance between two consecutive time instances, t_i and t_{i+1}, represents the *granularity* of the application; and can be adjusted to be equal to months, days, hours, minutes, seconds, or any other suitable time unit. A *single discrete time point* t is easily represented as an interval $[t, t]$, which we will denote simply as $[t]$.

Since interval representation is not closed under set operations, [201] suggested the concept of temporal element. A *temporal element* is a finite union of time intervals, denoted by $\{I_1, I_2, \ldots, I_n\}$ where I_i is an interval in T. Notice that union, intersection and difference operations on temporal elements are easily defined. In addition, set comparison predicates of two temporal elements using $=$, $\neq$, $\supseteq$, and $\subseteq$ can also defined.

In temporal databases, it is customary to include a number of different time dimensions. The most common kinds of time are: *valid time*, *transaction time* and *user-defined time* [530]. Valid time is the time that an event happened in the real world. It gives queries the capabilities to refer to past and future states of the database. The main difference between future time and historical time is that the latter refers to state changes that have already occurred, and thus are known to have happened, while future time refers to state changes that are planned to occur but may or may not happen according to the plan. Transaction time is the registration time; that is, the time when data is recorded in the database. User defined-time is provided and supported by the user of the database; its semantics are left to each application. Because of space limitations, we will consider only valid time in this chapter.

9.3 The Temporal Data Model

We will assume that the reader is familiar with the basic concepts of the *ER* model and *ER* diagrams [108], as well as its semantic extensions [175], and hence present only the main features of the *STEER* model.

9.3.1 Conceptual Objects: Entities

Our goal is to define guidelines for determining the basic aspects of an object life time. The conceptual existence of an object does not directly correspond to the birth, death, and change of the object. Objects need to be modeled in a mini-world when they become of interest. For example, employees exist in the real world as persons. However,

they become objects of interest to a company only when the company wants to hire them. At this point, the company may still want to record previous information about these persons. If an employee leaves the company, the employee remains an object of interest as long as the company still wishes.

Each *conceptual entity* e has an *existence time*, denoted by ET, which is unrelated to the concept of lifespan. The *start time point* ST of the existence time refers to the time the concept of the entity is materialized. There is no *end time point* of an existence time. The end time can be considered to be infinity in our model, because a concept (an entity) once realized never ceases to exist. The only time that characterizes an entity is the start time of its existence. Hence, $ET = [ST, \infty)$. (We also use the notation $T(e)$ to refer to the existence time of an entity e.)

There are two important ramifications in associating existence time with entities:

1. We can define and treat future planning concepts using similar mechanisms to those used for historical concepts.
2. We can enhance the power of query languages and simplify their constructs while dealing with conceptual objects, by using start time point of existence time as the earliest possible time the entity can be referenced.

An *entity type* is a set of entities of the same type; that is, entities that share the same properties. An entity type is diagrammatically represented by a rectangular box (see Figure 9.1).

9.3.2 Temporal Objects: Roles

Entities describe one aspect of the real world, the conceptual one. The other aspect is captured by temporal objects. The classification of objects as temporal and conceptual gives our model the capability to faithfully represent the way people perceive the real world. Temporal objects materialize the active role that conceptual objects play in the temporal dimension.

We call a temporal object an *entity role*, since it represents the time that the entity is participating in that role. A *role type* is a set of entity roles of the same type; that is, roles that share the same properties. Each role type is associated with a single entity type called its *owner entity*. Hence, $owner(entity\ role) = entity \Leftrightarrow role(entity) = entity\ role$. A role type is diagrammatically represented by a dotted rectangular box, and connected to an owner entity (see Figure 9.1). Each entity role ro of a role type RO is associated with a temporal element $T(ro) \subset [t_0, \infty)$ which gives the lifespan LS of the role.

The following general set of rules must hold on roles:

1. Start time of the lifespan of an entity role must be greater or equal to the start time of the existence time of the (conceptual) owner entity. This implies a top-down approach in creation of role types; that is, before a role is created its corresponding (owner) entity must exist.
2. A role type is restricted exactly to one owner entity type.
3. A role type can have only temporal attributes.
4. (Temporal) attributes of a role type are *public* to the owner entity type; that is, an owner entity refers to these attributes as though they are attributes of the owner entity.

5. Similarly, (temporal and non-temporal) attributes of an entity type are public to all associated role types.
6. A role can access all relationship instances for relationship types in which the owner entity participates.
7. Similarly, an entity can access all relationship instances for relationship types in which the associated role participates.

9.3.3 Temporal Constraints among Roles

Our model enforces two temporal constraints among roles:

1. *Existence Constraint*: A sup-existence/sub-existence constraint, denoted by RO_i/RO_j, holds between two role types RO_i and RO_j iff the following holds: $\{\forall\ ro_{jk} \in RO_j, \exists\ ro_{il} \in RO_i\ such\ that\ ro_{jk} \equiv ro_{il}\}$; that is, every entity role RO_j represents the same entity role in RO_i. The existence constraint implies a top-down approach in the creation of roles. A member role of a sub-existence represents the same real world entity as some member of the sup-existence. An entity role cannot exist in the database merely by being a member of a sub-existence; it must be also a member of the sup-existence.
2. *Lifespan Constraint*: A sup-lifespan/sub-lifespan constraint, denoted by RO_i/RO_j, holds between two role types RO_i and RO_j iff the lifespan of any entity role $ro_{jk} \in RO_j$ is a subset of the lifespan of the entity role $ro_{il} \in RO_i$ where $ro_{jk} \equiv ro_{il}$; that is, $T(ro_{jk}) \subseteq T(ro_{il})$. Notice that the lifespan constraint implies the existence constraint, but not vice versa.

9.3.4 Non-Temporal Attributes

Attributes are properties of objects. Non-temporal attributes can be only properties of conceptual entity types but not of role types. The value of a non-temporal attribute of an entity holds over the entire existence time of the entity. We assume that the reader is familiar with the properties of non-temporal attributes of the *ER* model [175], and discuss only the properties of temporal attributes below.

9.3.5 Temporal Attributes

Each entity type E_i (role type RO_i) may have a set of basic temporal attributes TA_{i1}, TA_{i2}, ..., TA_{in}, and each temporal attribute TA_{ij} is associated with a domain of values $dom(TA_{ij})$. For example, a temporal attributes of the *PERSON* entity type is *Name*,, and a non-temporal attribute is *SSN* (see Figure 9.1).

The following definitions are similar to those given in [178]. For roles, the temporal value of each attribute TA_i of ro, which we refer to as $TA_i(ro)$, is a partial function $TA_i(ro) : T(ro) \rightarrow dom(TA_i)$. The subset of $T(ro)$ in which $TA_i(ro)$ is defined is denoted by $T(TA_i(ro))$. It is assumed that TA_i has a $NULL$ (or $UNKNOWN$) value during the intervals $T(ro) - T(TA_i(ro))$.

In the case of entities, the temporal value of each attribute TA_i of e, which we refer to as $TA_i(e)$, is a partial function $TA_i(e) : ET(e) \rightarrow dom(TA_i)$. The subset of $ET(e)$ in which $TA_i(e)$ is defined is denoted by $T(TA_i(e))$. It is assumed that TA_i has a $NULL$ (or $UNKNOWN$) value during the intervals $T(e) - T(TA_i(e))$.

The partial function that describes the values of a temporal attribute is also called a *temporal assignment* [201, 179]. The subset of chronons during which a temporal attribute is defined is called the *temporal element of the temporal assignment*.

Several types of temporal attributes exist:

1. A *temporal single-valued attributes* has at most a single atomic value for each entity (role) at each time instant [t].
2. A *temporal multi-valued attribute* can have more than one value for an entity (a role) at a given time instant [t]; hence, its domain is the power set $P(V)$ of some simple domain V.
3. A *temporal composite attribute* is a list of several component temporal attributes, and its value for each entity at time instant [t] is a concatenation of the values of its components. The temporal element of a temporal assignment of a composite attribute is the union of the temporal elements of the temporal assignments of its components.

In our model, each entity will be associated with a system-defined non-temporal *SURROGATE* attribute whose value is unique for every entity in the database. The value of this attribute is not visible to users, and is never altered.

9.3.6 Classes and Superclass/Subclass Relationships

Our data model supports the concepts of (conceptual) subclasses and superclasses and their related concepts of specialization and generalization. A *class* is any set of entities; hence, an entity type is also a class. Additional groupings of entities that are *subclasses* (subsets) of the entities in another class are often needed. A superclass/subclass relationship is implicitly defined for each subclass. Subclasses can be used to represent generalization and specialization hierarchies and lattices. A more complete discussion of subclasses in the *EER* model is given in [175].

A member entity of a subclass represents the same real world entity as some member of the superclass. An entity cannot exist in the database merely by being a member of a subclass; it must also be a member of the superclass. An entity that is a member of a subclass will have the same existence time as the corresponding entity in its superclass because the entity in the subclass represents the same real world entity as the one in the superclass. An important concept associated with the subclass is that of attribute inheritance. An entity that is a member of a subclass inherits all (temporal and non-temporal) attributes of the corresponding entity in its superclass. Note that since (temporal) attributes and relationships of a role type are public to an owner entity type, they become public to all subclasses of the owner superclass; and thus any query over a subclass can access those public (temporal) attributes. An entity also inherits all relationship instances for relationship types in which the superclass participates.

9.3.7 Conceptual Relationships

A *conceptual relationship type* R of degree n has n participating entity types $E_1, E_2, \ldots, E_n$. Each *relationship instance* r in R is an n-tuple $r =< e_1, e_2, \ldots, e_n >$ where each $e_i \in E_i$. Each relationship instance r in R has an existence time ET. The start time of the existence time of a relationship instance must be greater or equal to the start time of the existence time of each of the participating entities; that is, $ST(r) \geq ST(e_i)$ for each $e_i \in E_i$ $(i = 1, 2, \ldots, n)$.

9.3.8 Temporal Relationships

Our model supports temporal relationships. A *temporal relationship type* TR of degree n has n participating entity types (role types) $O_1, O_2, \ldots, O_n$ where O_i is either an entity type ($O_i \equiv E_i$) or a role type ($O_i \equiv RO_i$). Each *temporal relationship instance* tr in TR is an n-tuple $tr =< o_1, o_2, \ldots, o_n >$ where o_i is either an entity ($o_i \equiv e_i$, $e_i \in E_i$) or an entity role ($o_i \equiv ro_i$, $ro_i \in RO_i$).

Each temporal relationship instance tr in TR is associated with a temporal element $T(tr)$ which gives the lifespan of the temporal relationship instance. If all participating objects are entity roles, then the lifespan of the temporal relationship instance must be a subset of the intersection of the lifespans of the roles; and if all participating objects are entities, then the start time of the lifespan of the temporal relationship instance must be greater or equal to the start times of all existence times of the entities.

9.3.9 Temporal Constraints among Relationships

Our model enforces two constraints on temporal and conceptual relationships:

1. *R-existence Constraint*: A sup-R-existence/sub-R-existence constraint, denoted by R/TR, holds between a conceptual relationship R and a temporal relationship TR where all participating object types are role types iff $\forall\ tr_i = (ro_1, ro_2, \ldots, ro_n) \in TR$ the following two conditions must be satisfied:

   (a) $\exists\ r_i = (e_1, e_2, \ldots, e_n) \in R$, such that $owner(ro_j) = e_j$, for $j = 1, 2, \ldots, n$.

   (b) The start time of the lifespan of the temporal relationship instance tr_i must be greater or equal to the start time of the existence time of the conceptual relationship r_i.

2. *R-lifespan (time order) Constraint*: A sup-R-lifespan/sub-R-lifespan constraint, denoted by TR/R holds between a temporal relationship TR and a conceptual relationship R where all participating object types are role types iff $\forall\ r_i = (e_1, e_2, \ldots, e_n) \in R$ the following two conditions must be satisfied:

   (a) $\exists\ tr_i = (ro_1, ro_2, \ldots, ro_n) \in TR$, such that $owner(ro_j) = e_j$ for all $j = 1, 2, \ldots, n$.

   (b) The start time of the existence time of the conceptual relationship instance r_i must be greater or equal to the start time of the lifespan of the temporal relationship tr_i.

The R-existence and R-lifespan constraints are denoted diagrammatically in a similar way to existence and lifespan constraints for role types. Notice that the R-lifespan constraint is, in some sense, the reverse constraint of the lifespan constraint on role types. It is used to model the cases where a conceptual relationship cannot exist until after a temporal relationship has started. For example, students cannot get transcript entry for courses until *after* they have been enrolled.

9.3.10 An Example

Consider the example database schema in Figure 9.1, which describes a simplified organization for part of a *UNIVERSITY* database. The database includes the (conceptual) entity types *PERSON*, *STUDENT*, *FACULTY*, *COURSE*, and *SECTION*. Any entity

instance that is a member of any of these entity types is associated with an existence time. The entity types *STUDENT* and *FACULTY* are subtypes of the entity type *PERSON*. The role types are diagrammatically represented by a dotted rectangular box, and connected to their owner entity types. The role types and their owner entities are:

owner(LIVING-PERSON) = PERSON
owner(ENROLLED-STUDENT) = STUDENT
owner(CURRENT-FACULTY) = FACULTY
owner(VALID-COURSE) = COURSE
owner(ACTIVE-SECTION) = SECTION

The conceptual relationship types are:

CS between *COURSE* and *SECTION*
TAUGHT between *FACULTY* and *SECTION*
TRANSCRIPT between *STUDENT* and *SECTION*

The temporal relationship types are:

ACTIVE-CS between *VALID-COURSE* and *ACTIVE-SECTION*
IS-TEACHING between *CURRENT-FACULTY* and *ACTIVE-SECTION*
ENROLLED between *ENROLLED-STUDENT* and *ACTIVE-SECTION*

9.4 Temporal Query Language Constructs

In *non-temporal* databases, a typical query will select certain entities based on boolean predicates that involve attribute values of an entity (and of related entities). Following that, certain attributes or relationships of each of the selected entities are displayed. Other queries involve aggregate functions on groups of entities or their attributes. In a *temporal database*, selection criteria may be based not only on attribute values but also on temporal conditions. In addition, once an entity is selected, the user may be interested in displaying the complete history of some of its attributes or relationships, or to limit the displayed values to a certain time interval. To allow for temporal constructs in queries, we will use the concepts of temporal boolean expressions, temporal selection conditions (or temporal predicates), and temporal projection [179].

A *(temporal) boolean expression* is a conditional expression on the attributes and relationships of an entity (or an entity role). For example, a boolean expression can be *Classification = 'Senior'*. The boolean condition when applied to one entity e (or one entity role ro), evaluates to a function from $T(e)$ (or $T(ro)$) to { *TRUE, FALSE, UNKNOWN* }. We call this function a temporal assignment.

The *true_time* of a boolean expression, c, denoted by $[\![c]\!]$, evaluates to a temporal element for each entity e (or each entity role ro). The temporal element is the time for which the condition is *TRUE* for e (or ro). As an example, the boolean condition *Classification = 'Senior'*, when applied to an *ENROLLED-STUDENT* ro (Figure 9.1), returns a function from $T(ro)$ to { *TRUE, FALSE, UNKNOWN* }. If $T(ro)$ is equal to [9/1/83, 8/31/87], and the student classification was senior during [9/1/86, 8/31/87], the temporal assignment result would be:

$$\{\ [9/1/83,\ 8/31/86] \rightarrow FALSE,\ [9/1/86,\ 8/31/87] \rightarrow TRUE\ \}.$$

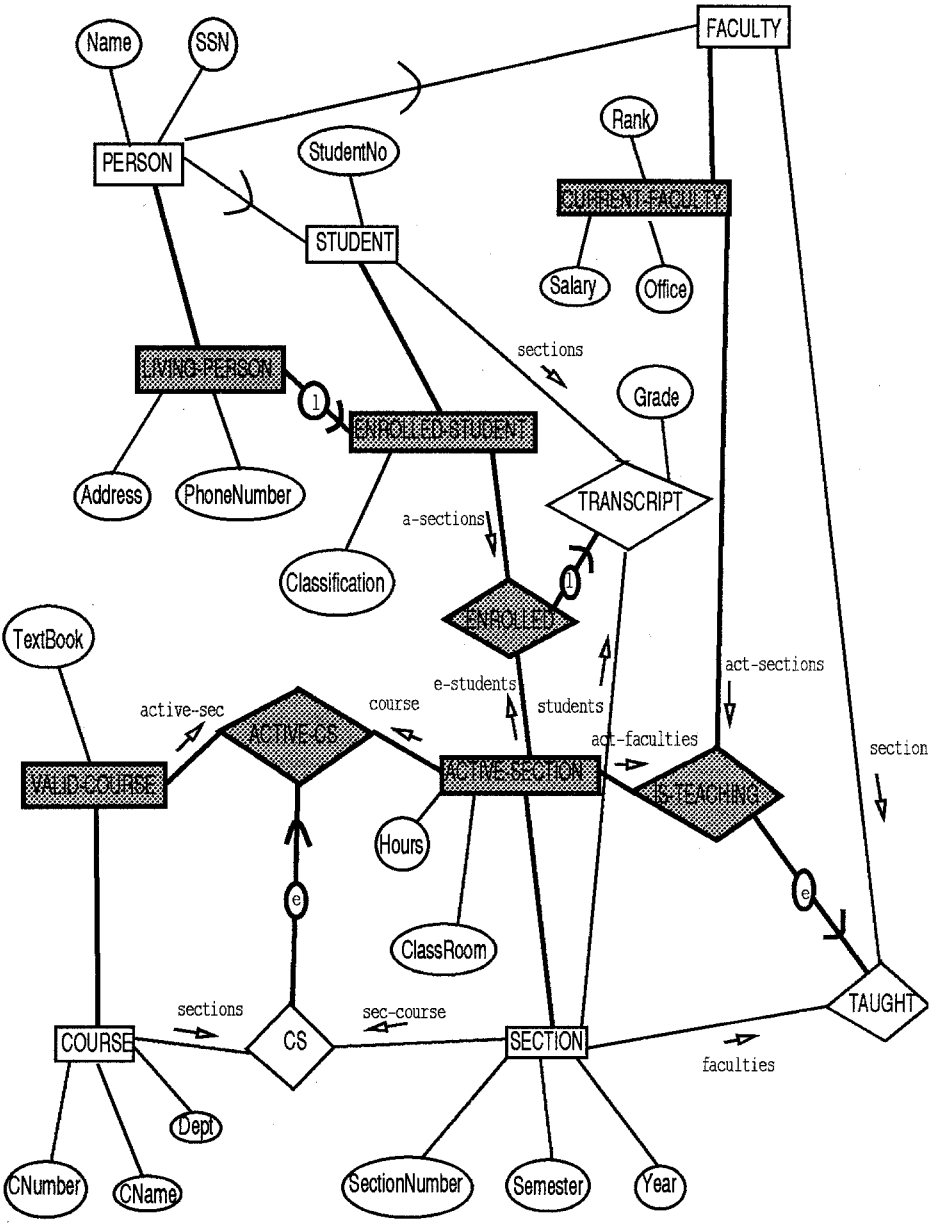


Figure 9.1: A Temporal EER Schema for part of a University Database

The value for the true_time $[\![$ *Classification = 'Senior'* $]\!]$ would be [9/1/86, 8/31/87].

Next we define a *(temporal) selection condition*, which compares two temporal elements using the set comparison operators $=$, $\neq$, $\supseteq$, and $\subseteq$. When applied to an entity type (or class), it evaluates to those entities that satisfy the temporal selection condition. For example, consider the following temporal selection condition applied to the *ENROLLED-STUDENT* entity role type of Figure 9.1:

$$[\![\text{ Classification } = \text{ 'Senior' }]\!] \supseteq [9/1/86,\ 5/31/87]$$

This selects all *ENROLLED-STUDENT* entity roles whose classification was *'Senior'* during the period [9/1/86, 5/31/87]. The condition is evaluated for each *ENROLLED-STUDENT* entity role individually, and returns either a *YES* or *NO* answer. All entity roles for which the answer is *YES* are selected.

We also define *temporal projection*. This is applied to a temporal entity and restricts all temporal assignments (attributes and relationships) for that entity to a specific time period specified by a temporal element T.

Temporal selection conditions are used to select particular entities based on temporal conditions, whereas temporal projections are used to limit the data displayed for the selected entities to specific time periods. Temporal boolean conditions may be used as components in the expressions for both temporal selections and temporal projections.

9.5 The Temporal Query Language

Much of the flexibility and power provided by a query language is dependent on the data model. Our temporal query language derives its simplicity and expressiveness from the *STEER* data model; in particular from the distinction between temporal and conceptual objects, and temporal and conceptual relationships. The query language used is a temporal extension of *GORDAS* [177, 176, 179]. We briefly recall that *GORDAS* is a functional query language with two clauses: *GET* and *WHERE*. The *WHERE*–clause specifies conditions for the selection of entities from a *root entity type*, while the *GET*–clause specifies the information to be retrieved for each selected entity. For example, consider the following (non-temporal) *GORDAS* query specified on the database of Figure 9.1:

Q1: **GET** < Name, SSN, < CName **of** sec-course, Semester, Year >
of sections > **of** STUDENT
WHERE Address **of** STUDENT = 'Arlington'

Here, the root entity type, specified at the end of the *GET*–clause, is *STUDENT*. The *WHERE*–clause is evaluated individually for each entity in the root entity type, and selects each entity that satisfies the *WHERE*–clause. In this query, each *STUDENT* entity who lives in *'Arlington'* is selected. (Note that the *Address* attribute is visible to *STUDENT* by being inherited from *LIVING-PERSON* via *PERSON* as we will describe in section 9.5.1.) The *of STUDENT* in the *WHERE*–clause is optional, and can be left out. For each selected entity, the *GET*–clause retrieves the student *Name*, *SSN* (both inherited from *PERSON*) and *sections*, and for each of the student's sections the *CName*, *Semester* and *Year* are retrieved. The connection names such as *sec-course* and *sections* are used to specify related entities of the root entity type in a

functional way as though they were attributes of root entities. Hence, the path *sections of STUDENT* specifies the *SECTION* entities related to each *STUDENT* entity via the *TRANSCRIPT* relationship. A full discussion of non-temporal *GORDAS* is outside the scope of this work, and is given in [177, 176].

In temporal *GORDAS*, we will adopt the philosophy that a non-temporal *GORDAS* query is also valid, and will default to the current database state. Hence, if a temporal *GORDAS* query is specified with no temporal selections or projections, we will assume that a snapshot of the database is taken at the time instant t_{now} when the query is evaluated, and the query is processed using this database state. This will make it possible to specify both temporal and non-temporal queries on the database within the same framework.

In section 9.5.1 we discuss temporal projection and introduce additional temporal element constructor operators. Section 9.5.2 presents temporal selection. Section 9.5.3 describes temporal version restriction operators. Finally, section 9.5.4 introduces operators that allow multiple temporal scopes in a temporal projection.

9.5.1 Temporal Projection

A temporal query may involve a temporal selection condition or a temporal projection condition or both. The general philosophy of *GORDAS* is to maintain a clean separation between the specification of conditions for selection of entities (in the *WHERE*–clause) and the specification of information to be displayed (in the *GET*–clause). To maintain this philosophy, we will specify a temporal projection on the data to be displayed at the end of the *GET*–clause, as in [179]. For example, consider the query to retrieve the history of the *Address* and *PhoneNumber* of *'John Smith'* during the period 1985 to 1990:

Q2: **GET** < Address, PhoneNumber > **of** PERSON : [1/1/1985, 12/31/1990]
WHERE Name = 'John Smith'

The term *PERSON*: [1/1/1985, 12/31/1990] at the end of the *GET*–clause specifies that the temporal assignment for *'John Smith'* is to be retrieved during the period [1/1/1985, 12/31/1990]. On the other hand, the next query is non-temporal, and displays the current (at time instant t_{now}) *Address* and *PhoneNumber* of *'John Smith'*:

Q3: **GET** < Address, PhoneNumber > **of** PERSON
WHERE Name = 'John Smith'

As seen from query *Q2*, the temporal projection of selected entities is specified by a temporal element at the end of the *GET*–clause. The temporal element may be a time period (as in *Q2*) or may itself be derived from the database for each entity (as in *Q4* below). For example, suppose we want the full history of the *Address* and *PhoneNumber* of *'John Smith'*:

Q4: **GET** < Address, PhoneNumber > **of** PERSON : ET
WHERE Name = 'John Smith'

This retrieves the values of address and phone number over the whole existence time (*ET*) of the entity. If *:ET* is left out, only the current *Address* and *PhoneNumber* (at time instant t_{now}) are retrieved.

Temporal attributes of a role type are *public* to the owner entity type; that is, an owner entity can refer to these attributes (through inheritance) as though they are attributes of the owner entity. Similarly, (temporal and non-temporal) attributes of an entity type are *public* to all associated role types. The definition of attributes and relationships of a role type and its owner entity type as public to each other gives queries the flexibility to specify selection conditions and to retrieve information involving attributes of a role or an entity type by referring to each other's attributes. For example, in queries *Q2*, *Q3* and *Q4*, the entity *PERSON* is able to refer to the attributes *Address* and *PhoneNumber* of the entity role *LIVING–PERSON* since the owner of *LIVING–PERSON* is the entity *PERSON*. Notice that we can specify similar queries to the queries *Q2*, *Q3* and *Q4* by referring to *LIVING–PERSON* explicitly, as in *Q5*, *Q6* and *Q7*, since they only display temporal attributes:

Q5: **GET** < Address, PhoneNumber > **of** LIVING-PERSON
: [1/1/1985, 12/31/1990]
WHERE Name = 'John Smith'

Q6: **GET** < Address, PhoneNumber > **of** LIVING-PERSON
WHERE Name = 'John Smith'

Q7: **GET** < Address, PhoneNumber > **of** LIVING-PERSON : LS
WHERE Name = 'John Smith'

However, *Q6* and *Q7* will only retrieve entities that are *LIVING–PERSON*s at time t_{now}, whereas *Q3* and *Q4* may retrieve deceased persons (since conceptual entities have no end time) but then find that their attributes may be *NULL* at time t_{now}.

The projection of (temporal) attributes over a lifespan displays information about a conceptual entity during the time period it participates as a particular entity role. For example, in the next query, the history of the *Address* and *PhoneNumber* of *'John Smith'* is retrieved, during the time he was an enrolled student:

Q8: **GET** < Address, PhoneNumber > **of** ENROLLED–STUDENT : LS
WHERE Name = 'John Smith'

Here, the *Address* and *PhoneNumber* history are retrieved only during the lifespan (*LS*) that *'John Smith'* exists in the *ENROLLED–STUDENT* entity role. If *:LS* is left out, the current *Address* and *PhoneNumber* are retrieved if end time $ET(LS) \geq t_{now}$; if $ET(LS) < t_{now}$, the entity will not be selected since it is not valid as an *ENROLLED–STUDENT* any more.

The next query retrieves all sections that *'John Smith'* has completed:

Q9: **GET** < CName of sec–course, Semester, Year > **of** SECTION
WHERE Name **of** students **of** SECTION $\supseteq$ {'John Smith'}

In this query, there is no need to project the query result over a time period since the attributes *Semester* and *Year*, and the relationship *CS* (specified by *sec–course*) are non-temporal attributes and relationship of *SECTION*, and hence always exist. It is this type of query that becomes cumbersome to specify when no distinction is made between temporal and conceptual objects, as in [179]. For instance, if the root entity *SECTION* of query *Q9* is replaced by *ACTIVE–SECTION*, we get all sections that *'John Smith'* is currently enrolled in:

Q10: **GET** < CName of sec–course, Semester, Year > **of** ACTIVE–SECTION
WHERE Name **of** e–students **of** ACTIVE–SECTION $\supseteq$ {'John Smith'}

This query is implicitly temporal since it refers to the temporal entity role *ACTIVE–SECTION*. The query displays the current (at time instant t_{now}) sections that *'John Smith'* is enrolled in. The capability to express such temporal queries by referring to an entity role without explicit reference to time is one of the advantages of this model.

A temporal query may explicitly specify a temporal projection that is derived from a temporal boolean expression. For example, suppose we want the history of *Name*, *Office* and *Salary* of each *CURRENT–FACULTY* entity role only when the value of its attribute *Rank* was either *'Assistant Prof'* or *'Associate Prof'*:

Q11: **GET** < Name, Office, Salary > **of** CURRENT–FACULTY
: [[(Rank = 'Assistant Prof') **OR** (Rank = 'Associate Prof')]]

In this case, a different time projection is applied to each selected entity role based upon the time that entity was an assistant or associate professor; that is, the time restriction is correlated to each individual entity role.

When we deal with temporal intervals and elements in *STEER*, we need additional functionalities that are not needed in other temporal query languages [178]. For instance, [[*entity* : *ET*]] – [[*role* : *LS*]] returns the time period (temporal element) when an entity does not participate in a specific role. Hence, to retrieve the *Name*, *SSN*, and *Salary* of each faculty during the time period she/he is not *CURRENT–FACULTY* (e.g. on sabbatical or working for industry), we write:

Q12: **GET** < Name, SSN, Salary > **of** FACULTY
: [[FACULTY : ET]] – [[CURRENT–FACULTY : LS]]

Here, the *Name*, *SSN*, and *Salary* of a faculty are retrieved only during the period [[*FACULTY : ET*]] – [[*CURRENT–FACULTY : LS*]], which is different for each selected entity. Note the difference between the temporal expression in queries *Q11* and *Q12*. In both queries *Q11* and *Q12*, temporal element constructor operators are used to define temporal elements at the end of the *GET*–clause. However, in query *Q11*, the boolean condition *c = ((Rank = 'Assistant Prof')* **OR** *(Rank = 'Associate Prof'))* is based on a boolean predicate that involves attribute values of an entity role, whereas in query *Q12*, the boolean condition refers only to the existence time of *FACULTY* and the lifespan of *CURRENT–FACULTY*. In query *Q11*, the temporal element at the end of the *GET*–clause is the true_time of a boolean condition, whereas in query *Q12*, the temporal element is the difference between two true_times, namely the existence time of a *FACULTY* entity and its lifespan as a *CURRENT–FACULTY* entity role.

The next query retrieves the history of the *Name*, *Address* and *PhoneNumber* of living persons during the period they were not enrolled students:

Q13: **GET** < Name, Address, PhoneNumber > **of** PERSON
: [[LIVING–PERSON : LS]] – [[ENROLLED–STUDENT : LS]]

The usual set theoretic operations of *UNION*, *INTERSECTION*, *DIFFERENCE* and *COMPLEMENT* can be combined with temporal element constructor operators. Both previous queries *Q12* and *Q13* use the *DIFFERENCE* operator. The next query uses the *COMPLEMENT* operator to retrieve the history of the *Name*, *Address* and *PhoneNumber* of persons before they become faculty members:

Q14: **GET** < Name, Address, PhoneNumber > **of** PERSON
: **COMPLEMENT** [[FACULTY : ET]]

The idea of applying a temporal boolean condition to entity roles and entities can be extended to temporal attributes. The true_time of a boolean condition reduced to a temporal attribute name is represented as [[*temporal_attribute : time_period*]]. This corresponds to the true_time of the *temporal_attribute* during *time_period*. For example, the next query retrieves the history of the *Name*, *StudentNo*, *CName*, *Semester* and *Year* of enrolled students during the period they had a valid *Classification* (that is, a *Classification* value that is not *NULL*):

Q15: **GET** < Name, StudentNo, < CName **of** sec-course, Semester, Year >
of sections > **of** ENROLLED-STUDENT : [[Classification : LS]]

9.5.2 Temporal Selection

Next, consider the specification of temporal conditions to select entities. These will usually involve the specification of temporal selection predicates in the *WHERE*-clause. For example, consider the query to retrieve the *Name* and *PhoneNumber* of all persons who lived in *'Arlington'* on 3/30/1992:

Q16: **GET** < Name, PhoneNumber > **of** LIVING-PERSON : [3/30/1992]
WHERE [[Address = 'Arlington']] $\supseteq$ [3/30/1992]

In query *Q16*, the *WHERE*-clause is a temporal selection condition. For each *LIVING-PERSON* entity role, it first calculates the temporal boolean expression c = *(Address = 'Arlington')*; if the true_time $[\![c]\!] \supseteq$ [3/30/1992], the temporal selection condition evaluates to *YES* and the *LIVING-PERSON* entity role is selected by the *WHERE*-clause. Note that it is still necessary to specify the temporal projection [3/30/1992] again in the *GET*-clause since leaving it out would retrieve the current *Name* and *PhoneNumber* of each selected entity rather than those on 3/30/1992.

The next query retrieves the *SectionNumber* and *ClassRoom* of all active sections that were held in room *'EB119'* during the period 1990–1991:

Q17: **GET** < SectionNumber, ClassRoom > **of** ACTIVE-SECTION
WHERE ([[ClassRoom = 'EB119']] $\cap$ [1/1/1990, 1/12/1991]) $\neq \emptyset$

When we deal with time periods, we sometimes need to access the first and last time points of temporal elements. For example, to retrieve the *Name*, *SSN* and *Address* of all current students who lived in *'Arlington'* when they first enrolled as a student, we write:

Q18: **GET** < Name, SSN, Address > **of** ENROLLED-STUDENT
WHERE [[Address = 'Arlington']] $\supseteq$ ST(LS)

Here, the temporal selection condition evaluates to *TRUE* if $[\![c]\!] \supseteq ST(LS)$, where c = *(Address = 'Arlington')*. The term $ST(LS)$ means the start time point of a lifespan. Note that $ST(LS)$ is implicitly applied to *ENROLLED-STUDENT* since it is the root entity role. This can also be written as *ST([[ENROLLED-STUDENT : LS]])*.

The lifespan of an entity role can be a continuous time period. This may happen if either an entity role has come into existence in the mini-world and never ceased to exist, or an entity role has come into existence for a while then has ceased to exist and has never reexisted in the mini-world. In order to support the concept of continuous and discontinued lifespans in our query language, we introduce the keywords *CONTINUOUS* and *DISCONTINUED*. For example, suppose we want to display the courses that have been continuously taught every semester:

Q19: **GET** < Cname, CNumber, Dept > **of** VALID-COURSE
WHERE CONTINUOUS LS

This is similar to the temporal *ALWAYS SINCE* operator in temporal logic [542].

As a final example, note that a name related with any lifespan besides the root entity must be explicitly specified in a temporal query. For instance, the next query explicitly specifies the lifespan of attribute *Address* in the *WHERE*-clause , and retrieves the *Name*, *SSN* and *Address* of all current students whose initial *Address* value was *'Arlington'*:

Q20: **GET** < Name, SSN, Address > **of** ENROLLED-STUDENT
WHERE [[Address = 'Arlington']] $\supseteq$ ST([[Address : LS]])

9.5.3 Temporal Version Restriction Operators

In the *STEER* data model, the complete history of an entity (or an entity role) is kept. The temporal versions of an entity (or an entity role) are ordered and queries may be restricted to specific versions of an entity (or an entity role). A temporal version restriction operator may be specified in the *GET* or *WHERE* clause of temporal *GORDAS* queries. The syntax of our version restriction operator is:

$$:([\![NAME]\!] : INTERVAL < INDEX >)$$

where the term [[*NAME*]]: is optional and the term $INTERVAL < INDEX >$ is required. The term [[*NAME*]] is a true_time, where *NAME* may be either a boolean condition, or may be reduced to an entity name, an entity role name, or a temporal attribute. The term $INTERVAL < INDEX >$ indicates a projection either over a single interval if $< INDEX >$ is an integer or over a range of intervals if $< INDEX >$ is an integer range. (Note that we assume that the intervals of a temporal element are disjoint and in the canonical temporal element representation.) As an example, the version restriction operator *:(INTERVAL 1)*, when applied to a *CURRENT-FACULTY* entity role *ro* (Figure 9.1) restricts the temporal element to the first interval of its lifespan. In this case, the term [[*NAME*]]: is not used in the version restriction operator *:(INTERVAL 1)*. However, if the term [[*NAME*]]: is used in the version restriction operator such as *:([[Address]] : INTERVAL 1)*, then when it is applied to a *CURRENT-FACULTY* entity role *ro* (Figure 9.1) it restricts the temporal element to the first interval of the lifespan of attribute *Address*.

The next query retrieves the *Name* and the first three *Salary* values for each faculty:

Q21: **GET** < Name, Salary : (**INTERVAL** 1 to 3) > **of** FACULTY

The term *:(INTERVAL 1 to 3)* in the *GET*–clause specifies that the projection displays the first three *Salary* values for each *FACULTY*. Notice that once a temporal version restriction operator appears in either the *GET* or *WHERE* clause of a query, we immediately deal with the full temporal entity in that clause, rather than the current entity version only.

Temporal operators may be nested and are evaluated from *left to right*. For example, suppose we want to display the *Name*, *SSN* and the current *Address* for each person whose first *Address* was *'Houston'* and third *Address* was *'Arlington'*:

Q22: **GET** < Name, SSN, Address > **of** PERSON
WHERE (Address : (**INTERVAL** 1) = 'Houston') **AND**
(Address : (**INTERVAL** 3) = 'Arlington')

The term *Address : (INTERVAL 1) = 'Houston'* in the *WHERE*–clause means that we first apply the temporal ordering restriction operator *:(INTERVAL 1)* and then compare it with *= 'Houston'*. Similarly, the term *Address : (INTERVAL 3) = 'Arlington'* in the *WHERE*–clause means that we first apply the temporal ordering restriction operator *:(INTERVAL 3)* and then compare it with *= 'Arlington'*.

As seen from queries *Q21* and *Q22*, if the term $[\![NAME]\!]$: is omitted from the version restriction operators, then the term $INTERVAL < INDEX >$ is applied to the specific attribute. However, if we would like to display the *Name* and *PhoneNumber* of a person during the time period she/he first lived in *'Arlington'*, we could write:

Q23: **GET** < Name, PhoneNumber : ([[Address = 'Arlington']]
: **INTERVAL** 1) > **of** PERSON

In this case, the true_time of the boolean expression *c = (Address = 'Arlington')* is evaluated for each entity and then the temporal element is assigned to the first interval of each true_time. Note that the projection over *PhoneNumber* may result with multiple values. However, we could even further restrict the previous query, *Q23*, by displaying only the first value of the *PhoneNumber*:

Q24: **GET** < Name, PhoneNumber : ([[Address = 'Arlington']]
: **INTERVAL** 1) : (**INTERVAL** 1) > **of** PERSON

Temporal version restriction operators are not limited to attributes; they may be applied to entities and therefore restrict queries to a specific range of lifespans. For example, the next query displays the *Name*, *SSN*, *Address*, *PhoneNumber*, *CName*, *Semester*, *Year* during the second interval of the lifespan of each *ENROLLED–STUDENT* who currently lives in *'Arlington'*:

Q25: **GET** < Name, SSN, Address, PhoneNumber, < CName **of** course,
Semester, Year > **of** a–sections > **of** ENROLLED–STUDENT
: (**INTERVAL** 2)
WHERE Address = 'Arlington'

As a final example, note that any restriction condition specified on an entity is applied before any other restriction operator is applied to its attributes. Hence, if we would like to display for current full professors, their *Name*, and the initial *Salary* as associate professors, we could write:

Q26: **GET** < Name, Salary : (INTERVAL 1) > **of** CURRENT–FACULTY
: [[Rank = 'Associate Prof']]
p **WHERE** Rank = 'Full Prof'

9.5.4 Temporal Scope Operators

In the *GORDAS* language, one can reference the attributes of an entity related to the root entity by using a connection name. In the temporal *GORDAS*, related entities must be projected over the temporal elements of connection names. To generalize our temporal projection capabilities, we introduce the scope operator, denoted by *SCOPE*, which overwrites the temporal projection of a root entity (or related entities). For example, if we would like to retrieve the *Name* and *Rank* attribute values of each current faculty during their $LAST - 1$ interval but we would like to retrieve their initial *Salary*, we could write:

Q27: **GET** < Name, Rank, Salary : **SCOPE**(**INTERVAL** 1) >
of CURRENT-FACULTY : (**INTERVAL** LAST - 1)

In this case, the *SCOPE* operator at the end of *Salary* attribute overwrites the temporal projection at the end of the *GET*-clause.

9.6 Conclusions

This chapter was a summary of our work in temporal conceptual models and query languages [173, 178, 174, 179]. Our model distinguishes between conceptual and temporal objects, and characterizes the properties of entities (conceptual objects), entity roles (temporal objects), and (temporal and non-temporal) attributes. It also defines temporal constraints among entity roles, differentiates between temporal and conceptual relationships, and provides rules for preserving temporal integrity constraints.

The query language is a temporal extension of *GORDAS* [177, 176, 178]. The temporal query language derives its power from the distinction between temporal and conceptual objects/relationships. It provides natural and high level temporal element constructor operators that simplify temporal query expressions. These operators utilize the entity existence times and the role lifespans in query formulations, without having to refer explicitly to time values. They also use the concepts of boolean conditions, true_times, and temporal projections [178].

Our query language allows temporal element constructor operators to be defined over entities, entity roles and temporal attributes. It supports temporal version restriction operators and allows multiple temporal scopes in a temporal projection. In addition, the concept of *CONTINUOUS* and *DISCONTINUED* temporal elements can be used to specify conditions such as *ALWAYS* and *SOMETIME* from temporal logic.

Chapter 10

A Data Model for Time-Series Analysis

Arie Segev *, Rakesh Chandra†

10.1 Introduction

Researchers in the field of survey statistics often deal with observations of individual units at a single point in time. This type of data is called cross-sectional data. On the other hand, in the field of econometrics, researchers use time series data. Time series data are series of observations of a single unit over several points in time. Often data analysts come across data that is a combination of cross-sectional and time-series data. This type of data is known as pooled data [159].

For example, pooled data would refer to any database describing every individual of a group across a sequence of time periods. In the domain of finance, the description of a market database is an example of pooled data. This database may contain the end-of-day closing prices of all securities traded on the stock exchange. Like cross-sectional data, this database contains observations on several securities and like time series data, it contains observations over different periods of time. Thus, pooled data is important to a data analyst because it contains both intertemporal dynamics as well as individual information.

The broad goals of a data analyst studying pooled data are:

1. Study of an individual data unit over time
2. Study of samples of data units by summarizing data and drawing inferences from these summary statistics.

Analysis of pooled data presents unique problems. The main problem encountered when analyzing time series data is the autocorrelation of error terms, while non-constant variance is a problem when analyzing cross-sectional data. In pooled data the analyst faces these problems simultaneously. In addition, there is the problem of finding correlations between cross-sectional disturbances of different individual units. [328] has also pointed out the following difficulty. The relationship between dependent and independent variables may be different for different individuals. It has also been observed that the regression coefficients in time series equations change over time (either systematically or randomly). A proposed data model for pooled data must give analysts the capability of handling these errors.

*Walter A. Haas School of Business, University of California at Berkeley, and Information and Computing Sciences Division, Lawrence Berkeley Laboratory Berkeley, CA 94720. email: segev@csr.lbl.gov, crakesh@csr.lbl.gov

†Walter A. Haas School of Business, University of California at Berkeley, CA 94720

In this chapter, we propose a data model for time-series and pooled data analysis. This model incorporates features for temporal and statistical data management. In addition, the model provides an interactive environment for data analysis and a query language that incorporates frequently used statistical operators. We also discuss data models for temporal and statistical databases proposed in the literature and explain why no existing data model completely captures the complexity of time series and pooled data.

10.2 Main Features of The Data Model

The main features of the data model, developed for the analysis of pooled and time-series data, are briefly discussed below. Each point is discussed in detail later with relevant examples. The data model for pooled data features:

1. A variable vector based data model that conforms to the way data analysts visualize their problem. Data need not be fitted in the rigid framework of rows and columns. Instead data should be represented as vectors, with individual data items in a time series conforming to a user specified frequency. Each vector has a function that specifies the value of the time series for time points that haven't been recorded and some user-defined information associated with it, e.g., lifespan, granularity.

2. Collections of these vectors are organized into semantic units termed *Concepts*. Different Concepts can be brought together to form a hierarchy of Concepts. Descendants of a Concept in the hierarchy would inherit the user-defined information from their ancestors. Descendants would also be allowed to override the inherited information with user-specified values. (We note that the word *Concept* is used in the singular and plural form throughout the text.)

3. Rules necessary to maintain semantic information, constraints and to trigger actions based on conditions.

4. Built-in calendars which implicitly understands the date associated with each data item in a time series and the capability of defining other calendars. Data manipulation routines work in close association with the calendar.

5. Uniform treatment of the different notions of time. One notion of time is a linearly ordered sequence of points or intervals along the time line. Another notion is the set of topological relationships between events (event X before event Y or version 1, version 2 etc.). Both these notions of time are manipulated in a uniform manner.

   The data model also supports two different timelines. Each element of a time sequence is associated with a valid time and a transaction time. Valid time is defined in [294] as "the time when the fact is true in modeled reality". Valid time is used in *Event Construction* (explained later in the chapter). Transaction time [294] of a database fact is the time when the fact is stored in the database. Transaction time is used in creating a *Concept_History* and to allow the database to rollback to a previous state (also discussed later in the chapter).

6. Version Management capability to account for data and forecast revisions.

7. The capability of transforming data of a particular frequency to a different frequency.

8. The feature that constructs pooled data from time-series data. This is called Event Construction. Different time-series can be "joined" together to create a combination of cross-sectional and time-series data.
9. Sophisticated interpretation and handling of missing values in the data.
10. An interactive environment which gives the data analyst the following features:
    - freedom to create and store temporary data sets as versions of the original dataset.
    - ability to store the steps used for data analysis
    - ability to backtrack during the analysis phase without much overhead
    - fast and efficient concurrent data access
    - a sophisticated graphical or icon-based user-interface.
11. A database query language which supports operators used frequently in data analysis and metadata manipulation.
12. An interface to external mathematical and statistical routines.

The following section presents data models for temporal and statistical databases that have been proposed in the literature. We discuss models which support a subset of the features mentioned above. We also explain why no existing data model completely captures the complexity of time series and pooled data and the rationale for proposing this data model.

10.2.1 Relevant Research

[508] define "Time Sequences" as "sequence of values in the time domain for a single entity instance". This definition accurately describes time-series data. Their representation of a time sequence as $< s, (t, a)^* >$ is similar to the variable vector representation for time-series data that is proposed in this chapter. Here s is the surrogate, t is a time point or interval and a is an attribute or vector of attributes. $(t, a)^*$ represents the variable vector for the surrogate s. Their data model, TDM, also defines the *type* of a time sequence as the information that determines the "value of the time sequence for time points that do not have explicit data values."

A relational representation of TDM [509] includes the "family" construct as a collection of First Normal Form (FNF) temporal and non-temporal relations with the same surrogate type. The following section, discusses how the notion of Concepts is an extension of the idea of "families" and illustrates its usefulness in representing time-series and pooled data.

Clifford uses a Non-First Normal Form(NFNF) to represent temporal relations [122]. Their representation suppresses surrogate repetition and lists the time-value pairs of each temporal attribute in successive tuples. This is similar to the vector representation of time-series data, proposed in this chapter.

The design and implementation of rules in databases were discussed in [554]. Temporal Rules are discussed in [183] and [566]. In this chapter, rules are used to trigger transactions (updates, modifications, numerical computations) based on clock triggered events and/or database changes. The following sections describe the utility of rules in data analysis, Concept Materialization and maintaining database consistency.

Temporal data models assume the association of "transaction time" or "valid time" [530] with every tuple in a temporal relation. The proposed model associates every

time-series with a "Calendar" and every element of the time-series with parts of that Calendar. Each element in the time-series would have a transaction time - the time it enters the database and a valid time - the time that reflects when the fact actually occurred. This is illustrated in the following section. Coupling data manipulation routines to the appropriate calendar is necessary and could possibly require the knowledge of the relationship between two different calendars. Implementation of multiple calendars in a temporal database have been described in [535].

Version Management and Change databases have been described in [428]. Though temporal databases and change databases have traditionally been considered separately in the literature, we adopt the viewpoint of [632]. They define a datatype that "carries the most generic semantics of time". This datatype is used to develop a uniform treatment of time.

Operations common in Relational Databases like Join and Select [126] have been extended to deal with temporal data, e.g., [119], [528]. In addition, researchers have extended the standard query languages (SQL, QUEL) to include temporal operators [528] [488] [427]. [428] have described additional temporal operators like Time-Slice and Moving Window. Aggregate and statistical operators have been described in [531]. These extensions to the standard query languages are further enhanced by including sampling operators [436], event construction (similar to the event-join described in [507]) and operators to handle missing values and transformations (changing from one frequency to another). In addition, the query language should also be used to create complex operators based on pre-defined and built-in operators (in the sense of abstract data types [552]) and have an interface to mathematical and statistical routines written in other languages.

Data analysis is similar to engineering design where users have private copies of data, which is analyzed externally. Analysts typically create different versions of the data, making their own conclusions and storing them. [311] and [97] have described a database environment for engineering analysis. We adopt their ideas to the domain of complex data analysis. Data analysis should also be tightly coupled with a sophisticated graphical user interface. [10] describe a graphical interface for temporal summary management and [89] describe a graphical system for statistical databases. We build on this research by defining a graphical user interface which includes the data manipulation operators described below.

Even though the above research contains many elements necessary in our work, no data model captures the entire scope of features required for pooled data representation and manipulation. Furthermore, no data model for temporal databases captures the essential features necessary for statistical analysis of time-series data. Models for statistical and scientific data management do not incorporate the environment required for data analysis and provide weak support for temporal data management.

Time-series and pooled data analysis is an integral part of empirical finance, marketing, economics and scientific research. Thus, there is an imperative need to design and implement a database model which incorporates the entire complement of features discussed above. In the following sections, the important aspects of the proposed data model are discussed in greater detail.

10.3 Vector Based Data Model

Each time series is essentially an n-ary vector and is associated with a set of user-defined information. This information (M) is classified into (a) information that must

be present with every time-series (M_s) (if not supplied by the user, appropriate defaults are used) and (b) Information optionally supplied by the user (M_u). M_s consists of:

1. Name : The identifier of the time series to be used in data retrieval and data manipulation routines.
2. Calendar/Granularity : a set of pre-defined time points. This item specifies the calendar with which the time-series is associated. For example, the time-series IBM-DAILY-CLOSING would be associated with the calendar AMERICAN-BUSINESS-DAYS. This means that on every day in the calendar AMERICAN-BUSINESS-DAYS, the time-series should have a value. Granularity is a specification of the points in time in the defined calendar that can potentially have data values [508]. The defined calendar will thus determine the granularity of the time-series. The advantage of associating a time-series with a calendar is that there is no need to physically store the individual time points with the values of the time-series. When the time-series is retrieved due to a query, the individual time points can be generated using the specification of the calendar. This is especially advantageous for time-series with large lifespans. Since the individual time points are not saved on disk, there are large savings in disk space utilization. Thus, all time points of the time-series are physically stored only when the calendar cannot be pre-defined. This is possible in the case of randomly updated time-series like tick-by-tick stock prices.
3. Exception-Set: is a set of time points (within the calendar) on which values of the time-series are not recorded. For example, even though IBM-DAILY-CLOSING should be recorded on every day in the calendar AMERICAN-BUSINESS-DAYS, there may be an important announcement on a particular day that stops trading in the stock. Thus the value of the time-series is not recorded on that day. The exception-set will include such time points. Thus, the actual calendar for a time-series is the set difference of *Calendar* and *Exception-Set.*
4. Lifespan : This indicates the start time and end time of the time-series. The end time can be specified to be ∞. The lifespan is used in conjunction with the calendar and exception-set to generate the set of time points for which the time-series has values.
5. Update Mode : This indicates whether the time-series is derived from another time-series(s) or is base data. If the series is derived, the rule for update is specified here. Time-series are allowed to have a hybrid update mode. For example, a time-series recording the value of an option will change whenever the price of the underlying
6. Frequency : This specifies the frequency with which the time-series is updated. The time of update refers to the valid time. Valid time is defined in [294] as "the time when the fact is true in modeled reality". Frequency is always specified with respect to the calendar with which the time-series is associated and may be a non-trivial function on the set of time points in this calendar. For example, suppose EMP, a time-series which records the level of employment in the country, has the Calendar/Granularity: "the last day of the month unless the day is a holiday in which case it is the preceding business day". The frequency of EMP would be *monthly.* If a time-series is derived from other time-series, the frequency would be the frequency of the base data or some function of it. For example, consider the time-series DJIA and DJIAHILO. DJIA, the Dow Jones Industrial Average, is a weighted average of the price of a given set of stocks. It is computed every

time the price of a component stock changes. Thus, it is a derived time-series with the same frequency of update as the set of underlying stocks. DJIAHILO, is a time-series that contains the daily high and low values of the DJIA. Thus, DJIAHILO has a daily frequency which is different from the frequency of DJIA, even though DJIAHILO is derived from DJIA.

It is important to stress the difference between frequency and granularity. A time-series is said to be "regular" [508], If it contains a value for each time point in the time-series lifespan. In a regular time-series, the granularity is the same as the frequency. In this case, the exception-set is a null set.

7. Time-series type [508]: The type of a time-series determines the value of the time-series at time points where the value isn't explicitly specified. It falls into one of the following categories:

   - Step-wise constant: if (T_i, A_i) and (T_k, A_k) are two consecutive pairs in the time-series such that $T_i < T_k$, then $A_j = A_i$ for $T_i \leq T_j < T_k$.
   - Continuous : a continuous function is assumed between (T_i, A_i) and (T_k, A_i) which assigns A_j to T_j, $T_i \leq T_j < T_k$ based on a curve-fitting function.
   - Discrete : each value (A_i) is the time-series is not related to other values. Consequently missing values cannot be interpolated. We take the interpretation of not available for any analysis based on these values.
   - User defined type : missing values in the time-series can be computed based on a user defined interpolation function.

An example of a one-dimensional time-series vector with the associated user-defined information is shown in Table 1.

The table shows observations of a country's Gross National Product (GNP) and its associated user-defined information. The calendar associated with GNP is a function of "American Business Days" calendar. GNP is not derived from any other time-series and thus its update mode is "Base Data". The frequency of update is specified as quarterly and reflects the dollar value of the sum total of economic activity in the quarter. The frequency of the time-series refers to the valid time of update. The transaction time of the update is "the last day of the month succeeding the month in which the quarter ends." For example, in Table 1 an entry for GNP with the valid time of update as March 31^{st} (end of first quarter) has transaction time of update as April 30^{th} (the month succeeding the first quarter). The type of the time-series GNP is user-defined. This means that user-defined functions will be used to determine the value of GNP at time points where it has not been explicitly recorded. For example, the GNP on April 30^{th} (valid time) is not recorded in the time-series. This could be derived by a function which uses the previous values of GNP as parameters or through a function which uses other economic indicators. Rules (discussed below) can be used to define the type of a time-series and build in the desired level of complexity. The item "Forecast Source" is part of the user-defined information, M_u, optionally supplied by the user. It is a list of sources used to obtain forecasts for the time-series GNP. Table 1 lists these sources as the Wall Street Journal (WSJ) and the Federal Reserve Board (FRB).

Now consider the case of a 2-ary vector such as the Dollar price on the Foreign Currency exchange market.

The two dimensions of the 2-ary vector are bid-price and ask-price, which represent traders' spreads in foreign currency markets. In this example, the transaction time of

GNP in Trillions		
valid time	trans time	value
03/31/91	04/30/91	1.6
06/30/91	07/31/91	1.5
09/30/91	10/31/91	1.4
.	.	.
.	.	.
.	.	.

M	
M_s	
Name	GNP
Calendar/Granularity	AMERICAN-BUSINESS-DAYS
Update Mode	Base Data
Frequency	Last day of quarter. If holiday, then next business day
Type	User-Defined
Lifespan	start- 1923 ; end - ∞
M_u	
Forecast-Source	WSJ; FRB

Table 10.1: GNP Time-series

an element in the time-series is the same as the valid time of that element. Since the time-series is updated every time there is a trade, e.g., buy/sell of the currency, the frequency is specified to be "on-event". The type of the time-series is Step-Wise Constant since the price remains the same between events and every time there is an event, the price is updated. For time-series Ask-price and Bid-price, the M_s part of the user-defined information is the same and thus they can form a 2-ary vector.

The main advantage of the vector representation is that it reflects the way data analysts perceive data. It also facilitates direct translation from vector manipulation routines to actual database operations.

The representation of time-series vectors is different from the way time sequences are represented in [508]. There, time sequences are defined as sequences of values in the time domain for a single entity instance. Thus, time sequences are of the form $< S, (A,T)^* >$ where S is the surrogate, A the attribute and T the time point associated with A. In the proposed model, a time-series is in the form $< (A,T_s)^* >$. The name of the time-series is part of the user-defined information associated with it.

In the case of a 2-ary time-series, the representation is $< (A_1, A_2, T_s)^* >$, where A_1 and A_2 are the two attributes which have the same M_s. Note that M_u can be different for components of an n-ary vector. However, two vectors with different frequency cannot be dimensions of a 2-ary vector. For example, if the frequency of GNP is quarterly and the frequency of EMP is monthly, they must be defined as separate 1-ary vectors. Since there is a need to group these vectors in spite of the difference in frequency, the idea of Concepts is introduced in the next section.

10.4 Concepts

Ideas similar to Concepts have been discussed previously in the literature. [509] define a "family" as a collection of First-Normal Form (FNF) temporal and non-temporal relations that have the same surrogate type. Concepts can be thought of as a collection of relations but the definition presented below also allows the construction of a complex hierarchy of Concepts with inheritance. Thus, Concepts can be considered an object-oriented extension of the idea of "families".

More formally, a Concept can be defined as a collection of n-ary vectors grouped together to represent a semantic unit. These Concepts are called *Basic Concepts*. Basic Concepts are Time-Series also and can be grouped together in a hierarchy to form other Concepts. These Concepts are called *Complex Concepts*, and a hierarchy of Complex Concepts is a *concept tree*. This is illustrated in Figure 1.

Figure 1 shows the Complex Concept ECONOMIC-ACTIVITY. This is a collection of Concepts describing different economic activity. The solid links in the hierarchy describe an "is-a" link while the dotted links describe a "derived-by" link or a dependency between two time-series. Concepts are shown in ovals while time-series vectors are shown in rectangles.

The Concept ECONOMIC-ACTIVITY is described by the Concept US-INSTRUMENTS, a collection of financial investment instruments, and USA-ECON-IND, a collection of indicators of economic activity. The time-series GNP, INF and EMP are the economic indicators recorded for USA-ECON-IND. The Concepts DERIVATIVES and EQUITIES are types of the Concept US-INSTRUMENTS. Concepts OPTIONS and FUTURES are types of the Concept DERIVATIVES while STOCKA and STOCKB form the Basic Concept EQUITIES. Time-series vectors STOCKA and STOCKB are time series which record the price of the respective stocks

over time. These time-series also form the Basic Concept NYSE-STOCKS. As shown by Figure 1, the data model allows a time-series vector to be a part of one or more Concepts. The time-series STOCKA-MAR93 records the price of a European option on STOCKA which expires in March 1993. This time-series is a derived time-series since its value at any point in time is a function of several variables including the value of the time-series STOCKA. STOCKB-JAN93 is a similar time-series. The time-series STOCKA-MAR93 and STOCKB-JAN93 form the Basic Concept OPTIONS. As in the case of time-series vectors, user-defined information (M_u) is also associated with Concepts. Concepts in the concept-tree hierarchy inherit M_u from their ancestors. Only the Name and Lifespan elements of M_s are specified for Concepts. The Update Mode, Type, Calendar and Frequency are not specified since the time-series that constitute the Basic Concepts in the concept tree may have different M_s and the semantics of inheritance are ambiguous.

Part of the example in Figure 1 is used to illustrate the idea of Concepts and describe the data model in greater detail.

There are several economic indicators watched by economists to track the state of the economy. As shown in Figure 1, among these economic indicators are the monthly inflation figures (INF), the monthly employment figures (EMP) and the quarterly GNP numbers. If the nations' Economic Indicators were to be recorded over time, the individual indicators must be recorded separately and there must be a way to group them together.

In the proposed data model, individual indicators (INF, EMP and GNP) are represented as sequences of values in the time domain (since the observations of each indicator over time is a unary vector). These time series are then grouped together by the Concept USA-ECON-IND. The linkage between time-series GNP, INF and EMP to Concept USA-ECON-IND is shown in Figure 1.

Economists could use USA-ECON-IND to see how the economy has performed over a period of time. A snapshot of the economy would be given by the *Current_Value* (*CV*), which is a cross-section of the economic indicators today. Each Concept also has a *Most_Recent_Value* (*MRV*). MRV is a cross-section of the most recent values of each time-series vector in the Concept. In the context of the example, CV is determined by using the MRV and the type of the time-series. However, in general, CV need not be a function of MRV. It is possible for the CV to be derived from values of several periods in the past. For example, a time-series with type "Step-Wise Constant" will have the same MRV and CV. However, a time-series whose type is determined by a rule may have a different CV than the MRV. When any of the components of the Concept change (a more recent observation is added), the MRV of the Concept changes. At these time points the MRV $\equiv$ CV. But at any other point in the future the CV may be different from the MRV.

Logically, the Concept is an extension of a relational database view which joins all the component vectors of the Concept. To define USA-ECON-IND, it would seem that GNP, EMP and INF should be joined. But, economic indicators are not always recorded at the same points in time. For example, the GNP of the United States is reported every quarter while figures for employment and inflation are reported monthly. This is in contrast to interest rates which are recorded daily and the buy/sell price of the US Dollar which is recorded almost every second !

The presence of time in the vectors and the fact that the time-series may be associated with different calendars and have different frequencies based on these calendars complicates the join. We refer to this type of join as an *Event Construction* and it is discussed later in the chapter. Concepts can be difficult or impossible to express in a

temporal relational language like TQUEL [528]. This is discussed in a later section.

Table 2(a) shows the component time-series of the Concept "USA-ECON-IND" and the MRV. The type of INF is defined by a rule. This rule states that the CV is the sum of the MRV and the average inflation over the last twelve periods multiplied by the difference between the present time and the valid time of the MRV. This is illustrated in Table 2(b). Note that $INF(-i)$ in Table 2(b) refers to the inflation in the i^{th} period before the current one. Table 2(c) shows the Concept "USA-ECON-IND" as a time-series. The time-series is derived by Event-Construction.

Forecasts and predictions must also be accommodated by the data model. Economists make predictions/forecasts of several economic indicators and collect forecasts from several different sources. It is useful to assess the accuracy of these predictions/forecasts as time-series reveal themselves over time.

Time-series are represented as $< (A,T)^* >$ where A is the value of the time-series at a particular point in time, T. If A is considered a vector of the actual value (A_a) and the forecasted value (A_f) or a set of forecasted values (from different sources), the forecasts/predictions can be stored. The accuracy of these forecasts can be evaluated when the actual values are revealed. For example, consider the case where there is just one source of forecast for the GNP. At any point of time the GNP time-series would have values of A_a and A_f for all time points from [time-start,valid-time-of-most-recent-update] and A_f from [valid-time-of-next-update,time-end]. Here $time-end$ is the time till which forecasts are provided. There are three constraints on the forecast and actual values. These constraints are necessary because the model supports only transaction time and valid time.

1. A_f, transaction-time $< A_a$, valid time. The forecast must be entered before the actual event occurs in the modeled reality.
2. A_f, transaction-time $< A_f$, valid time. Forecasts describe an event in the future
3. A_a, transaction-time $\geq A_a$, valid time. Actual values enter the database as soon as or after the event in modeled reality.

The data model must also be capable of dealing with revisions of existing data. Most often, economists receive preliminary estimates of time-series data which are then refined and revised over time. It is useful to retain the preliminary data in addition to the revised data because it helps in:

1. assessing the impact of errors in prediction using the preliminary data.
2. assessing the quality of information of a particular source of information.
3. assessing the magnitude and number of revisions in the data [349].

In the data model, revisions and preliminary data are considered versions. But, versions can be thought of as topological relationships between events, which is just another notion of time. The data structure Concept has been defined as a logical grouping of several time-series. This notion is extended to accommodate relationships between different versions of data. This is explained with an example below. The Concept USA-ECON-IND is generated by the data in the time-series GNP, INF and EMP shown in Table 3. If GNP were revised once (Table 3(a)), INF revised twice (Table 3(b)) and EMP figures were never revised, the Concept would have a "history" of its own (different values at different points in time). A part of this Concept-History (upto valid time 4) is shown in Table 3(c). This history is constructed by using the transaction times of the entries in the time-series. An additional advantage of the Concept-History is that it allows rollback to any previous state of the database.

GNP			EMP			INF		
v time	t time	value	v time	t time	value	v time	t time	value
12/31	.	1.6	01/15	.	102	01/30	.	101.1
03/31	.	1.6	02/14	.	103.1	02/28	.	101.3
06/30	.	1.5	03/15	.	101.1	03/31	.	101.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
12/31	x	1.81	12/14	y	102.1	12/31	z	101.5

(*a*)

M	
M_s	
Name	INF
Type	User-Defined : CV = MRV $+\Sigma_{i=1}^{12} INF(-i)/12 *$ $(NOW - ttime\ MRV)$

(*b*)

v time	GNP	EMP	INF
01/31	1.6	102	101.1
02/28	1.6	103.1	101.3
.	.	.	.
.	.	.	.
12/31	1.81	102.1	101.5

(*c*)

Table 10.2: Concept USA-ECON-IND @ trans-time x; v time: valid time, t time = trans time

GNP			Revised GNP		
trans time	valid time	value	trans time	valid time	value
1	0	d	1	0	d
4	3	a	11	3	a'
7	6	b	11	6	b'
10	9	c	10	9	c

(*a*)

INF			1^{st} Revision			2^{nd} Revision		
t time	v time	value	t time	v time	value	t time	v time	value
2	1	p	7	1	p'	10	1	p''
3	2	q	7	2	q'	10	2	q''
4	3	r	7	3	r'	7	3	r'
5	4	t	7	4	t'	10	4	t''

(*b*)

Concept	@ trans-time	5	Concept	@ trans-time	7
valid time	GNP	INF	valid time	GNP	INF
1	d	p	1	d	p'
2	d	q	2	d	q'
3	a	r	3	a	r'
4	a	t	4	a	t'

Concept	@ trans-time	10	Concept	@ trans-time	11
valid time	GNP	INF	valid time	GNP	INF
1	d	p''	1	d	p''
2	d	q''	2	d	q''
3	a	r'	3	a'	r'
4	a	t''	4	a'	t''

(*c*)

Table 10.3: Concept-History (does not show EMP)

The query language would always access the "most-recent" version in the Concept-History by default. Versions are Step-Wise Constant, and thus the Current version is always the same as the Most-Recent version . Additional operators to allow access to other entries in the Concept-History are required in the query language.

10.5 Rules

Rules have been shown to be powerful in maintaining views, integrity constraints and versioning in relational databases [554]. In the context of a relational database, time-series vectors could be thought of as relations and the Concept as a view. Rules provide a natural way to maintain the Concept, MRV and CV. These rules can be either data rules or action rules [510]. Thus, they are of the form "if Condition then Action" or "on Update do Action". Consider the case where a Concept is to be materialized every time a component vector is updated. To do this, when the Concept is defined, the database system would create a rule for every component in the Concept. This rule would specify that an update to a component time-series would update the Concept. In the context of our example, the rules defined by the database system would ensure that whenever a component in the Concept USA-ECON-IND is updated, e.g., new monthly figures for EMP are recorded, the rule would modify the MRV of USA-ECON-IND with the new value. Note that an appropriate strategy for the materialization of Concepts and Concept Histories has not been discussed. This will be addressed in a later chapter.

It has been assumed for most part of the discussion that the data is append only. But often, statistics are revised. For example, Base Years of indexes are changed frequently. This may lead to changes in time-series that are dependent on the original base year and in any conclusions drawn from these time-series. Rules can be used to handle the semantics of a change in historical data. An extensive discussion of the issues involved in retroactive changes to a database is provided in [183]

Rules can also be used to describe the links between different time-series. These links are essentially mathematical transformations applied to one or more time-series to generate/update other time-series. Rules would store the links and the conditions under which the mathematical transformations defined by these links are to be applied. This point is illustrated with an example from the financial domain. The example is shown in Figure 2.

In the foreign exchange market, currency traders quote the bid price and ask price for spot rates and forward rates of different currencies. Concept FOREX-MRKT is defined to capture this data. FOREX-MRKT is made up of the daily, weekly and 3-month ask price and bid price of the currencies, US Dollar (USD), French Franc (FFr), German Mark (DM) and Pound Sterling (Stg). Figure 2 shows only the USD Daily (USD.DY), USD Weekly (USD.WK) and the USD 3-Month time-series (USD.3MTH). These time-series are of type "Step-Wise Constant" and thus MRV $\equiv$ CV. The transaction time of any entry to the database is also assumed to be the same as the valid time. The later assumption is realistic in currency markets.

Assume we also need to maintain the value of a derivative instrument, the US Dollar currency option over time. The Concept, CURRENCY-OPTION-MRKT is defined as a grouping of the time-series, OPT-DOLLAR-DM, OPT-DOLLAR-FR and OPT-DOLLAR-STG. These time-series record the value of options on the mark/dollar, franc/dollar and dollar/pound exchange rate respectively. Thus, CURRENCY-OPTION-MRKT will contain the historical and current values of the above options. One of the determinants of the value of an option at any point in time is the current

price of the underlying asset, in this case, the US Dollar as quoted on the foreign exchange market, e.g., USD.DY, USD.WK, USD-3MTH. Thus, the time-series OPT-DOLLAR-DM, OPT-DOLLAR-FR and OPT-DOLLAR-STG are updated:

1. when the option is traded over the counter or on the market. In this case the value is just the price at which the option changed hands.
2. when there is a change in the value (price) of the underlying asset. When the price of the US Dollar changes, the option must be revalued by an appropriate valuation scheme [399] [263]. The value obtained from the computation is then stored in the time-series. This in turn will update the MRV of CURRENCY-OPTION-MRKT.

Thus, a change in one time series (e.g., USD.DY) causes a non-trivial series of updates in other time-series. Rules provide a way to implement this schema. The following rule system will accomplish the series of updates described above.

```
On update to USD.DY.ASKPRICE do update OPT-DOLLAR-DM.MRV
= proc-for-valuing-options(other parameters, USD.DY.ASKPRICE)


On update to OPT-DOLLAR-DM.MRV do update CURRENCY-OPTION-MARKET.MRV
= proc-for-maintaining-MRV(OPT-DOLLAR-DM.MRV, OPT-DOLLAR-FR.MRV,
   OPT-DOLLAR-STG.MRV)
```

This example shows that even a simple dependence between time-series can involve many computations. A more realistic model of the financial trading market would have more relationships and links between different time-series. Modeling these interactions and implementing the rule wake-up and update scheme without seriously impairing performance are implementation issues to be dealt with in the future.

10.6 Calendar

A calendar can be thought of as a set of time points. A built-in calendar is provided by the system and users can define their own calendars. When a time-series is defined, the user must specify the calendar and frequency of update. By default, the frequency will be *on event*. On-event means that the frequency is not fixed by a function on the calendar. Instead, the update is a random event with a certain distribution (which is not necessarily known). Updates are captured by the database in a predictable (deterministic) amount of time, T where T is small enough for any element of an on-event time-series to have its transaction time approximately equal to its valid time. When a frequency is specified, the database implicitly knows the dates and times of all elements in the time-series. These dates needn't be specified by the user. In addition, once a calendar has been established as the frame of reference for a particular time-series, other calendars can be defined on this base calendar. For example, suppose the calendar associated with time-series IBM-STOCK is "American-business-days". Calendars like "Holidays" and "Business-Hours" can be defined as a function of "American-business-days". Given the additional calendars, rules can be used to specify constraints for real-world facts, e.g., IBM-STOCK will only have values in "Business-Hours" during "American-Business-Days" excluding "Holidays".

10.7 Temporal Query Language

The query language must be able to support temporal representation and reasoning. It must also be able to recognize natural language expressions referring to collections of time intervals. The following section discusses the main features of the temporal query language.

1. Support for Calendar operations: As described in the previous section, a Calendar is a collection of an infinite sequence of intervals that span the timeline. The query language allows definition of a calendar based on primitive time units and also on previously defined calendars. The first interval in the calendar (the first time we recognize) should be specified but there is no theoretical last interval in the calendar because the timeline is an infinite set.
2. Support for operations on the time-line, e.g., Overlaps, When, Meets, Between, Contains [528] [427] [488].
3. Support for New Operators : Two new operators are defined on the calendar (based on [350]).
   (a) Division: This operator provides a way of dividing a time interval into smaller intervals. The operator takes time intervals as arguments and generates a collection of time intervals. It is always used in conjunction with the temporal operators discussed above. For example, if we wanted to find the Weeks that overlapped with the month of December in 1991, the division operator (:), could be used as:

   ```
   (WEEKS:overlaps:Dec-91) =
                        Collection of weeks that overlap Dec-91.
   ```

   In the above example, WEEKS is the collection of weeks in 1991 and Dec-91 is the collection of days in the month of December in 1991.

   We note that the division operator has a strict and relaxed interpretation. In the strict interpretation (:), the above query would generate only those weeks or parts of weeks that strictly overlapped with December in 1991. On the other hand, the relaxed interpretation (.), would generate all weeks which had some overlap with December 1991. Thus, in the relaxed interpretation, we could get parts of weeks that overlapped with Jan'92 and Nov'91.

   (b) Selection : This operator selects a time interval from a collection of time intervals. For example, 1/WEEKS selects the first Week from the collection of WEEKS.

The query language, allows the construction of computer understandable expressions corresponding to natural language expressions used in reasoning about time. A few examples are given below.

1. First Day of every month $\equiv$ 1/DAYS:during:MONTHS

   Assume the reference year is 1992. Then, DAYS and MONTHS are two self-explanatory collections of time points. Each month in the collection MONTHS is divided into days by the *division* operator, in conjunction with the *during* operator. The *select* operator then selects the first day from this collection of days.

2. First Week of the month $\equiv$ 1/WEEKS.overlaps.MONTHS
   In the above expression, WEEKS refers to the collection of weeks in the year 1992. Each month in the collection MONTHS, is divided into weeks based on the *division* operator in conjunction with the *overlaps* operator. From this collection of weeks, the *select* operator chooses the first.

10.8 Special Operators for Time-Series Database

1. Moving-Average Operator: MA(X,n). This operator performs the operation
   $$MA(X,n) = \{y_i | y_i = \sum_{j=i}^{i+n-1} x_j/n\}$$
   where X is the time-series and n is the number of time-periods for which the moving-average is required.
2. Autoregression Operator : AR(X,n), The regression of time-series X against its lags upto n periods.
3. LAG operator: This operator does the following operation on the vector X:
   $$Y_i = X_{i+1} - X_i, \ \forall i = 1, n-1$$
   where X is the time series, X_i indicates the i^{th} element of X, Y is the one-lag time-series derived from X and n is the size of X. The lag operator also takes a parameter(λ) of type integer and of size ($\leq n$). This lag parameter indicates the number of lags required. With the lag parameter, the lagged formula is:
   $$LAG(X,\lambda) = X_{i+\lambda} - X_i, \ \forall i = 1, n-\lambda$$
   The operators LAG and AR are used for Vector Autoregression. Vector Autoregression has proven to be a successful technique for forecasting systems of interrelated time series variables. Vector Autoregression is also used for analyzing the dynamic impact of different types of random disturbances and controls on systems of variables. It involves a system of equations that make each endogenous variable a function of its own past (LAGS) and the past of other endogenous variables in the system.
4. Aggregation Operators like MAX, MIN, AVG, MED, frequency distribution calculation. Extensions to TQUEL to incorporate aggregates and their semantics have been proposed by [531]. Aggregates in time-series databases must incorporate three distinct semantic interpretations:
   - aggregate on time-series, e.g., average stock price over the time period $[0,T]$.
   - aggregate on cross-sectional data, e.g., the average salary of all employees now.
   - aggregate on the time-series of aggregations on the cross-section, e.g., average stock price of computer industry stocks over the time period $[0,T]$. This would involve computing the average stock price of the computer industry at each time point between 0 and T and then averaging over these values.

5. Matrix Manipulation Operators: These operators include
   (a) Matrix Addition, Subtraction, Multiplication: useful in constructing time-series vectors from other time-series vectors.
   (b) Matrix Inversion: Used in computing regression coefficients.
   (c) Computation of Eigen Values.
   (d) Special Algorithms to deal with Sparse Matrices: Useful in recording time-series that are irregular.
   (e) Solving a system of linear and non-linear equations or generically if $f : R^n \mapsto R^n$, we would want an efficient algorithm to solve the system of equations $f(x) = 0$. These algorithms are useful for regression analysis especially when performance is an issue.
6. Transformation
   This operator is used to convert from one time frequency to another. Conversions from a lower frequency to a higher frequency are allowed only if the semantics of the transformation are clear. For example, conversion of a time series with weekly frequency to a time series with a monthly frequency is permitted, but conversion of the same time-series to a daily frequency is not permitted since the semantics are not clear. The transformation operator must be able to interact with the calendar and transform data, accounting for the calendar associated with the time-series.
7. Event Construction This operation is used to construct cross-sectional and pooled data from the underlying time-series data in the database. Semantically, event construction is similar to the event join described in [507]. One could think of this operator as similar to a join in a relational database (with the join performed over dates). Problems arise when joining over dates because of the different frequencies of time-series data. Even when the frequency of two time-series is the same, they could be based on different calendars. For example, if the application required the weekly closing prices of IBM-Japan and IBM-USA, joining the prices on Friday for each week may not necessarily give the correct result. It is possible that in some weeks Fridays are holidays in Japan and working days in the US and vice versa. In these weeks the closing price on Thursday defines the weekly closing price for IBM-Japan while the weekly closing price for IBM-USA is the closing price on Friday. Equi-Join over dates would give incorrect results because matches may never be found.
   Event Construction has two interpretations:
   (a) perform an event-join over the time-attribute.
   (b) transform all the data to a common frequency and then perform the event join.

   Users must be able to choose the relevant interpretation. Event-join processing strategies have been outlined in [507] but they must be modified to account for the proposed data model. One area of difference is that they assume time-start and time-end attributes exist for the data, while we store only the time-start attribute.
   Additional problems that the Event Construction operator must be able to solve are:

- If time-series have different calendars, the time values must be converted to a common base. This assumes that the relation between different calendars are known.
- Handling of missing values.
- If the lifespans of time-series are different, the operator must know how to handle the Event Construction tuples that have missing values.
- Forecasts and revisions are not used in the Event Construction unless specified in the query. When revision is specified the operator must reconstruct the Concept and if necessary the Concept-History. When forecasts are necessary, the operator must use the forecasts when joining and also know what to do when different time-series have forecasts for different horizons in the future.

  Event construction is an extremely important part of data analysis because it must always be done before any meaningful regression analysis. We emphasize that event construction is only meaningful within a Concept. Query optimization of the Event construction operator is to be discussed in a later chapter.

10.9 Handling of Missing Values

There are several reasons why users often encounter missing values in observational cross-sectional and time-series data. We attach four different interpretations to missing values:

1. indeterminate
2. not applicable
3. not available
4. actually zero but not explicitly mentioned in the database

The latter case is the easiest to handle and the indeterminate case can be clubbed with the interpretation of "not available at this time". The problem of handling missing data has been researched extensively in the database literature. [125] describes an algebra for handling null values while [599] and [365] have a detailed treatment of the "not applicable" and "not available" interpretations. The database will assign a value of "NA" in cases where the value is not applicable. In a time-series with the calendar "American-Business-Days", Saturdays, Sundays and specific holidays will have the value "NA". Thus, once a calendar has been specified for a time-series, the database will assign "NA" at appropriate time points. Users may define the value of the time-series to be "NULL" or "0". "NULL" has the interpretation of indeterminate or not available.

10.10 User Environment

The underlying data in this kind of database is rarely (if ever) updated by users. Data is obtained through real-time data feeds (e.g., Reuters) or batch loading of the data. In any case, users aren't expected to enter data on their own or modify and delete tuples from the database. A typical user session is an interactive and iterative process

of model selection, parameter estimation and error analysis. A typical user session is envisaged as one in which:

1. Users copy relevant time-periods of data into their own "work area". The work area is a private sub-directory which is maintained by the database.
2. Users "refine" the data, by removing outliers, transforming the data using operators like lags, mathematical transformations (logs) or event-constructions.
3. Users construct scripts which are essentially strategies for data analysis. These scripts would typically be composed of declarative statements in the query language and would contain the logic and methodology for data analysis. For example, these scripts would normally contain commands for regressions, error analysis or diagnostics.
4. Since this is an interactive session, users will store temporary files. These files would normally contain results of the data analysis and temporary data files. In addition to the data files, users are expected to store the sequence of commands that they used to get the results.
5. Users interface with graphical software in order to see diagrams like histograms, residual plots or boxplots.
6. Users construct their own data analysis procedures using existing primitive building blocks or through the use of an external language.
7. Users interface with sophisticated mathematical and statistical libraries available on the computer system.

Given the sophisticated environment required for data analysis, the database must provide the following features:

1. A transaction management scheme that allows users to work on the same dataset at the same time in private work areas, check-in and check-out datasets from the main database and save and recover the temporary data files they create as well as the scripts used for analyzing the data files.
2. Support for statistical, temporal and the operators outlined in this chapter.
3. Support for creation of scripts using the query language which can be augmented by an interface to external mathematical libraries and graphical routines.
4. Support for the creation of complex operators in the query language based on the primitive database operators, as described in [417].

10.11 Conclusion

In this chapter, we have defined the requirements of a database model meant for the analysis of time-series data. We treat time-series as n-ary vectors and introduce Concepts to logically group these vectors together. Different time-lines, transaction time and valid time are supported in the model and different notions of time, e.g., versions, time points on different calendars, are treated uniformly. The importance of Rules have been discussed to maintain semantic integrity, derive time-series based on other time-series or database transactions, for implementation of operators like Transformation and Event Construction and for uniform management of the different notions of time. We provide an interactive environment that helps users perform data analysis. The environment accounts for the fact that data is never updated by users and that

users keep copies of parts of the data in their private workspaces along with data manipulation routines. A query language that incorporates frequently used statistical and temporal operators is provided. This includes the capability of creating new operators based on the primitive ones and an interface to mathematical and statistical routines.

An extremely important issue that we do not raise in this chapter is : How far should we go in building in extra functionality in the database to accommodate different data environments ? Building in extra functionality means bulkier database code and high overhead. Also, the data model is not independent of the application. On the other hand, not building in functionality, forces users to couple database systems with external programs. This can be highly inefficient because optimizers aren't able to optimize the external code. Also, users tend to be impatient with systems that must be customized before use.

Our future research plans are briefly outlined below.

- Materialization : We briefly described an immediate update strategy for Concepts and Concept Histories. The cost analysis for different materialization strategies is important for performance. A cost analysis of the strategies of immediate update, deferred update, random update and hybrids, would determine the appropriate strategy given the characteristics of the database. By characteristics, we refer to the number and size of time-series vectors, the rate of update of vectors and the frequency of queries on Concepts and their component vectors.
- Query Optimizer : We have briefly discussed the optimizer in the context of data analysis procedures. But we have not specified the exact scope and role of the optimizer with respect to the database environment proposed in the chapter. Efficient Sampling of data, indexes for temporal data to facilitate efficient search and retrieval and stochastic modeling of the temporal data process to obtain selectivity estimates are problems that must be addressed.
- Indexing and Access Methods: Conventional access methods (B-trees) aren't of much help because the data is sequential. Also, unlike other database applications, sequential "browsing" is not a major issue. We believe the access methods must reflect operations that are frequent and important in statistical data analysis. A more thorough analysis of data analysis procedures will give us a better idea of what access methods are appropriate.

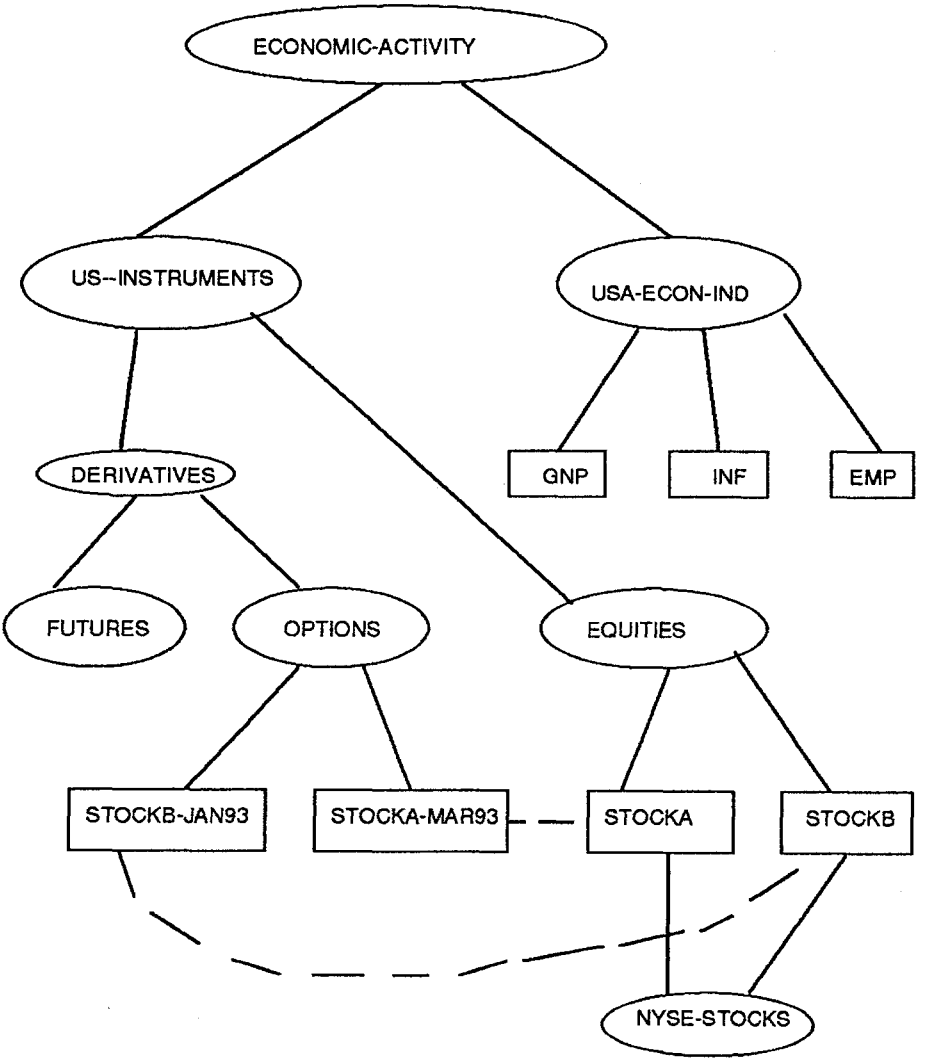


Figure 1: "Concept" ECONOMIC-ACTIVITY

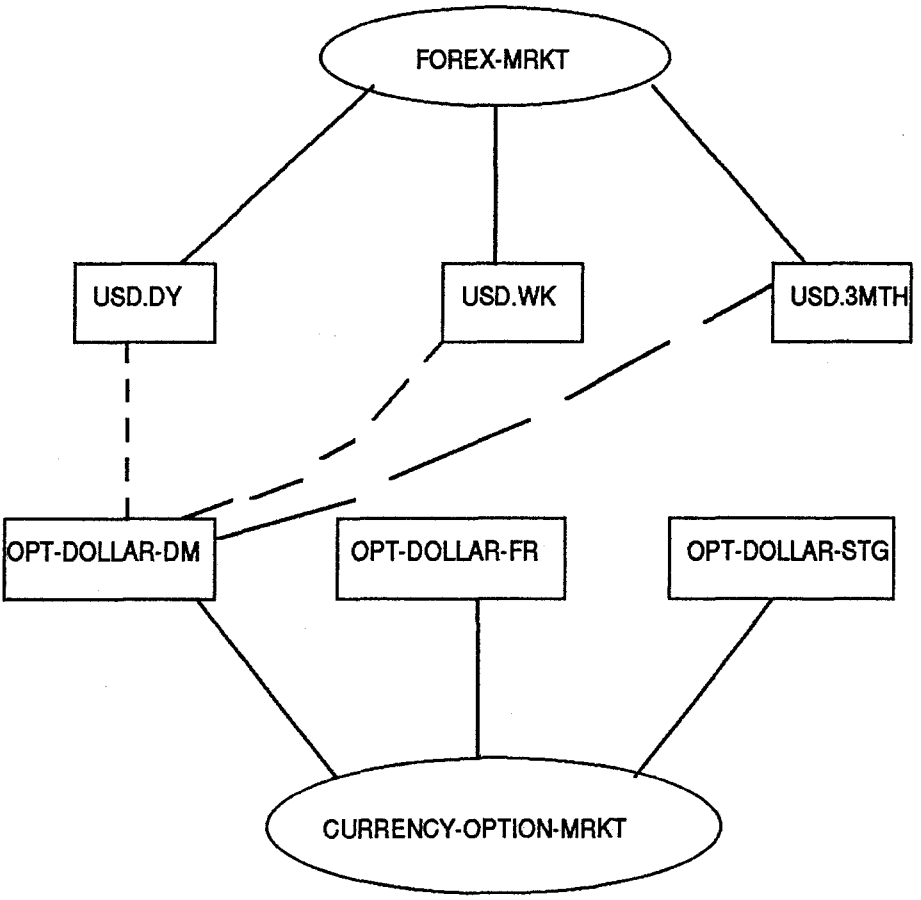


Figure 2: Simplistic view of Financial Trading Market

Chapter 11

A Relational Model and SQL-like Query Language for Spatial Databases

Shashi K. Gadia*, Vimal Chopra†

11.1 Introduction

Spatial databases have experienced enormous growth in application environments, such as agriculture, environmental studies, geography, geology, city-planning, aerospace industry etc. More recently spatial databases have attracted attention in the database community. A considerable research has been done in physical implementation of spatial databases. This is particularly true of access methods for spatial data [238, 483, 512, 230, 231, 39, 506]. On the other hand, abstract modeling and querying of spatial data have received relatively less attention. The need for such a study becomes even more important because of diverse techniques proposed for representing spatial regions.

Like [440] we favor that the logical view and the physical implementation of spatial data should be considered orthogonal issues. The users should be given a simple view of data and freed of the worry of how it is physically represented. This is even more important because physical implementation will continue to be a topic of study for quite some time to come. Conventional database techniques are inadequate in spatial databases because of the spatial structure implicit in spatial querying. We present a model and an SQL-like query language called SpaSQL (read space-Q-L) for spatial data. Without tying ourselves down to a choice of representation of spatial regions, we propose certain desirable closure properties for them to make SpaSQL seamless.

11.1.1 Related Works

Several techniques of physical representation spatial regions have emerged. One way to represent a region is to lay it on a fine grid, and approximate the region by the set of grid elements covered by the region. Another technique is to describe a region by a chain of arcs, and associating a bi¿t with each arc to encode whether the region being specified is on the right or left of the arc. A third way is to view a region as a hyper plane in an n-dimensional space, and use techniques of computational geometry to manipulate them.

*Department of Computer Science, Iowa State University, Ames, IA 50011-1040, U.S.A.

†This work was done while the second author was at Iowa State University. His current address is: Geoquest, Data Management Division, 5725 Paradise Drive, 100, Corte Madera, CA 94925

The representation of spatial regions is a challenging and enormous problem by itself, and we feel it will be investigated for several years before it is completely understood. It is quite possible that different problem domains would require different techniques. [232, 441, 230, 231, 562] detail several techniques for physical representation of spatial regions.

At the user interface level, Scholl and Voisard [499] define elementary regions as lines, polygons etc. A region is defined either as an elementary region or as a set of elementary regions. They treat maps as relations and introduce operations like projection, cover, map overlay, superimposition, selection windowing and clipping over the relation (map). Guting [237] identifies several different types of spatial domains *e.g.* point type, line type (curves), polygon type etc. He also introduces some basic operations like inside, outside, intersection, diameter, length etc. over these domains. He uses strong data typing. Orenstein and Manola [441] view spatial regions as mathematical abstractions called *point sets.* They introduce interesting algorithms to implement spatial overlay, union, intersection, difference and joins. [20] covers SQL-type querying, but their main focus is on architecture for spatial data. Additional papers on relational approach relevant to our work include [98, 99, 100, 473, 299, 438]. There is also a trend in spatial databases toward object orientation [439, 411, 388, 567, 5].

An important objective of a query language is to provide a natural interface to a user. As stated above, in this chapter we present a model and an SQL-like query language SpaSQL to achieve this objective. Now we discuss features of our approach.

11.1.2 Our Concept of A Spatial Region

We assume that we are given some universal region $\mathcal{R}$. A user views $\mathcal{R}$ as a set of points. Every region that the user or the system encounters will be some subset of $\mathcal{R}$. It is not possible to have a tractable or finite description of all possible subsets of $\mathcal{R}$. However, all possible subsets of $\mathcal{R}$ are not of interest to us either. We postulate that the set of subsets of $\mathcal{R}$, that the user is interested in, is denoted *REG*. A particular implementation may implement *REG* in any convenient way so long as the following closure properties are satisfied: if *reg*, reg_1 *and* reg_2 are regions in *REG*, then $reg_1 \cup reg_2$, $reg_1 \cap reg_2$, $reg_1 - reg_2$ and $\neg reg$ (complement of *reg*) are regions in *REG*.

There are important potential advantages of our concept of a region as introduced above. At the model level, the closure properties make it possible to store a single object in a single tuple. At the query language level, one may use $\cup, \cap$ and $\neg$ to capture the booleans *or*, *and* and *not* of English in a seamless way. Our model and SpaSQL incorporate these ideas and eliminate the *boolean seams.* Justification for these claims will be given in Section 11.4. In that section we will compare PSQL of [440] with SpaSQL. Even though [440] assume the closure properties for spatial regions, PSQL is not as seamless as SpaSQL. This is because in PSQL the attribute values are not modeled as functions of time, but rather region is used as an attribute at the tuple level. The boolean seams in their model stem from the fact that they split the description of a single object in several tuples.

Note that Scholl and Voisard [499] define a region to be a set of (elementary) subregions, whereas we view a region as a union of subregions. We feel a union is a better abstraction, because a given region can have many different representations as a set of subregions, whereas a union has a canonical representation. A user can view our region as a flat object, a set of points having no seams, rather than as a set of set of points as in [499]. Our regions simplify the semantics of algebraic operators in spatial databases. Also, unlike [499], we do not have any inherent distinction between

elementary regions and regions. We suggest only regions of type *REG* as an abstraction of all spatial regions. We emphasize that at the implementation level, a region may be a mixture of finitely many elementary types, such as points, lines, curves, polygons. The idea is to free the user from seams arising from strong data typing of regions. At the same time, to facilitate navigation, we favor that the user be supplied with functions to determine the type of a region, e.g. IsPoint(reg).

11.1.3 Weak Data Typing

Unlike [237] discussed above, we favor weak data typing of operations on regions. The reason for this is to avoid unnecessary run time errors. For example consider the construct Interior($reg_1 \cap reg_2$), where reg_1 and reg_2 are regions (of type *REG*). The construct Interior(reg) returns the interior of reg, after removing its boundary. For most tuples $reg_1 \cap reg_2$ may be a region but for some it may degenerate into a curve or a point if reg_1 and reg_2 are merely touching each other. In such degenerate cases we want Interior to return an empty region (denoted ϕ) rather than an error. In relational systems such constructs are mainly used in selection; the problem becomes more serious when such a selection is nested inside another query and an error at the inner level could abort a computation unnecessarily.

11.1.4 Uniformity of Attribute Values

Roussopoulos et al [440] add region as an attribute to a relation in order to deal with spatial information. In our model the spatial aspect is incorporated at the level of an attribute value which is defined as a function of time. For example consider the data value taken by the attribute CROP in Figure 11.1. The semantics of this data is "the value of CROP is wheat in the region $creg_1$ and it is CORN in the region $creg_2$.

CROP
$creg_1$ wheat
$creg_2$ corn

Figure 11.1: A data value

11.1.5 Experience From Temporal Databases

[203] introduced a *temporal element* as a finite union of intervals. Temporal elements are closed under $\cup, \cap$ *and* complementation, and their use as timestamps hides the seams arising from *and*, *or* and *not* appearing in English queries. In temporal databases the issue of interval v/s temporal elements as timestamps has been debated for sometime. The basis for this debate is that interval timestamps make it possible to have fixed length tuples, and this brings temporal databases within the realm of first normal form. On the other hand use of intervals alone causes history of an object to split over several tuples, leading to query languages which are complex from the point of view of users [205].

Like spatial databases a temporal database can have more than one dimension. However, even in higher dimensions unions of rectangles suffice. The reason for this is perhaps that the time changes linearly along every axis giving rise to a rectangular

granule. This is not the situation in spatial databases where the whole space is there, all at once.

The remainder of this chapter is organized as follow. Our model for the spatial database is presented in Section 11.2. In Section 11.3 we present our query language SpaSQL and some example queries. We have taken [440, 499] as samples of works on querying in spatial databases. In Section 11.4 we discuss [440] in detail; we show how our queries remove boolean seams arising from *and*, *or* and *not*, mentioned above. The operators from [499] are expressed in SpaSQL in Section 11.5. We are able to express all the queries given in [440, 499] with some minor exceptions. The chapter is concluded in Section 11.6.

11.2 Our Model

In this section we informally introduce our model for spatial data along the lines discussed in the introduction. Querying in our model is covered in the next section.

11.2.1 Spatial Regions

As stated in Section 11.1, we assume an underlying universal region $\mathcal{R}$. The user views it as a set of points. We have also postulated a set *REG* consisting of subsets of $\mathcal{R}$ which are of interest to users, and that *REG* is closed under union ($\cup$), intersection ($\cap$), subtraction (-) and complementation ($\neg$). Throughout this chapter by the term *region*[‡] we mean an element of *REG*.

Note that we do not make specific assumptions about the constitution of $\mathcal{R}$. $\mathcal{R}$ can be an n-dimensional Euclidean space, surface of a sphere, portion of a plane, a curve and so on. We do not assume that $\mathcal{R}$ is discrete or continuous. Our main hypothesis is that the regions in *REG* should have some reasonable description. Note that a set may be infinite, but its description may be finite. For example suppose $\mathcal{R}$ is the 2-dimensional Euclidean plane $\{(x, y) : x \text{ and } y \text{ are real numbers}\}$. We may describe the upper half plane simply as $y \geq 0$, and the right half plane as $x \geq 0$. Although the two half planes are infinite, their intersection is easily computed as $x \geq 0 \wedge y \geq 0$. The union of the two half planes is described as $x \geq 0 \vee y \geq 0$. Complement of this union may simply be described as $\neg(x \geq 0 \vee y \geq 0)$, or $x < 0 \wedge y < 0$. (We do not imply that the system should reduce such a description immediately when it is encountered; we consider this as an implementation issue.) In the above notation there is no need to describe $\mathcal{R}$ itself in a complicated manner; it is described by the constant predicate TRUE.

11.2.2 Attribute Values

As stated in Section 11.1, to capture the value of an attribute, we introduce the notion of a spatial assignment which would be a function from a spatial region into dom(A), the domain of A. We are not interested in allowing an arbitrary function to be considered a spatial assignment. The following example illustrates this point and motivates the formal definition of a spatial assignment.

Example 11.2.1 Suppose $\mathcal{R}$ is the interval $[0, 1] = \{x : 0 \leq x \leq 1, x \text{ is a real number}\}$. We define a function χ on $\mathcal{R}$ as follows. $\chi(x) = 0$ if x is a rational real number (a

[‡]In our more recent papers [109, 204] we use the term *spatial element* for a region in *REG*

fraction), and $\chi(x) = 1$ if x is an irrational real number. Although χ takes only two values, its description is very complex. This may be because the set of rational points in [0,1] is difficult to describe and we may not want to consider it a region in *REG*. Note that in functional notation $\chi^{-1}(a)$ denotes the inverse image of a under the function χ. Thus $\chi^{-1}(0)$ is the set of rationals in [0,1] and $\chi^{-1}(1)$ is the set of irrational in [0,1]. We want all inverse images to be regions in *REG*. Thus we may not wish to consider χ to be an spatial assignment.

The above example motivates the following definition of a spatial assignment. A spatial assignment (or simply assignment) ξ to an attribute A is a function from some region reg in *REG* into domain of A, such that (i) ξ takes only finitely many values, and (ii) inverse image of every value taken by ξ is a region in *REG*. This allows us to represent a spatial assignment as $\langle reg_1\ a_1, reg_2\ a_2, \cdots, reg_m\ a_m\rangle$, where $reg_1, reg_2, \cdots, reg_m$ are regions in *REG*, and for each i, $1 \leq i \leq m$, a_i is the value of the assignment at every point in reg_i. Figure 11.1 is an example of a spatial assignment to the attribute CROP, which can also be represented as $\langle reg_1$ wheat, reg_2 corn$\rangle$.

The domain of an assignment is called its *spatial domain*. The operator $[\![.]\!]$ denotes the spatial domain of a spatial assignment. Thus $[\![\langle reg_1$ wheat, reg_2 corn$\rangle]\!] = reg_1 \cup reg_2$. The restriction of a spatial assignment ξ as a function to the spatial region reg is denoted $\xi\lceil reg$.

11.2.3 Value Navigation

We assume that certain binary operators such *as* $=, \leq$ *etc.* are available. In classical databases these operators allow us to make comparisons, such as $2 \leq 4$, which evaluates to TRUE, and $5 < 3$ which evaluates to FALSE. Syntactically, this leads to boolean expressions of the form $A\theta B$ where A and B are attributes. Such boolean expressions are used in selection operators. In the spatial context we need a way of comparing spatial assignments, *i.e.*, compare functions of space. Clearly, such a comparison does not make sense at points where one or both of the assignments are not defined. Also it may happen that at some points the comparison returns TRUE and at other points it returns FALSE. Thus it is not useful to view the overall result of $A\theta B$ as a TRUE or FALSE value. In fact it should yield the set of points where the comparison of their values returns TRUE

More formally if ξ_1 and ξ_2 are spatial assignments then we define $[\![\xi_1\theta\xi_2]\!] = \{x :$ ξ_1 and ξ_2 are defined at x and $\xi_1(x)\theta\xi_2(x)$ is TRUE$\}$. The construct $[\![\xi_1\theta\xi_2]\!]$ is of fundamental importance in spatial databases. As stated above, it evaluates to the set of points where ξ_1 is in θ relationship with ξ_2 and its value lies between ϕ and $[\![\xi_1]\!] \cap [\![\xi_2]\!]$. The value is ϕ if the spatial assignments are never related.

It is natural to expect $[\![\xi_1\theta\xi_2]\!]$ to be a region. To see this suppose ξ_1 is $\langle reg_1\ a_1,$ $reg_2\ a_2, \cdots, reg_m\ a_m\rangle$ and ξ_2 is $\langle reg'_1\ b_1, reg'_2\ b_2, \cdots, reg'_n\ b_n\rangle$, where reg_i and reg'_j are regions in *REG*. Clearly, $[\![\xi_1\theta\xi_2]\!] = \cup\{reg_i \cap reg'_j : 1 \leq i \leq m,\ 1 \leq j \leq n$ and $a_i\theta b_j$ holds$\}$, which is a region in *REG* because *REG* is closed under $\cup$ and $\cap$.

11.2.4 Spatial Tuples

A *tuple* is a concatenation of spatial assignments whose spatial domains are the same. The spatial domain of a tuple τ, denoted $[\![\tau]\!]$, is simply the spatial domain of any of its spatial assignments. The assumption that all spatial assignments in a tuple have the same domain is called the *homogeneity* assumption [203]. The *restriction* of a tuple

τ to a spatial region reg, denoted $\tau\lceil reg$, is the tuple obtained by restricting every assignment in τ to reg.

A literal implementation of a tuple may have substantial redundancy, as the underlying region is repeated from one attribute to another. We assume that a clever implementation attempts to minimize such redundancy. Note that in spatial databases inconsistency between different data layers (attribute values) is a common place due to complexity of mapping spatial regions [227]. It is a nontrivial problem, and introduces errors in spatial data. We feel that expert techniques for reconciling different attribute values should be incorporated at the implementation level. When several techniques for reconciling are available the choices should be made available to the user at a high interface level.

11.2.5 Spatial Relations

A spatial *relation* r over R, with $k \subseteq R$ as its *key*, is a finite set of non-empty tuples, such that no key attribute value of a tuple changes from one spatial point to another, and no two tuples agree on all their key attributes. Figure 11.2 shows a county relation with the schema CNAME CROP. We designate CNAME as its key; this satisfies both the requirements of a key: within the same county the CNAME of a county does not change from one place (point) to another, and no two counties have the same CNAME. The figure also shows a state relation with the schema SNAME CNAME, with SNAME as its key. Note that the key attributes are underscored.

CNAME		CROP
$creg_1 \cup creg_2$	story	$creg_1$ wheat $creg_2$ corn
$creg_3 \cup creg_4 \cup creg_5$	orange	$creg_3$ wheat $creg_4$ barley $creg_5$ rice
$creg_6$	polk	$creg_6$ wheat

The county relation with CNAME as its key

SNAME	CNAME
$creg_1 \cup creg_2 \cup creg_6$ IA	$creg_1 \cup creg_2$ story $creg_6$ polk
$creg_3 \cup creg_4 \cup creg_5$ CA	$creg_3 \cup creg_4 \cup creg_5$ orange

The state relation with SNAME as its key

Figure 11.2: An example database

If r is a spatial relation over R, then the *spatial domain* of r, denoted $[\![r]\!]$, is the union of the spatial domains of all its tuples. From closure properties of REG, $[\![r]\!]$ is clearly seen to be a region. For example, for the state of the county relation shown in Figure 11.2, $[\![\text{county}]\!] = creg_1 \cup creg_2 \cup creg_3 \cup creg_4 \cup creg_5 \cup creg_6$. The *restriction* of a relation r to a spatial region reg, denoted $r\lceil reg$, is the relation obtained by restricting every tuple of r to the spatial region reg.

11.2.6 Weak Equality and Restructuring

Suppose r and s are relations over the same scheme and have the same key. Then we say that r is *weakly equal* to s, provided at every point p in the spatial universe, the restrictions of r and s to $\{p\}$ are the same (*i.e.* for every point p, $r\lceil\{p\} = s\lceil\{p\}$). Two weakly equal relations differ in structure, but in some sense they have the same information content. Sometimes in forming a query, it is necessary to view the relation to have a different key. This is achieved by computing a weakly equal relation with the desired key. The following theorem tells us when this is possible.

Theorem 1 Suppose r is a relation with K as its key, and K' is such that K' is a key of $r\lceil\{p\}$ for every point p in the spatial universe. Then there exists a unique relation s, such that r and s are weakly equal, and K' is the key of s.

Example 11.2.2 Suppose for the county relation, instead of CNAME we want to use CROP as the key. This is achieved by restructuring the county relation to obtain the county′ relation as shown in Figure 11.3.

CNAME	*CROP*	
$creg_1$ *story* $creg_3$ *orange* $creg_6$ *polk*	$creg_1 \cup creg_3 \cup creg_6$	*wheat*
$creg_2$ *story*	$creg_2$	*corn*
$creg_4$ *orange*	$creg_4$	*barley*
$creg_5$ *orange*	$creg_5$	*rice*

Figure 11.3: The county′ relation with CROP as key

11.3 Querying in The Model

In this section we introduce the SQL-like query language SpaSQL. At the end of this section we will give several interesting example of queries.

We assume that a database consisting of spatial relations is given. The set of all SpaSQL expressions can be divided into three mutually exclusive groups: *spatial expressions*, *boolean expressions* and *relational expressions*. Note that we want to allow constructs like SNAME = CA in the queries. This causes a technical problem: SNAME evaluates to a spatial assignment during query evaluation, but CA is a constant. This difficulty is easily removed by identifying a constant c with the spatial assignment which is a constant function over the universe $\mathcal{R}$ of space.

11.3.1 Spatial Expressions

Spatial expressions, the syntactic counterpart of spatial regions, are formed from $[\![A]\!]$, $[\![r]\!]$, $[\![A\theta B]\!]$, $[\![A\theta b]\!]$, $\cup, \cap$, and $\neg$. Additional spatial expressions are formed using built-in functions such as *Point*(x,y), *Circle*(center,radius), *Window*$(x \pm a, y \pm b)$, *Boundary*(reg), *Interior*(reg) and *Exterior*(reg).

If μ is a spatial expression, and τ is a tuple, then $\mu(\tau)$, the result of substituting τ in μ, is defined in a natural manner. Instead of giving a formal definition of tuple substitution, we illustrate it by an example.

Example 11.3.1 Let us consider the county relation of Figure 11.2. [[CROP = wheat]] is a spatial expression. Suppose τ denotes story county's tuple. Then [[CROP = wheat]](τ) is $creg_1$. The result of substituting τ in the spatial expression [[CNAME]] *is* $creg_1 \cup creg_2$.

11.3.2 Boolean Expressions

Boolean expressions are formed from TRUE, FALSE, $reg_1 \subseteq reg_2$, $\wedge$, $\vee$ and $\neg$. Additional boolean expressions are formed using built-in functions such as IsPoint(reg), IsLine(reg) IsPolygon(reg), reg_1 Equal reg_2, reg_1 Inside reg_2, reg_1 Outside reg_2 and reg_1 Intersects reg_2.

11.3.3 Relational Expressions

Relational expressions in SpaSQL are the syntactic counterparts of spatial relations. Now we introduce relational expressions.

Union and Difference

Suppose r and s are spatial relations with the same schema and key. To arrive at r union s we first compute the union of r and s treating them as sets, and then collapse each pair of tuples of r and s which agree on all key attributes, into a single tuple. Computation of r difference s is similar; in union non-overlapping part is added while in difference overlapping part is removed from a tuple of r.

The Select Statement

The select statement of SQL is important and it is the counterpart of SPJ-expressions of an algebra. (SPJ stands for select project join). In SpaSQL, the select statement is of the form

select attribute-list
restricted_to spatial-expression
from relation-list
where boolean-expression

The semantics of the above select statement is as follows. A tuple τ is formed by selecting tuples from each relation in the relation-list. For this tuple τ, boolean-expression is verified. If τ does not satisfy boolean-expression, it is rejected. If τ satisfies boolean-expression, then spatial-expression is evaluated for this tuple. This gives us the portion of domain of τ, which is of interest to us. The tuple τ is now restricted to this domain. The tuple τ is made homogeneous, if necessary. If its domain becomes empty, it is rejected; otherwise its attribute values specified by attribute-list are retrieved.

Like the classical SQL, the where clause is optional; when omitted, it defaults to TRUE. This amounts to qualifying all tuples for retrieval. The restricted_to clause is also optional. When omitted, it defaults to $\mathcal{R}$, meaning no part of a qualifying tuple is removed. A detailed discussion of key is omitted. It is clear that the boolean-expression and spatial-expression are both recursive as they can involve relational expressions. This makes the select statement of SpaSQL is very powerful.

11.3.4 Examples

We assume that our database consists of the relations as shown in Figure 11.4. Although the spatial domain of each of the relations is of type *REG*, underlying assumption is that in the implementation city is represented as point, highway as a curve, and county and state as polygons.

counties (COUNTY, STATE, CROP)
cities (CITY, COUNTY, POP)
hwys (HWY)
states (STATE, TIME-ZONE)

Figure 11.4: An example database

Example 11.3.2 The query *for the highways which pass thru IA, find the portion which is inside MN* is expressed in SpaSQL as follows.

```
select *
restricted-to [[select * from states where STATE = MN]]
from hwys
where [[HWY]] Intersect [[select * from states where STATE = IA]]
```

Example 11.3.3 The query *find all states whose boundary length is more than 2000 miles* is expressed in SpaSQL as follows.

```
select STATE
from states
where Boundary([[STATE]]) > 2000
```

Example 11.3.4 The query *find all the cities along with their population (POP), which are within 200 miles of Chicago* is expressed in SpaSQL as follows.

```
select CITY, POP
from cities
where [[CITY]] Inside Circle([[select * from cities where CITY = Chiago]], 200)
```

Example 11.3.5 The query *find all the counties in IA which grow wheat in a total of more than 1000 square miles* is expressed in SpaSQL as follows.

```
select COUNTY
from counties
where Area([[select * restricted_to [[CROP = wheat]] where STATE = IA]]) > 100
```

Example 11.3.6 The query *find all the counties in central time zone* is expressed as follows. (Note that a state can be in more than one time zone.)

```
select COUNTY
from counties
where [[COUNTY]] Inside [[select *
                           restricted_to [[TIME-ZONE = central]]
                           from states]]
```

Example 11.3.7 The query *find the cities which are in the central time zone and within 50 miles from the intersection of two highways and rice is grown in that area* can be expressed as follows. (Note that "from hwys H1 H2" is meant to create two aliases H1 and H2 of hwys.)

```
select CITY
restricted_to [[select * restricted_to [[CROP = rice]] from counties]]
from cities
where [[CITY]] ⊆ [[select * restricted_to [[TIME-ZONE = central]] from states
      and (Distance([[CITY]],
                [[select * restricted_to [[H1]] ∩ [[H2]] from hwys H1 H2]]) ≤ 50)
```

11.4 Seamlessness of SpaSQL

In this section we justify our claims of boolean seamlessness of SpaSQL. We say that a query system does not have a boolean seam if it handles *and*, *or* and *not* of natural languages symmetrically. Instead of formalizing this idea into a more precise notion, we illustrate it through examples. We consider the above constructs of natural languages and show how they are handled in PSQL of [440] and SpaSQL.

CNAME	CROP	CREGION
story	wheat	$creg_1$
story	corn	$creg_2$
orange	whealt	$creg_3$
orange	barley	$creg_4$
orange	rice	$creg_5$
polk	wheat	$creg_6$

Figure 11.5: The county relation of Figure in PSQL framework

We consider the county(CNAME,CROP) relation of Figure 11.2. In the PSQL framework, this relation would be represented as county(CNAME,CROP,CREGION), and it is shown in Figure 11.5. Note that the tuple of story county in Figure 11.2 has now split into two tuples. In general a tuple for SpaSQL would split into an unbounded number of tuples for PSQL. We explain the adverse implications of phenomenon through several examples.

Example 11.4.1 First we consider the query *retrieve complete information about counties which grow wheat or corn.* In SpaSQL this query is expressed as follows.

SpaSQL: select *
from county
where $[\![\text{CROP} = \text{wheat}]\!] \neq \phi$ or $[\![\text{CROP} = \text{corn}]\!] \neq \phi$.

To express this query in PSQL we need two variables x and y. The variable x is needed to make sure that the crop is wheat, and y is needed to make sure that the county is same as that in x. Note that the from clause "from county x y" in the following expression is meant to create two aliases of the county relation. The PSQL expression is as follows.

PSQL: select $x.*$
from county x y
where (x.CROP = wheat or x.CROP = corn)
and x.CNAME = y.CNAME

Example 11.4.2 As our next example, we change *or* to *and* in the English query of Example 11.4.1. The resulting query is retrieve *complete information about counties which grow wheat and corn.* This is expressed in SpaSQL simply by replacing *or* in $[\![\text{CROP} = \textit{wheat}]\!] \neq \phi$ or $[\![\text{CROP} = \text{corn}]\!] \neq \phi$ by *and.* Thus the SpaSQL expression is as follows.

SpaSQL: select *
from county
where $[\![\text{CROP} = \text{wheat}]\!] \neq \phi$ and $[\![\text{CROP} = \text{corn}]\!] \neq \phi$.

The corresponding transformation does not work in PSQL. To express the new query in PSQL, we need three independent variables, and it is expressed as follows.

PSQL: select $x.*$
from county x y z
where (x.CROP = wheat and y.CROP = corn)
and x.CNAME = y.CNAME and x.CNAME = z.CNAME

Note that for every occurrence of *and* in an English query of the form given above, we need an additional variable in the PSQL query. Thus if there are n properties to be checked for a given county, we need $n+1$ variables in PSQL leading to an $(n+1)$-way join; in SpaSQL simply one variable suffices.

Example 11.4.3 Now we consider *not* of natural languages. First consider the query *retrieve information about counties that grow wheat.* In SpaSQL and PSQL it is expressed as follows.

SpaSQL: select *
from county
where $[\![\text{CROP} = \text{wheat}]\!] \neq \phi$.

PSQL: select $x.*$
from county x y
where x.CROP = wheat and x.NAME = y.CNAME

Now we insert a *not* in the English query. The new query is *retrieve information about counties that do not grow wheat.* In SpaSQL it is simply expressed by replacing

$[\![\text{CROP} = \text{wheat}]\!] \neq \phi$ by $\neg[\![\text{CROP} = \text{wheat}]\!] \neq \phi$, or equivalently, by $[\![\text{CROP} = \text{wheat}]\!] = \phi$. But it is more complex in PSQL. It needs the difference operator to express it.

SpaSQL: select *
from county
where $[\![\text{CROP} = \text{wheat}]\!] = \phi$.

PSQL: (select *
from county)
difference
(select x.*
from county x y
where x.CROP = wheat and x.CNAME = y.CNAME)

Thus we see that the complexity of a PSQL expression increases with every occurrence of *and* and *not* in the given natural language query.

11.5 Algebraic Nature of SpaSQL

SpaSQL is algebraic in nature. In this section we show how the counterparts of usual operators of the classical relational model can be captured as additional operators in SpaSQL. Then we show how operators in [499] can be expressed in our model.

We note that we have already introduced (i) union and (ii) difference operators. (iii) Projection $\Pi_X(r)$ is equivalent to the select statement "select X from r." (iv) Our selection is of the form $\sigma(r; f; reg)$ and it is equivalent to the select statement "select * from r restricted_to reg where f." (v) The join $r(AB) \bowtie s(BC)$ is equivalent to the select statement "select ABC from r, s restricted to $[\![r.B = s.B]\!]$."

Scholl and Voisard [499] introduce some interesting operators for handling spatial data. Without going into considerable details we show how these operators may be expressed in SpaSQL.

1. Projection. is the usual projection operator, and it is covered above.
2. Cover. Their cover(r) is our $[\![r]\!]$, where r is a spatial relation.
3. Map overlay. Their $r \times_G s$ is a homogeneous cross product. Assuming that the schemes of r and s are R and S, respectively, and that R and S are disjoint, this operator is expressed in SpaSQL as "select R, S from r, s."
4. Selection, clipping and window. Their selection, clipping, and window operators are incorporated in our selection operator given above.
5. Superimposition. The effect of r superimpose s is similar to our $(r - s) \cup s$, if r and s have the same scheme. However, the superimpose operator of [499] can be used in more complex ways. For example, superimposition can be used to caption a relation (map) r with a set of labels s.

11.6 Conclusion

In this chapter we have presented a model and a query language for spatial data. The query language is perhaps as seamless as possible. SpaSQL can serve as a powerful query language for retrieval of spatial data. In our model we used some ideas from

temporal databases. We feel that the work presented here only serves as a starting point, because spatial databases are far more complex than temporal databases. This complexity basically arises from the underlying spatial domains. The implementation of spatial databases is truly a challenging problem. It should be noted that the implementation of a spatial relation can be different from its logical representation. It is possible, for example to implement our logical spatial relations as the spatial relations used by PSQL. We hope that this model will serve as a basis for further research. There is considerable similarity between spatial and temporal data. Perhaps a more appropriate term for all these forms of data is *parametric data*. Parametric data have the notion of an underlying parameter space. In case of spatial and temporal data, the parametric spaces consist of spatial points, and time instants, respectively. The model and SpaSQL presented in this chapter have been generalized to parametric databases in [204]. We have also given an object oriented model for parametric databases in [109].

Chapter 12

Parallel Query Processing

P. S. Yu*, M.-S. Chen†, J. L. Wolf‡, J. Turek§

12.1 Introduction

With the advent of inexpensive microprocessors and high bandwidth interconnects, coupling a large number of processors to form a highly parallel system has become an increasingly popular method for improving the cost-performance ratio of computer systems [129, 130, 353, 380, 577]. Recent work has shown that this method is also applicable to database systems with increasing benefits as the queries become larger and more complex. The objective of this chapter is to examine the various issues encountered in parallel query processing as well as the techniques that are available for addressing these issues.

Research on parallel processing over the last two decades has mainly focused on scientific applications. In the past few years, there has been a growing interest in applying general purpose parallel machines to database applications [94, 104, 260, 360, 361, 469]. Several research systems have also been developed to explore this trend, including GAMMA [155], XPRS [555], DBS3 [46], and BUBBA [72]. Relational databases have a certain natural affinity to parallelism. Relational operations are set oriented and this provides the query optimizer lots of flexibility in selecting the parallelizable access path. (This is in contrast to navigational type databases such as those using the hierarchical and network paradigm.) Also, the parallelism tends to be coarse grained, i.e., each task will involve a lot of I/O and processing, with a well-defined and, in general, infrequent communication pattern between the tasks. The difficulties associated with query parallelism arise because of the less than perfect predictability of task times (primarily due to a dependency on the data itself) and the enormous search space that the query optimizer needs to consider in order to make its access plan selection.

As pointed out in [498], the methods for exploiting parallelism in a database environment can be divided into three categories: namely *intra-operator, inter-operator,*

*IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.

†IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.

‡IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.

§IBM Research Division, Thomas J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A.

and inter-query parallelism. First of all, parallelism can occur in each operator within a query in such a way that multiple processors work concurrently on a single database operation. Here we will be concentrating on the join operations, as they are the most expensive operations to execute, and also the most difficult to parallelize properly. In intra-operator parallelism the major issue is task creation, and the objective is to split an operation into tasks in a manner such that the load can be spread evenly across a given number of processors. The second form of parallelism is termed inter-operator parallelism, meaning that several operators within a query can be executed in parallel. This can be achieved either through parallel execution of independent operations or through pipelining,[¶] where multiple joins are pipelined so that the early resulting tuples from a join can be sent to the next join for processing simultaneously. In either case, the major issues are the *join sequence selection* and *processor allocation* for each operation. Join sequence selection, which is also called query plan generation, determines the precedence relations among the operations. Thirdly, parallelism can be achieved by executing multiple queries simultaneously within a multiprocessor system. This is termed inter-query parallelism. For inter-query parallelism, the issue again is processor allocation, but among the multiple queries.

Various techniques to address each of these issues have been proposed in the literature, albeit under different assumptions and generally with a focus on only one of the issues. In this chapter, we explore the query parallelism based on a hierarchical approach and a unified framework, so that the potential integration of the techniques used to address each type of parallelism can be illustrated. Both sort-merge joins and hash joins are considered.

Due to the complexity of the problem, join sequence selection is still mainly based on heuristics [105, 107], and there is generally no formal analysis (such as worst case bounds) on the suboptimality of these heuristics. However, given a join sequence, the processor allocation problem can generally be formulated as an optimization problem [376, 628], which can often be solved by the use of *resource allocation problem* methods [270] (which we note in passing also have a wide range of applicability to various problems in computer science [284, 549, 574, 626]). Generating the optimal join sequence and processor allocation together through exhaustive search can be prohibitively expensive for complex queries, due to the enormously large design space. This is generally not attempted in the literature even for the sake of validating the optimality of the heuristics.

Preliminaries are given in Section 12.2. The various issues encountered in parallelizing query processing are presented in Section 12.3. In Section 12.4, we discuss the implication of the different parallel processing architectures. The various techniques to address intra-operator, inter-operator, and inter-query parallelism are then considered. Section 12.5 addresses the issue of data skew on intra-operator parallelism. We then consider parallelism for multi-join queries in Section 12.6. Parallelism in a multiple query environment is explored in Section 12.7. Finally, Section 12.8 provides a summary.

12.2 Preliminaries

In this chapter, we assume that a query takes the form of conjunctions of equi-join predicates. A join query graph can be denoted by a graph G = (V, E), where V is the

[¶]Pipelining is also referred to as data-flow scheduling [614].

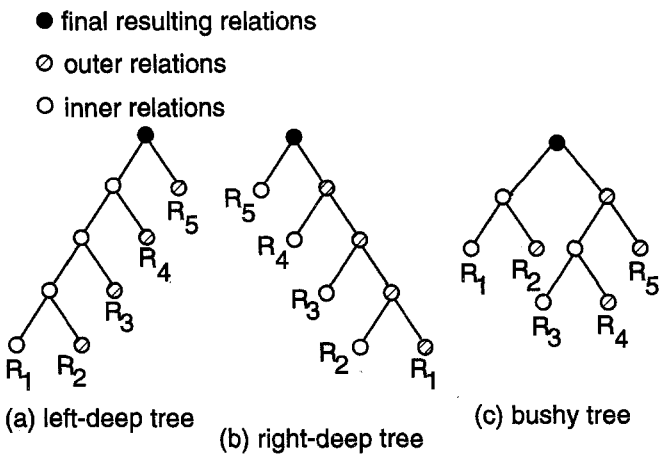


Figure 12.1: Illustration of different query trees.

set of vertices and E is the set of edges. Each vertex in a join query graph represents a relation. Two vertices are connected by an edge if there exists a join predicate on some attribute of the two corresponding relations. We use $|R_i|$ to denote the cardinality of a relation R_i and $|\mathcal{A}|$ to denote the cardinality of the underlying domain of an attribute $\mathcal{A}$.

The execution of a query can be denoted by a query execution tree, which is determined by the join sequence selected by the query optimizer. Three forms of query execution trees are explored in the literatures: *left-deep trees*, *right-deep trees*, and *bushy trees*. We note that left-deep trees and right-deep trees are sometimes referred to as *linear trees* as they can be represented by a permutation of the vertices. In a query tree, a leaf vertex represents an input relation, an internal vertex represents the relation resulting from joining the two relations representing its child vertices, and the query tree is executed in a bottom up manner. Conventionally, the left and right child vertices of an internal vertex denote, respectively, the inner and outer relations of a join [497]. In the context of hash joins (as explained in Section 12.2), the inner relation is the relation used to build the hash table and the outer relation is the one whose tuples are applied to probe the hash table. Examples of the three forms of query trees are shown in Figure 12.1, where the inner and outer relations are indicated for illustration.

12.3 Issues

We next examine the issues encountered in applying the three types of parallelism. In later sections, some techniques to address these issues are presented.

12.3.1 Intra-operator Parallelism

Exploiting intra-operator parallelism requires that the join operations be divided into multiple subtasks that can be run simultaneously on the various processors. The effectiveness of such an approach depends upon the ability to equally divide the load among the processors while simultaneously minimizing the coordination and synchronization overhead. A factor which can impair the ability to parallelize join operations successfully is the amount of skew present in the data to be joined.

In real databases it is often found that certain values for a given attribute occur more frequently than other values [63, 111, 259, 415]. [63] notes, for example, that for many textual databases the data distribution follows a variant of Zipf's Law [646], and can thus be characterized as *Zipf-like* [329]. This nonuniformity is referred to as *data skew* [344]. It is inherent in the data itself and does not depend on the access pattern. Since identical values always are hashed to the same hash bucket, this problem *cannot* be remedied by an appropriate choice of hash function. Similarly, although they are important concepts, neither bucket tuning [324] nor bit filters [596] will help with this particular problem.

The problem is exacerbated for join operations as opposed to sorts because correlation in the data skew of each relation results in a join output that is quadratic in nature. Previous studies on join performance have largely ignored this phenomenon and assumed uniform distribution of data, thus overestimating the potential benefit of parallel query processing using conventional join algorithms. Lakshmi and Yu [344] were the first to observe that in the presence of data skew the speedup from conventional join algorithms can be very limited. This is because the data skew can result in some processors being overutilized while others are underutilized. Even one frequently occurring (or *large skew*) value can cause the processor to which it is assigned to become overloaded. In [498] some aspects of data skew on parallel join methods are considered. However, in their study the case where both relations to be joined have data skew (*double skew*) was explicitly not examined. Some examples of skew in the distribution of join column values from real workloads can also be found in [624].

12.3.2 Inter-operator Parallelism

Effective exploitation of inter-operator parallelism for the execution of a multi-join query depends primarily on the following two major issues: (i) join sequence selection, i.e., scheduling the execution sequence of joins in the query, and (ii) processor allocation, i.e., determining the number of processors for each join obtained in (i) so that the execution time required for the query can be minimized. Note that different execution sequences of joins in a query will result in different execution costs [511], and some sequences may be more parallelizable than others. Also, in a multiprocessor system the execution time of each join strongly depends on the number of processors allocated and the way intra-operator parallelism is handled.

Note that in the study of intra-operator parallelism, the objective is usually to determine the processor allocation which achieves the minimum execution time of a single join operation. Such a selection is referred to as *operational point selection.* In exploiting inter-operator parallelism, we are, in contrast, dealing with the execution of a complex query consisting of multiple joins, where different joins are allowed to be executed in parallel in different clusters of processors. To minimize the execution time of a multi-join query, one needs to take into consideration not only the operational point selection (as in the study of intra-operator parallelism), but also factors such as *execution*

dependency and *system fragmentation*. Execution dependency means that some joins cannot be performed until their operands generated by prior joins are available. Also, after a sequence of processor allocations and releases, there might be a few processors left idle since they do not form a cluster large enough to execute any remaining join efficiently. This phenomenon is termed system fragmentation [106]. Clearly, execution dependency and system fragmentation as well as the operational point selection have to be taken into account for a better processor allocation efficiency, thus complicating the minimization procedure for the query execution time.

Depending upon the join methods selected, different types of inter-operator parallelism can be explored. With hash joins, an additional form of inter-operator parallelism, pipelining, can be used, while with sort-merge joins, only parallelism of independent join operations can be employed.

The execution of a hash join consists of two phases: the table-building phase and the tuple-probing phase. In the table-building phase the hash table of the inner relation is built according to the hash function of the join attribute, and in the tuple-probing phase the hash function is applied to each tuple of the outer relation and the resulting hash value is used to probe the hash table of the inner relation for matches. Recall that in the context of hash joins, the left and right child vertices of an internal vertex in a query execution tree denote, respectively, the inner and outer relations of a join. It can be seen that in a left-deep tree, the result of a join is used to build the hash table for the next join, and all hash joins thus need to be executed sequentially. In contrast, in a right deep tree all the hash tables are built from the original input relations, and the resulting relation of a join is input into the next join as an outer relation. The tuples of the outer relation can thus be pipelined through the entire right-deep tree. Clearly to maximize the throughput of the pipeline execution, we need to allocate the processors so that all stages in the pipeline can proceed at the same pace: There should be no bottleneck stages.

The bushy tree, on the other hand, is not restricted to a linear form, meaning that the resulting relation of a join in the bushy tree does not need to be immediately used in the next join. The resulting relation of a join can in fact be used as either an inner or an outer relation for subsequent joins. This further complicates the problem and is explored in Section 12.6.

12.3.3 Inter-query Parallelism

The basic issue is how to schedule the multiple queries. Each query can involve multiple joins and take any number of processors to execute. There has been very little work done on this subject. An exception is [628], where various approaches to inter-query parallelism are studied. One of the suggested approaches is to first allocate the processors at the query level and then devise a sub-schedule for inter-operator parallelism to allocate these processors among the different operations within the query. This is essentially a hierarchical approach. Since each operation, and hence each query, has different operating efficiencies depending on the number of processors allocated, the operational point selection and system fragmentation have to be taken into account in processor allocation at the query level. This is similar to inter-operator parallelism. However, in contrast to the case of inter-operator parallelism, there is no execution dependency or precedence relations among the queries. This type of processor allocation algorithms corresponds to a *malleable task* scheduling problem [588, 589], an extension to the classical multiple processor scheduling problem [27, 127] with the additional twist that the number of processor allocated to each task is now a variable to

be optimized. We will discuss this in Section 12.7 together with other variations on hierarchical approaches to inter-query parallelism.

12.3.4 Remarks

We take a hierarchical approach on addressing the various issues on query parallelism. The inter-operator parallelism needs to address the join sequence selection and processor allocation issues. This can be built on top of the algorithm for intra-operator parallelism, which focuses on balancing the load across the multiple processors allocated to each join operation. As discussed in the previous subsection on inter-query parallelism, alternative ways can be devised to exploit the algorithms on inter-operator parallelism and decide the processor allocation among the queries.

Finally, we comment on the CPU vs. IO requirements for parallelism. So far we have used the term *load* in a loosely defined fashion so that it can either mean CPU load or IO load. Most prior work assumes that the complex queries are CPU-bound and the focus is on balancing the CPU load. However, this can be regarded essentially as an issue of objective function selection. By properly defining the objective function to be IO load for IO-bound cases and CPU load for CPU-bound cases, the algorithms described in this chapter will work in either situation. For the case where some of the queries are CPU-bound and some are IO-bound, a composite objective function can also be devised [254, 255].

12.4 System Architectures

Here we examine different processor coupling architectures. In database environments, it is customary to classify the architectures according to the level of the memory hierarchy being shared.

In the shared everything (SE) architecture [62], which is also referred to as the tightly coupled processor architecture, all processors share a common main memory under a single copy of the operating system.

In the data sharing architecture, each node runs an independent operating system and all the processing nodes have direct access to all granules in the database. There are two variations. One is the shared disk (SD) architecture [410, 559, 641], where all disks are shared. The other variation is to have a shared intermediate level of memory between the main memory and disks. This is referred to as the shared intermediate memory (SIM) architecture [158, 323, 462, 639, 640]. The basic structure of SIM is otherwise the same as the SD architecture.

Finally, there is a shared nothing (SN) architecture [551]. Under this approach, the database is partitioned among multiple nodes, and can only be accessed directly by the owning node. This is in sharp contrast to the data sharing approach. A high speed interconnect is generally provided to facilitate message and data exchanges between the nodes.

In the rest of the chapter, we will assume an SD architecture when presenting the various parallel join algorithms. We briefly comment the general implications if other architectures are used and some additional comments are given in the later sections where appropriate. In the SN environment, there is an additional issue of data to processor affinity. If a task is not scheduled on the processor where the input data resides, an additional data transfer phase is required. This is in contrast to the SD architecture, where all disks are equally accessible by any processors, and hence there is no data to

processor affinity. This affinity issue is generally not pursued in the literature, even when assuming an SN architecture. Ignoring this issue, parallel join algorithms based on SD architectures are directly applicable to SN. The SE architecture offers more flexibility for dynamic load balancing or task scheduling than the SD architecture, since all processors share a common task queue. The SIM architecture, to a lesser extent, also provides flexibility in task scheduling. Aside from this additional flexibility, the parallel join algorithms for SD architectures are again applicable.

There is an additional issue involving the structure of inter-processor connections. In a symmetric structure, the communication delays between the various pairs of processors are roughly the same, as is the overhead. In an asymmetric structure, these delays and overheads can be different for different pairs. For example, in a hypercube like structure, it may be preferable to schedule the tasks related to the same database operation or the same query in the same low dimensional sub-cube so that the communication penalty is minimized [37, 197, 437]. However, with the advent of high speed interconnects and the communication processors to route data blocks, the difference between single hop and multiple hop communication becomes less significant. In this chapter, we will assume a symmetric structure.

12.5 Data Skew and Intra-operator Parallelism

Examples of relational database operations include joins of two relations, and scans or sorts of a single relation. Typically such operations are performed in several phases, even in uniprocessor environments. For example, a sort merge join might consist of a sort phase, in which sorted runs of each of the relations are generated, followed by a join phase in which the sorted runs are merged and joined. Similarly, a hash join might consist of hash and join phases. A sort might consist of sort and merge phases. Of these operations, only a scan can generally be performed in a single phase.

Effectively, all phases associated with a database operation must be performed in sequence, since one phase has precedence over the next. In a parallel environment, therefore, a reasonable goal is to partition the total work associated with each of the phases into subtasks in such a way that the processors complete their respective subtasks in approximately the same amount of time. This is called *load balancing*. For all practical purposes, balancing the load is equivalent to minimizing the *makespan* of the phase, namely the total processing time of the last processor to complete its assigned subtasks.

In this section we shall discuss several approaches to handling the problem of load balancing in the join phase of a single parallel join of two relations. These approaches span the spectrum in terms of sophistication, and in terms of their ability to handle the data skew problem. We focus on joins in general, and the join phase in particular, because this phase is the most difficult one as far as the load balancing in the presence of data skew is concerned. However, the algorithms presented have natural (and simpler) analogues to other phases of joins and other operations, and we shall point these out where they are interesting. Similarly, we shall concentrate on sort merge joins rather than hash joins: Again, analogous algorithms exist for hash joins, and we will point them out where appropriate. Finally, different architectures can also be handled within the general context of these algorithms. Much of the work in the literature has focused on the SN rather than the SD architecture. In an SN architecture, tuples may need to be physically moved from the disks of one processor to those of another, in order to be in the proper place for the next phase. Thus, for SN environments, a so-called transfer

phase may also be required. In an SD environment, however, which we are assuming in this chapter, this is obviously not necessary.

Suppose we wish to perform an equijoin two relations R_1 and R_2, and suppose the join column attributes are from some common underlying domain. The first approach, which we will call a *conventional* join, ignores the data skew problem completely. This algorithm partitions the join phase work into subtasks based on the underlying domain only, and the number of subtasks is equal to the number P of processors employed. The second approach, which we will call *dynamic*, is a bit more sophisticated. It also partitions the join phase work based on underlying domain considerations, but more than P subtasks are created in an attempt to balance the load. Finally, the most *sophisticated* algorithms employ an additional phase of some kind, in an attempt to gather statistical information about the relations themselves. Said differently, these algorithms are therefore capable of partitioning the join phase work based on estimates of the number of tuples rather than just the number of distinct values in the underlying domain. Accordingly, they are able to achieve consistently better load balancing in the presence of data skew.

In order to be as concrete as possible, we shall now introduce a simple example of the join of two relations. The same example will be used throughout this section in order to illustrate the approach and performance of the various join algorithms. Consider Table 12.1. There are 12 distinct values in the underlying domain of the join columns of the two relations R_1 and R_2. These values are labeled A through L. Table 12.1 shows the tuple count for each value and each relation. Notice, for example, that value G occurs 10 times in R_1 and 2 times in R_2. Similarly, the value B occurs 0 times in R_1 and 1 time in R_2. Again, the nonuniform distribution of tuple cardinalities is known as *data skew*. (Since this nonuniformity occurs in both relations, we call this an instance of *double* data skew.) For simplicity, we shall assume that there are no predicates involved in this join. Table 12.1 also shows the tuple count for each value in the joined relation $R_1 \bowtie R_2$. Notice that the values G and B occur 20 and 0 times, respectively. In [623, 624] a formula is given which accurately measures the amount of join phase work (time) associated with each distinct value in either a CPU- or IO-bound environment, but for the sake of exposition here we shall content ourselves with a very elementary approximation: Specifically, let us assume that the amount of work associated with a value v is given by $TIME_v = c_{v,1} + c_{v,2} + c_{v,1}c_{v,2}$, where $c_{v,1}$ is the number of tuples taking on value v in R_1 and $c_{v,2}$ is the number of tuples taking on value v in R_2. (The first two summands provide a simple measure of work done on input, while the last summand provides a measure of work done on output.) Thus $TIME_G = 10 + 2 + 10 * 2 = 32$, while $TIME_B = 0 + 1 + 0 * 1 = 1$, as is indicated in Table 12.1. (Ordering the values by this measure of work, we will call G the *largest* skew value in this example, and B the *smallest*.)

In our example, the sort merge join of R_1 and R_2 will be performed by $P = 4$ processors. In the sort phase, each processor will generate sorted runs of portions of both relations. Load balancing of this phase will ensure that the processors complete at roughly the same time. (In an SD environment, effective load balancing of the sort phase is quite easy to achieve: We will comment a bit more on this issue shortly.) Figure 12.2 shows a possible outcome of the sort phase. There are 4 sorted runs for R_1, each created by one of the processors, each of approximately the same size. A similar statement holds for R_2.

Our example gives an indication of just how serious the data skew problem is for joins: Notice from Table 12.1 that the join phase work associated with value G is more than a quarter of the total amount of join phase work. Given $P = 4$ processors,

Value	Tuple Cardinalities			Join Phase
	R_1	R_2	$R_1 \bowtie R_2$	Work
A	2	3	6	11
B	0	1	0	1
C	5	1	5	11
D	1	1	1	3
E	1	2	2	5
F	1	1	1	3
G	10	2	20	32
H	1	1	1	3
I	4	2	8	14
J	2	2	4	8
K	1	4	4	9
L	2	3	6	11
Total	30	23	68	111

Table 12.1: Join Example

1	2	3	4	1	2	3	4
C	A	A	C	A	A	E	B
C	C	G	D	A	E	F	C
F	C	G	E	H	J	J	D
G	G	G	G	I	K	K	G
G	I	G	G	I	K	L	G
G	I	I	H	L	K		L
J	I	J	K				
		L	L				

Figure 12.2: After Sort Phase

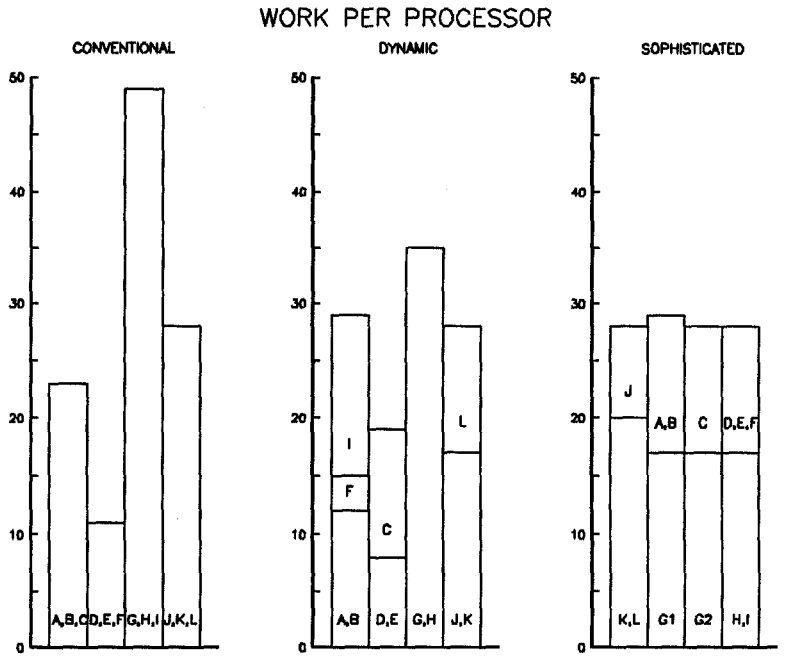


Figure 12.3: Join Phase Load Balancing

effective load balancing will not be easy. Obviously, the problem grows with the number of processors and the degree of data skew. The problem is particularly exacerbated for joins, because of the quadratic nature of the join output size.

12.5.1 Conventional Algorithm

A conventional sort merge join algorithm attempts to balance the load across the processors for the join phase by an elementary range partitioning algorithm. For this example, since there are 12 distinct values and 4 processors, the algorithm might simply assign the first three values (A,B,C) to processor 1, the next three (D,E,F) to processor 2, the next three (G,H,I) to processor 3, and the final three (J,K,L) to processor 4. Unfortunately, though the number of distinct values is precisely balanced, that number is not a good measure of the join phase work. This is shown in the left third of Figure 12.3. We note that the makespan for the join phase is 32+3+14=49, corresponding to processor 3. Note how unbalanced the processors are: While processor 3 is heavily overutilized, processor 2 (with 11 units of work) is heavily underutilized. Processor 1 (23 units) is also somewhat underutilized, since a properly load balanced processor would perform (23+11+49+28)/4=27.75 units of work. Only processor 4 (28 units) is more or less properly utilized. The poor load balancing is directly due to the data skew in the relations. In fact, as previously indicated, *any* processor handling the value G would automatically become overloaded.

There exists a corresponding conventional parallel hash algorithm, and it suffers from exactly the same problem. See [624] for details.

12.5.2 Dynamic Algorithms

Notice that the assignment of a single subtask to each processor in the conventional algorithm is highly inflexible, in that each processor is *statically* committed to perform a fixed unit of work. This observation suggests a relatively simple fix: Namely, create *more* than P subtasks, and assign new subtasks to processors *dynamically* as they complete their previously assigned subtask. This idea is originally due to [323, 325] in the context of a parallel hash join, and it obviously works especially well in an SD environment.

In its simplest variant, the underlying concept of the dynamic algorithm is to partition the join phase work into $N > P$ subtasks, each with the same number of values in the underlying domain. (In [325] the choice is based on the expectation that the tuples associated with each subtask should fit in processor memory, obviously a good idea.) The subtasks are then placed in an ordered list. The ordering of this list is not important, since the expected amount of work associated with each subtask will be the same. The first P subtasks on the list are initially assigned to each of the P processors. Subsequently, as each processor completes a subtask, it is assigned the next subtask on the list. Eventually, the list becomes empty, and the join phase is completed.

A refinement of this approach appears in [627]. The idea is to partition the join phase work into KP subtasks having *purposely* non-equal subtask work estimates. (Here K is some relatively small integer greater than 1.) For example, a reasonable approach would be to start with the range partitioning described in the conventional join algorithm, and further partition each of the original ranges into K new ranges with the property that the kth such range has approximately *half* the number of underlying domain values as the $(k-1)$st such range. The list of KP subtasks is now ordered according to the number of values in the underlying domain, from largest to smallest, and the algorithm proceeds as before. Effectively, this algorithm is a *dynamic* version of the standard *longest processing time* first algorithm [228]. We therefore call it DLPT. (LPT is itself an algorithm for heuristically solving the optimization problem known as the *minimum makespan* or *multiprocessor scheduling* problem, so its use here should not be surprising.) Although the subtask times have not been estimated with perfect precision, the ordering of the subtasks and the flexibility inherent in a dynamic algorithm help to limit the join phase load imbalance.

We illustrate the DLPT sort merge join algorithm via our running example. Here, we choose $K = 2$, and we create 4 subtasks with two values each, and 4 more with 1 value each. The ordered list is ({A,B},{D,E},{G,H},{J,K},{C},{F}, {I},{L}). See the middle third of Figure 12.3. The first 4 of these are assigned to processors 1 through 4. Processor 2 is the earliest to complete, and it is therefore assigned the subtask associated with value C. Processor 1 completes next, and is assigned the subtask associated with value F. Processor 1 completes next again, and is assigned the subtask associated with value I. Finally, processor 4 completes, and is assigned the subtask associated with value L. The join phase makespan using this dynamic algorithm is 32+3=35, associated with processor 3. This processor has worked only on the subtask associated with values G and H, one of which, not surprisingly, is the largest skew value. Still this makespan compares favorably with that (49) using the conventional algorithm, and the load balancing is also correspondingly more uniform.

Obviously, the same underlying problem remains: The subtask associated with the value G will automatically cause the processor to which it is assigned to become overutilized. Fixing this problem requires more sophistication, and that is the subject

of the next subsection.

We should point out that the dynamic algorithm described above is really a special case of a very generic algorithm: The reader can easily imagine how such an algorithm could be applied to load balance the first phase of a join, sort or scan. In fact, [268] recently described the same basic algorithm for an entirely different application, scientific as opposed to database. They call their algorithm *factoring*.

12.5.3 Sophisticated Algorithms

We shall now give an overview of an algorithm we call SMJ* [623]. In an SD environment this algorithm adds a *scheduling* phase between the sort and join phases. The scheduling phase creates an ordered list of join phase subtasks, in an attempt to balance the load for that phase.

In order to do a good job of load balancing, R_1 and R_2 are partitioned into some number of regions, typically larger than P. Each region will correspond either to a contiguous multiple range of values (which we will call a *type 1* region), or a single value (which we will call a *type 2* region). We have already commented that even a single large skew value can cause a processor to become overloaded. If we can identify and isolate such a type 2 region, we can give it special treatment: Specifically, we can assign it to an optimal number of processors. (One approach to joining a single value across M processors would be to partition the tuples with this value in R_1 into M equal size pieces, shipping each such piece to *one* of the M processors. The tuples with this value in R_2 would then be shipped to *all* of the M processors, this redundancy incurring some additional overhead, and the join would be completed correctly. Alternatively, we could swap the roles of R_1 and R_2.) Thus, associated with any type 2 region will be some currently optimal number of subtasks. This number, called the *multiplicity* of the type 2 region, will be less than or equal to P, but possibly greater than 1. Type 1 regions will always be assigned to a single processor, will correspond to a single subtask, and will be said to have multiplicity 1. The total number of subtasks N created over all the regions will be the sum across all of the regions of their associated multiplicities.

SMJ* will perform something of a juggling act. The initial step is the creation of a single region, corresponding to all possible values in the range of both relations, and presumably of type 1. From then on, the algorithm consists of the iterative application of the following 3 step process:

- computing good subtask time estimates and determining optimal multiplicities for the type 2 regions,
- generating an assignment of subtasks to processors in a way which achieves satisfactory join phase load balancing, if possible, and
- if the load balancing is unsatisfactory, partitioning the type 1 region with the largest subtask time estimate into (2 to 3) new regions, at least one of which will be of type 2, and repeating the 3 steps. (We hope to identify the largest type 2 regions during the repeated execution of this last step.)

It is the last two steps which form the mathematical foundation of SMJ*. In particular, step 2 is accomplished using LPT [228]. Step 3 is accomplished using a variant of a somewhat less well-known algorithm, which we will dub GM in honor of Galil and Megiddo [206]. The original version of this algorithm was designed by these authors to solve a special case of the so-called *selection problem*.

The selection problem solved by GM finds the jth smallest element in a matrix whose columns are monotone non-decreasing. What is the connection between our

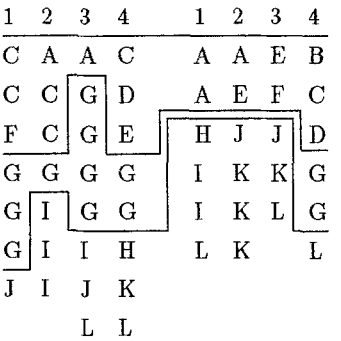


Figure 12.4: Snapshot After First GM Iteration

problem and this selection problem? The idea is simply this: We wish, in step 3 of SMJ*, to split the largest remaining type 1 region up into smaller pieces, hopefully identifying a large type 2 region in the process. It would also be a good idea to make this split as even as possible with respect to estimated subtask times. Our approach is to determine the *median* value of the $2P$ sets of tuples in the largest type 1 region. (There is one set of tuples for each processor and each relation.) But, because the tuples are *sorted*, a selection problem algorithm applies, with a few minor modifications. The GM algorithm has two very nice properties: First, GM automatically identifies, by binary searches in the algorithm itself, the set of all tuples whose value *equals* the median value. This set becomes the new type 2 region. Secondly, GM parallelizes quite naturally, each processor handling 2 sets of tuples. Now the tuples in the original type 1 region have values which are either...

- Less than the median value,
- Equal to the median value (the new type 2 region), or
- Greater than the median value.

The original type 1 region is partitioned into 3 sets according to the above alternatives. The first of these sets may or may not exist. If it does, it will most likely correspond to a type 1 region. The same comments apply to the last of these sets. However, since the original region was of type 1, either one or the other of these regions must exist. Probably, both exist, and are of roughly equal size. Figure 12.4 presents a 'snapshot' of the GM algorithm of SMJ* at work. (Figure 12.2 can be thought of as representing the initial type 1 region.) In Figure 12.4, GM has determined the type 2 region (associated with the value G) corresponding to the median element. The area above the Gs is a type 1 region, as is the area below. The latter is the larger of the two. Hence, in the next iteration (not shown), GM will subdivide it and identify the type 2 region associated with the value J, and two new type 1 regions ({H,I}, {K,L}). One more iteration (also not shown) will partition the other original type 1 region, identifying the type 2 region associated with value C, and two new type 2 regions ({A,B}, {E,F}). At this iteration there are 7 regions in all, 3 of which are of type 2. In general the process will continue, identifying the largest skew values (among others) with high probability as it goes.

The right third of Figure 12.3 shows the performance of the SMJ* algorithm, under the assumption of three GM iterations and the creation of a single multiplicity 2 region (associated with the largest skew value G). In this case it is better to split the 10 tuples in R_1 with value G into two sets of 5 tuples each, shipping the 2 tuples in R_2 with value G to both processors. The cost for each subtask will then be 5+2+5*2=17, not quite as efficient as performing a single subtask of cost 30, but better than performing two subtasks of cost 10+1+10*1=21, as would be the case if the roles of R_1 and R_2 were swapped. The two subtasks are labeled G1 and G2 in Figure 12.3. Note that the makespan is 17+12=29, associated with subtasks {G1} and {A,B} on processor 2. Also note how close to perfect the load balancing has become.

For further details on SMJ*, including various technical improvements to the base algorithm, see [623]. A hash join variant (called HJ*) also exists [624], and both algorithms are compared in [625].

Recently, [157] has introduced yet another hash join algorithm of comparable sophistication for dealing with data skew. The basic approach is to introduce an initial *sampling* phase to the hash join. This sampling phase estimates the degree of skew in the two relations, and thus serves the same role as the scheduling phase in the SMJ* or HJ* algorithm. [623] contains a few remarks on the relative merits of these two basic approaches.

There also exist algorithms to handle load balancing in the merge phase of a sort on a single relation. For example, [284] is philosophically similar to SMJ*, employing a variant of a selection problem algorithm. (In general, the load balancing problem is technically easier for sorts than for joins, and the consequences of imperfect load balancing less severe.) Similarly, [156] employs a sampling phase for sorts comparable to that of [157] for joins. Each of these algorithms is quite effective at balancing the load.

12.6 Complex Multi-join Query

Recall that inter-operator parallelism means that different operators within a query can be executed in parallel by different clusters of processors. This implies that two opportunities for parallelism exist in executing a multi-join query: Not only can each join be implemented by many processors, but also several joins can be executed concurrently. Note that different execution sequences of joins in a query will result in different execution costs in terms of the amount of CPU and IO processing. Also, the execution time of a join closely depends on the number of processors allocated for the execution of that join. Thus, the subject of exploiting inter-operator parallelism for the execution of a multi-join query consists of the two major issues: (i) join sequence scheduling (or query plan generation), i.e., scheduling the execution sequence of joins in the query, and (ii) processor allocation, i.e., determining the number of processors to execute each join obtained in (i) so that the execution time required for the query can be minimized.

The join method can affect the optimization procedure to exploit parallelism. Under hash joins, we have the opportunity of using pipelining to improve the performance [154, 219]. On the other hand, the pipelining approach is not applicable when a join method like sort-merge is used. Note that pipelining causes the effects on join sequence scheduling and processor allocation to be entangled. As a result, join methods without and with pipelining will be discussed separately in this section.

12.6.1 Join Methods without Pipelining

Here we consider join methods without pipelining. The emphasis is to exploit inter-operator parallelism. We first present some heuristics for generating join sequences, including those for linear join sequences and for bushy trees. The issue of processor allocation will then be discussed. The results on join sequence scheduling and processor allocation can be combined to form a final schedule to perform a multi-join query.

Join Sequence Scheduling

Linear join sequence

We first consider linear join sequence which does not allow for the exploitation of inter-operator parallelism. In this case, the join operations must be executed in sequential order, although intra-operator parallelism can be applied to speedup each join operation. A significant amount of research effort has been spent in developing linear join sequences to improve the query execution time, with the focus mainly on uniprocessor environments. The work reported in [511] was among the first to explore linear join sequences for left-deep trees, and sparked many subsequent studies. Generally speaking, the linear join sequences can be obtained by the following two methods: (1) the greedy method, denoted by S_{GD}, and (2) the optimal permutation, denoted by S_{OPT}. (Here, the subscripts correspond to the methods used.) The greedy scheme S_{GD} can be outlined as follows. First, the scheme finds the join which requires the minimal execution cost. It then tries to join the composite with the relation which has the minimal-cost join with the existing composite. The above step is repeated until all joins are finished. It can be seen that the complexity of S_{GD} is $O(|V|^2)$. Heuristics other than minimal-cost can also be used in S_{GD} to derive linear join sequences. A sophisticated scheme (referred to as the KBZ heuristic) to optimize multi-join queries with an enlarged search space was proposed in [334]. The KBZ scheme can be viewed as consisting of a 3-level hierarchy, where the top level picks a spanning tree from the join graph, the next level selects an optimal root for a given tree through iteration and the bottom level generates the join sequence for a given rooted tree with the root as the first relation in the sequence. The resulting query plan is derived by employing the three respective algorithms. Furthermore, the benefit of using optimization techniques such simulated annealing and iterative improvement (by moving to a better performing neighboring state or join sequence iteratively until no improvement can be made) to tackle large search space for query optimization was studied in [570]. The work in [570] was further extended in [569], where three heuristic methods, namely the augmentation heuristic, the KBZ heuristic and local improvement, were evaluated comparatively. The augmentation heuristic is mainly a general version of S_{GD} with various alternative heuristics to choose the next join. Local improvement means that a solution sequence is improved by an exhaustive search in a small solution space, e.g. permutation of a subset or a small cluster of the relations in the linear join sequence. It was found in [569] that a simple method such as the augmentation heuristic can perform very effectively, whereas more elaborate techniques such as simulated annealing do not fare well. It was then speculated that until significant new insights into the characteristics of the search space are obtained, it will not be profitable to experiment with complex methods for optimization. Stonebraker et. al. proposed a two-step approach in [255] to optimize sequential or linear join plans, with the emphasis on exploiting intra-operator parallelism in each operator. In [255] a collection of good sequential plans was first obtained based on buffer space considerations, and then parallelization of this

collection of plans was explored. In [107] the optimal linear join sequence, S_{OPT}, was implemented by a dynamic programming technique for comparison purposes. It was reported that simple heuristics for linear join sequences have performance fairly close to S_{OPT}, agreeing with the remark made in [569].

Bushy tree join sequence

Bushy tree join sequences have not attracted as much attention as the linear ones in the last decade. This can be explained in part by the observation that in the past the power and size of a multiprocessor system was limited, and that the query structures were too simple. It is noted, however, that these two limiting factors have been largely negated by the rapid increase in the capacity of multiprocessors and the trend for queries to become more complicated [638]. This justifies the necessity of exploiting bushy trees.

Recently, the use of bushy trees for parallel query processing has attracted increasing attention. For illustrative purposes, a heuristic implemented in [107] based on executing the minimal-resulting-relation join to form a bushy tree is outlined below.

Scheme G_{MR}: /* A scheme to execute the join with the minimal resulting relation. */
begin
1. **repeat until** $|V| = 1$
2. **begin**
3. Choose the join $R_i \bowtie R_j$ from G=(V,E) such that $|R_i \bowtie R_j| = min_{\forall R_p, R_q \in V}\{|R_p \bowtie R_q|\}$.
4. Perform $R_i \bowtie R_j$.
5. Merge R_i and R_j to $R_{min(i,j)}$. Update the profile accordingly.
6. **end**

end

(In the G_{MR} scheme, the subscript MR stands for a cost function selecting "the join with minimal resulting relation".) Clearly this heuristic is greedy in that only "local optimality" is considered, and thus need not lead to the minimal cost join sequence. Note that G_{MR} is of complexity $O(|V||E|) < O(|V|^3)$, rather close to $O(|V|^2)$ required by S_{GD}. From the simulation in [107], G_{MR}, despite its simplicity, performs significantly better than S_{GD} and S_{OPT}, and results in join sequences whose execution costs are reasonably close to that of the optimal one.

As in the case for linear join sequences, there are many possible heuristics which can be used to generate bushy trees. Another approach dealing with inter-operator parallelism was presented in [381], where a greedy scheme taking various join methods (without pipelining) and their corresponding costs into consideration was proposed. The scheme in [381] is similar to G_{MR} above in that it has the nature of "query graph reduction", i.e., working in a cycle of selecting relations, joining and updating query graph, until all joins are completed. But it is different from the latter in that it tries to build a bushy tree level by level from a given query graph, where the level of an internal (join) vertex is computed as the longest path from a leaf vertex. At each step, the algorithm picks the maximum number of pairs of relations to fill up a level so that the cost function considered is minimized. Several relation selection heuristics based on different cost functions were proposed and evaluated in [381]. In addition, various query plans in processing multi-join queries in an SN architecture were investigated in

[497]. A combination of analytical and experimental results was given in [280] to shed some light on the complexity of choosing left-deep and bushy trees.

Processor Allocation for The Execution of Each Join

As pointed out earlier, to minimize the execution time of a multi-join query it is necessary to address the following three issues: operational point selection, execution dependency and system fragmentation. Note that the execution time required for a join operation within a multiprocessor system depends on the number of processors allocated to perform the join, and their relationship can be modeled by an operational curve, as evidenced in results on intra-operator parallelism [344, 622]. Basically, increasing the number of processors will reduce the execution time of a join until a saturation point is reached, above which point adding more processors to execute the join will, on the contrary, increase its execution time. This is mainly due to the combining effects of limited parallelism exploitable and excessive communication and coordination overhead over too many processors. An example of an operational curve for this phenomenon is shown in the solid curve in Figure 12.5, where a dotted (hyperbolic) curve $xy = 30$ is given for reference. (Note that the curve is determined by the algorithm for intra-operator parallelism.) In such a curve, the operational point chosen from the curve is generally between the point which minimizes the execution time of the join, referred to as the minimum time point, denoted by p_M, and the one which optimizes execution efficiency, i.e., minimizes the *work* which is the product of the number of processors and the execution time, referred to as the best efficiency point, p_B. For example, p_B=5 and p_M=16 for the operational curve in Figure 12.5. To improve the processor allocation efficiency, we not only have to utilize the information provided in the operational curve for the operational point selection, but are also required to comply with execution dependency and avoid system fragmentation as much as possible.

To determine the number of processors allocated for the execution of each join, the heuristics proposed in the literature can be divided into two categories: (1) processor-allocation first (PAF) approaches, where the processor allocation at each join operation is determined prior to the join sequence selection, and (2) join-sequence-scheduling first (JSSF) approaches, which generates the bushy tree first and then determines the processor allocation based on a given bushy tree.

The PAF Approach

The three heuristics described below were introduced in [107] to determine the processor allocation for each join operation belonging to this category. A join selection heuristic, such as the minimal-resulting-relation G_{MR}, is used together to determine the next join when building a bushy tree.

(a). Sequential execution (SE): This heuristic allocates all processors in the system to execute each join in the query sequentially. It can be seen that inter-operator parallelism is absent when this heuristic is used, and the join sequence is the key factor in the performance in such a case. A variation of SE is to allocate a fixed number of processors for the execution of each join, to avoid system fragmentation.

(b). Minimum time point (MT): This heuristic is based on the minimum time point in the operational curve, i.e., the number of processors used to execute the corresponding join operation is p_M.

(c). Time-efficiency point (TE): As can be seen in Figure 12.5, a scheme based on the

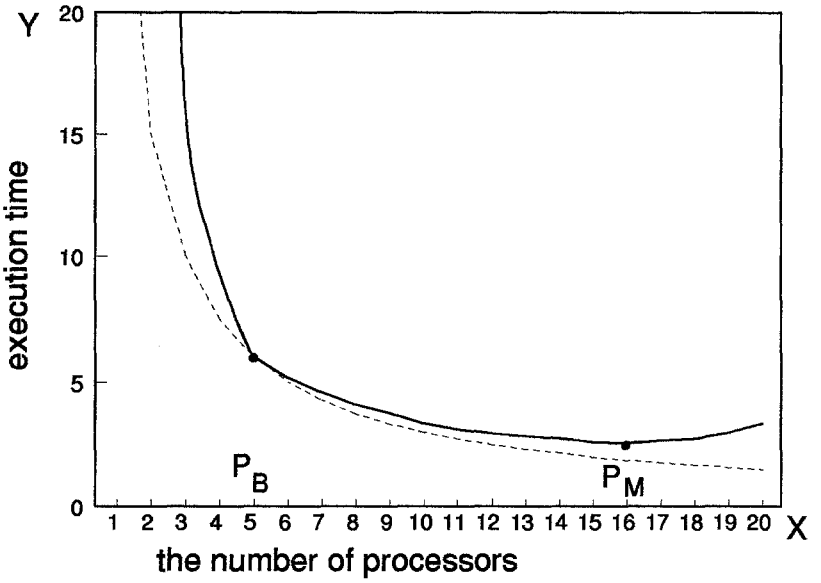


Figure 12.5: An Operational Curve of a Join

best efficiency point might suffer from execution dependency. This is because some join operating at its best efficiency point might take a long time to complete due to a small number of processors used to execute the operation, thus causing long waiting time for subsequent joins. On the other hand, a scheme based on MT may not use processors efficiently since it may require too many processors to reach the minimum time point. In view of this, a combination of the minimum time point and the best efficiency point, termed the time-efficiency point, is employed as a heuristic. In other words the number of processors $\alpha * p_M + (1 - \alpha) * p_B$ is used to execute each join operation, where $0 \leq \alpha \leq 1$.

Examples of SE and TE can be found in Figure 12.6a and Figure 12.6b, respectively. It can be seen that when an execution tree is built using the above approaches, the following two constraints have to be followed: (1) execution dependency is observed, i.e., the operands of the join selected to be performed next do not depend on the resulting relation of any ongoing join, and (2) the processor requirement is satisfied according to the processor allocation heuristic employed, i.e., the number of processors required by that join is not larger than the number of processors available then. Also, idleness of processors should be avoided.

The JSSF Approach

Several schemes, categorized as the JSSF approaches, were proposed to alleviate the above two constraints [107, 254, 381, 628]. In [381] a query tree is obtained first by the greedy method described earlier in Section 12.6.1. Then, processor allocation is done level by level via iterative refinements. Specifically, each join gets an initial processor

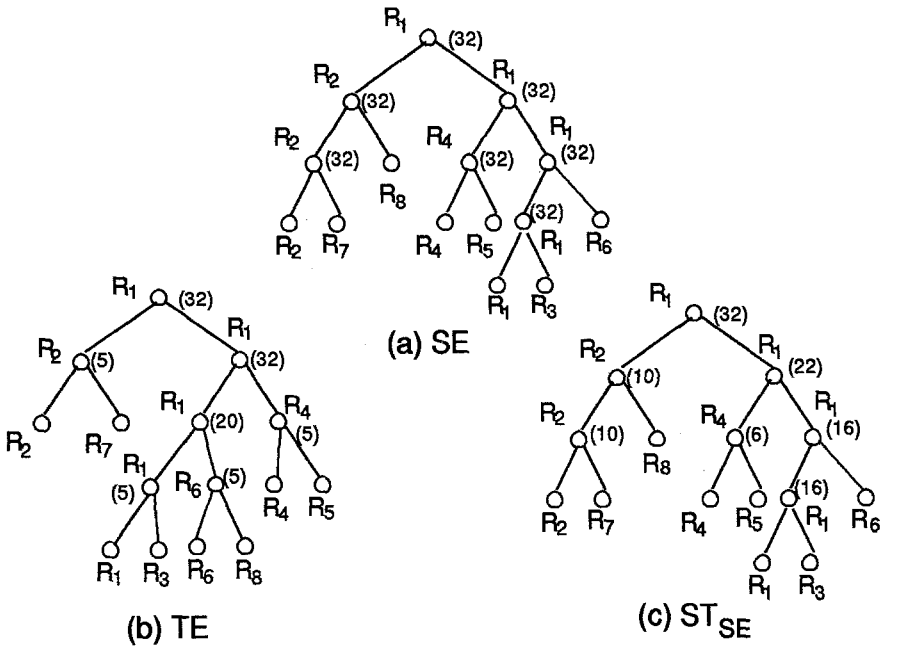


Figure 12.6: Processor Allocation

assignment. This assignment is then repeatedly adjusted by moving processors from less costly joins to more expensive ones to average the workload, until no further improvements can be made.

In [107] the number of processors allocated to each join in a bushy tree is determined top down based on the concept of synchronous execution time. Clearly, all processors are allocated to the join associated with the root in the bushy tree since it is the last join to be performed. Then, the processors allocated to the join on the root are partitioned into two clusters which are assigned to execute the joins associated with the two child vertices of the root in the bushy tree in such a way that the two joins can be completed approximately the same time. (Note that this is in contrast to [381] which is synchronized at each level of the join tree.) The above step for partitioning the processors for the root is then applied to all internal vertices in the tree in a top down manner until each internal (join) vertex is allocated a number of processors. Note that when the number of processors passed to an internal vertex in a lower level of the tree is too few to be further partitioned for efficient execution of joins, sequential execution for the joins in its child vertices is employed for better performance. There are many different bushy execution trees for a query, and the concept of synchronous execution time can in fact be applied to the bushy trees obtained by different heuristics. In [107], the numbers of processors allocated under SE, TE and ST_{SE} are compared under the same query and relation specifications. Figure 12.6 shows an example allocation considered in [107]. The bushy tree and processor allocation using ST_{SE} that applies synchronous execution time to SE is shown in Figure 12.6c. In spite of the fact that the bushy tree in Figure 12.6c is the same as that in Figure 12.6a, the resulting execution

times differ due to the variation in processor allocation. It can be seen that under ST_{SE}, processors are allocated to the execution of each join in such a way that two joins generating the two operands for a later join can be completed approximately the same time, thus alleviating execution dependency. Moreover, since the processors allocated to a vertex in a bushy tree are partitioned for the allocation to its child vertices, system fragmentation is eased. Among all the schemes evaluated in [107], the approach by ST_{SE}, which (1) first applies the join sequence heuristic to build a bushy tree to minimize the total amount of work required as if under a single processor system, and then, (2) in light of the concept of synchronous execution time, allocates processors to the internal vertices of the bushy tree in a top down manner, is shown to be the best solution to minimize the query execution time. A variation of ST_{SE} based on dynamic programming is considered in [628].

In [628], several iterative processor allocation schemes are considered for a given bushy tree. Basically each join operation is first given an allocation corresponding to the best efficiency point. After the query tree is generated, some criterion is used to identify the join operation which is most likely to be the bottleneck stage and to allocate that stage with more processors. Then the whole process repeats. Different criteria, dealing with the operation with the longest processing time and the one with the largest number of concurrent idle processors during its life time, are considered. This iterative approach is based on the algorithms studied in [589] for extending the scheduling algorithms for non-malleable tasks to those for malleable ones. Also, given the processor allocation, various scheduling heuristics with precedence constraints are explored in [628] to follow the precedence relations imposed by the bushy tree.

As an extension to [255], where only optimizing the parallelization of sequential plans was addressed, an algorithm dealing with processor scheduling for a bushy tree was proposed in [254]. The inter-operator parallelism is achieved by properly selecting the IO-bound and CPU-bound task mix to be executed concurrently and making processor allocation to reach the IO-CPU balance points. The algorithm matches up IO-bound and CPU-bound tasks with an appropriate degree of intra-operator parallelism so that both processors and disks can operate as close to their full utilization as possible, thereby minimizing the elapsed time. In order to ensure an efficient solution, [254] only explored schemes that execute at most two tasks at a time. The question of how to reach the IO-CPU balance point among more than two tasks appears to be a challenging problem.

12.6.2 Join Methods with Pipelining

A pipeline of hash joins is composed of several stages, each of which is associated with one join operation that can be executed, in parallel, by several processors. For illustrative purposes, the execution of a pipeline segment is shown in Figure 12.7. Recall that the hash join execution consists of two phases: the table-building phase and the tuple-probing phase. In the pipeline execution of a hash join, it is generally assumed that these two phases are disjoint across stages in the sense that all processing nodes will need to finish the table-building phase before the tuple-probing phase begins.

It is known that pipelining has the following two advantages. First, the IO cost is significantly reduced since the intermediate relations between stages in a segment need not be written back to disks, or even exist as whole tables in the memory. Second, the first tuples of the resulting relation of a pipeline segment can be produced earlier, not only reducing the perceived response time by an end user, but also enabling an application program to start processing the result earlier. To further improve the

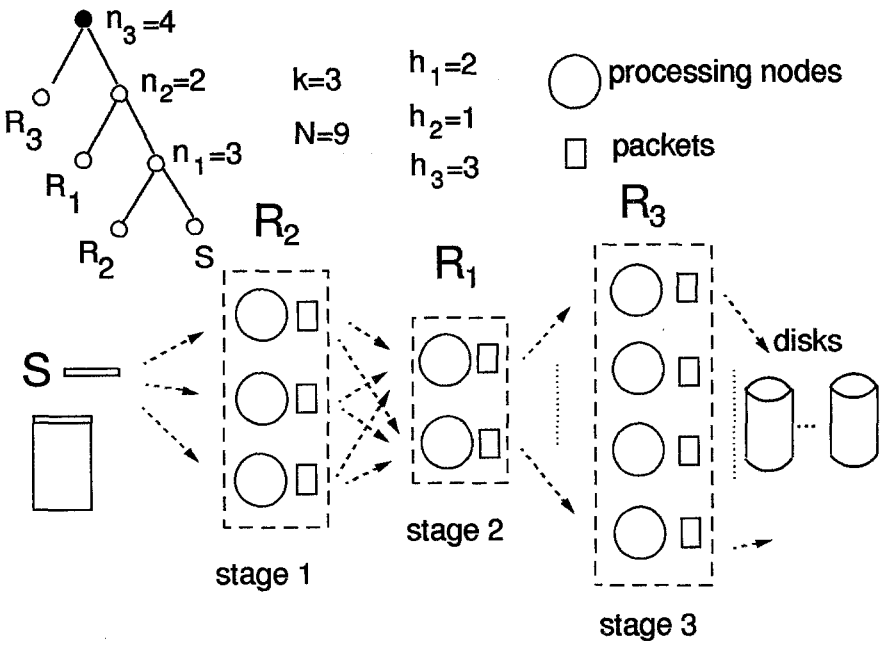


Figure 12.7: Execution of One Pipeline Segment

performance of pipelining, an algorithm using 2-way semijoins to pipeline multiple joins was proposed in [475]. In [614], an innovative approach based on using double hash tables was proposed to handle execution dependency. These methods are in fact applicable to both linear and bushy trees. (However, the focus of those works is not on join sequence selection or processor allocation.) In the following, we shall first discuss join sequence scheduling and then processor allocation for pipelined hash joins.

Join Sequence Scheduling

Right-deep tree

It can be seen that both right-deep and bushy trees allow the implementation of pipelining. Schneider and DeWitt were among the first to study the effect of pipelining for hash joins [497, 498]. Their focus is on the use of right-deep trees, due mainly to simplicity and the uncertainty of the improvement achievable by using bushy trees. Clearly, for a given query, the number of right-deep trees to be considered is significantly less than that of bushy trees, and simple heuristics can be applied with little overhead to generate a right-deep query plan. For example, a right-deep tree can be obtained by first constructing a left-deep tree by greedy methods and then taking a mirror image of the resulting left-deep tree [497]. However, right-deep trees suffer from the drawback of less flexibility in structure, which in turn implies a limitation on performance.

Since the amount of memory is usually not enough to accommodate hash tables of all inner relations, provisions such as static right-deep scheduling and dynamic bottom-up scheduling [498] are needed. The static right-deep scheduling decomposes the right-

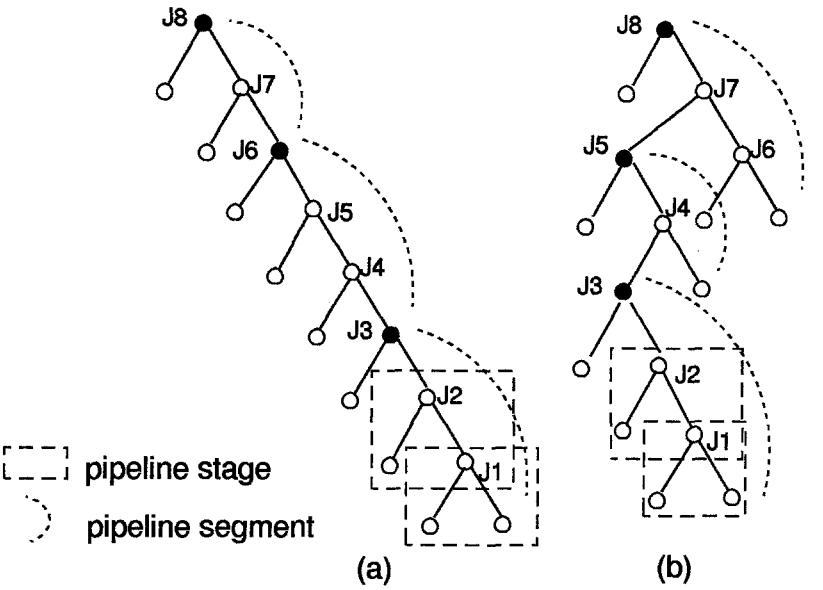


Figure 12.8: (a) a conventional right-deep tree, and (b) a segmented right-deep tree.

deep tree into segments off-line and loads hash tables of a segment into memory in parallel. The dynamic bottom-up scheduling, in contrast, loads one hash table into memory at a time and determines the break points of segments dynamically according to the memory constraint. In both scheduling methods, however, the execution of the whole query is implemented in one pipeline and is thus restricted to the structure of a right-deep tree. An example right-deep tree which is decomposed into three segments is shown in Figure 12.8a. Those joins whose resulting relations need to be written back to disks are marked black in Figure 12.8a.

Segmented Right-deep tree

The bushy tree, on the other hand, offers more flexibility in query plan generation at the cost of searching a larger design space. However, as far as the hash join is concerned, the scheduling for an execution plan of a bushy tree structure is much more complicated than that of a right-deep tree structure. In particular, it is very difficult to achieve the synchronization required for the execution of bushy trees in such a way that the effect of pipelining can be fully utilized. To generate effective query plans that fully exploit the advantage of pipelining while avoiding the above mentioned deficiencies of the bushy and right-deep trees, an approach based on *segmented right-deep trees* for the execution of pipelined hash joins was proposed in [105]. A segmented right-deep tree is a bushy tree which is composed of a set of right-deep subtrees. Formally, a segmented right-deep tree is defined recursively to be either (a) a right-deep tree, or (b) a right-deep tree with some of its leaf vertices replaced by segmented right-deep

trees. An example of a segmented right-deep tree of 3 pipeline segments can be found in Figure 12.8b. Some greedy schemes to deal with the relation selection for building a segmented right-deep tree were also proposed in [105]. As evaluated by the simulation in [105], the approach on segmented right-deep trees possesses more flexibility in query plan generation without incurring additional overhead on plan execution, and compares favorably with not only the right-deep trees generated by greedy methods but also the optimal right-deep tree that has the shortest execution time among all right-deep trees. This suggests that to efficiently execute pipelined hash joins in today's environment, one should exploit methods utilizing the bushy trees instead of improving the heuristics on generating right-deep trees.

Processor Allocation

While most prior work focused on query plan generation, there is relatively little literature regarding optimizing processor allocation among the pipeline stages. One approach to deal with processor allocation to explicitly avoid pipeline bottleneck while taking memory constraints into account is taken in [376]. As shown there, the characteristics of pipelined hash joins allow the processor allocation problem to be formulated as a two-phase mini-max optimization problem. Specifically, for a pipeline with k stages, the execution time of the pipeline, TS, can be expressed as

$$TS = \max_{0 \le i \le k-1} \frac{WB_i}{n_i} + \max_{0 \le i \le k-1} \frac{WP_i}{n_i}, \tag{12.1}$$

where WB_i and WP_i are, respectively, the workloads for the table-building and tuple-probing phases in stage i. Note that the processing time of each phase is determined by the maximal execution time among all stages, and that the same allocation of processors to a stage is retained across the table-building and tuple-probing phases. The execution time of a pipeline is thus the sum of two correlated maxima. Consequently, the processor allocation problem for pipelined hash joins can be stated as follows: Given WB_i and WP_i, $0 \le i \le k-1$, determine the processor allocation $\langle n_i \rangle = (n_0, n_1, \ldots, n_{k-1})$ which minimizes TS in Eq.(12.1), where n_i is the number of processors allocated to stage i.

For example, consider the workloads shown in Table 12.2 for a pipeline of five stages. First, it is observed that the workloads of stage 2 are less than those of stage 3, suggesting that stage 3 should be assigned more processors than stage 2. However, stage 3 has a heavier load in the table-building phase than stage 4, while the latter has a heavier load in the tuple-probing phase. In such a configuration, there is no obvious way to allocate processors to minimize the pipeline execution time specified in Eq.(12.1). For illustrative purposes, suppose the total number of processors available to execute the pipeline in Table 12.2 is 20. It can be seen that the allocation $\langle n_i \rangle$= (4, 4, 4, 4, 4) leads to $\max_{\forall i} \frac{WB_i}{n_i}$= 1.5, $\max_{\forall i} \frac{WP_i}{n_i}$= 1.2, and $TS = 2.7$, whereas the one $\langle n_i \rangle$= (6, 3, 3, 6, 2), which is based on the workloads of the table-building phase, leads to $\max_{\forall i} \frac{WB_i}{n_i}$= 1.0, $\max_{\forall i} \frac{WP_i}{n_i}$= 2.5, and $TS = 3.5$. Clearly, to develop an optimal processor allocation to minimize TS in Eq.(12.1) is in general a very difficult and important problem. Since the table-building and tuple-probing phases are executed one after the other, we minimize the sum of two correlated maxima in Eq.(12.1). In view of this, the optimal processor allocation problem in Eq.(12.1) is termed the two-phase mini-max optimization problem.

To develop the optimal processor allocation scheme for the two-phase mini-max optimization, the following three constraints were considered in [376]: (1) the total

stage	0	1	2	3	4
WB_i	6	3	3	6	2
WP_i	4	4	2	3	5

Table 12.2: The Workloads in Two Phases of Each Stage

number of processors available for allocation is fixed, (2) a lower bound is imposed on the number of processors required for each stage to meet the corresponding memory requirement, and (3) processors are available only in discrete units. By incrementally adding constraints to the optimization problem, solution schemes were derived. The effectiveness of the problem formulation and solution procedure was verified through a detailed simulation of pipelined hash joins. Simulation results show that these allocation schemes lead to much shorter query execution times than conventional approaches.

12.7 Scheduling Multiple Complex Queries

So far our study of query parallelism has focused on optimizing the execution of a single query. While it is important to understand how to schedule a single query across several processors, one should keep in mind that queries are not always executed in isolation. Thus, in order to fully exploit the available parallelism from a system, one must consider the interactions among different queries submitted to the same machine. This section discusses some of the issues associated with optimizing the combined execution of multiple queries.

In order to further motivate our discussion, consider the execution of two different queries, Q_1 and Q_2, submitted to the same parallel system consisting of $P = 6$ processors. Many of the systems that have been designed to date assume either that a fixed number of processors are available to a given query or that the query is subdivided into a fixed number of subtasks which are then distributed among the available processors. However, in practice the execution time of each of these queries can be optimized based on the number of available processors. As shown in Figure 12.9, the execution time of a particular query is actually a function of the number of processors allocated to that query. We say that such a query is *malleable* [588]: This means that as the number of processors allocated to a query increases, the execution time of the query decreases.

The problem occurs when one does not take advantage of the parallelizable nature of the queries. In particular, consider what happens if we statically assign m processors to any query in the system. Recall that the number of processors in the system is $P = 6$. If $m = 6$, then if both Q_1 and Q_2 in Figure 12.9 are submitted concurrently the total completion time of both queries will be 14. The optimal assignment in this case would have been to give Q_1 4 processors and Q_2 2 processors, resulting in a completion time of 10. Assume now, in an attempt to alleviate the problem described above, that we change m to 3. Then, if Q_1 is the only query submitted to the system the completion time of the query will be 10. The optimal assignment in this case would have been to give Q_1 6 processors, resulting in a completion time of 4.

In the event that a larger number of tasks are submitted concurrently to the system, the above problem would be exacerbated. In general, we wish to allocate enough processors to each query so as to minimize the completion time of that query. On the

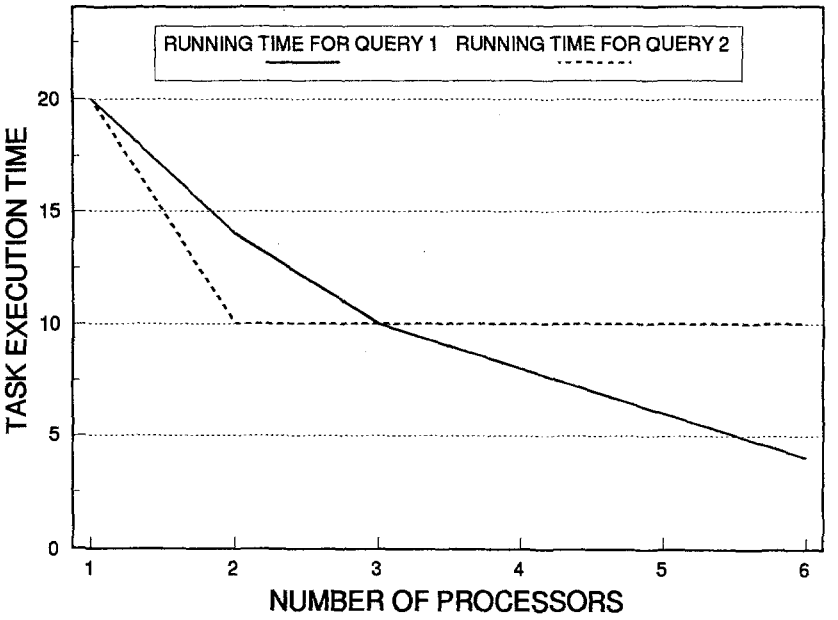


Figure 12.9: Query Execution Time as a Function of the Number of Processors

other hand, we do not want to allocate so many processors to any single query so as to negatively impact the overall efficiency of the system.

12.7.1 A Hierarchical Approach to Inter-Query Parallelism

Formally, the *malleable scheduling* problem can be defined as follows. Consider a multiprocessor system consisting of P processors, and a set of N tasks which we wish to schedule on this system. We assume that each task $j \in \{1, ..., N\}$ can be *allocated* an arbitrary number of processors $\beta_j \in \{1, ..., P\}$, and that its *task execution time* $t_j(\beta_j) > 0$ is a nonincreasing function of the number of allocated processors. All of the processors allocated to a task are required to execute that task in unison. That is, these β_j processors are all required to start task j at some *starting time*, say τ_j. They will then complete task j at some later *completion time* $\tau_j + t_j(\beta_j)$. The tasks are partially ordered by a *precedence* relation $\prec$. A *schedule* will consist, for each task $j \in \{1, ..., N\}$, of a processor allocation β_j, and a starting time τ_j. A schedule is required to be *legal* in the following two senses:

- For any time τ, the number of active processors does not exceed the total number of processors. In other words,

$$\sum_{\{j | \tau_j \leq \tau < \tau_j + t_j(\beta_j)\}} \beta_j \leq P.$$

- If one task j_1 has precedence over another task j_2, then the second task cannot begin until the first task completes. In other words, if $j_1 \prec j_2$, then $\tau_{j_1} +$

$t_{j_1}(\beta_{j_1}) \leq \tau_{j_2}$.

We seek an *optimal* schedule, one for which the overall *makespan* given by

$$\max_{1 \leq j \leq N} \{\tau_j + t_j(\beta_j)\}$$

is minimized. In other words, we wish to minimize the latest task completion time.[‖]

One reason this inter-query problem has not been extensively studied has to do with its inherent difficulty: As shown in [165], the malleable scheduling problem is NP-hard in the strong sense, even in the special case when there is no precedence. One is thus more or less forced to consider heuristic solutions. Typically, one would like to find algorithms having *provable* bounds on the suboptimality of a given schedule. That is, we would like to say that the makespan w, of the schedule given by our heuristic, is $w \leq k * w^*$ where k is some known constant and w^* is the makespan of the optimal schedule.

Later we will present algorithms having suboptimality bounds for the special case where there are no precedence constraints between the tasks. (In practice, these algorithms usually find a schedule having a makespan that is very close to that of the optimal schedule.) Unfortunately, there has been relatively little success with establishing such bounds for the general case of scheduling malleable tasks in the presence of precedence constraints. We know of only one result that addresses this issue [601]. This chapter deals only with the special case where the task execution time function is constrained to be inversely proportional to the number of processors allocated to the task up to some specified maximum degree of parallelism $0 < \delta_j \leq m$ for the task, and constant thereafter.[**] Formally,

$$t_j(\beta_j) = \begin{cases} \frac{t_j}{\beta_j} & \beta_j \leq \delta_j \\ \frac{t_j}{\delta_j} & \beta_j > \delta_j. \end{cases} \tag{12.2}$$

Here t_j denotes the task execution time on a single processor. The suboptimality bound given by the algorithms in both these papers is shown to be $(3 - \frac{2}{m})w^*$. Of course, in practice, speedup curves of operators in a query are unlikely to be completely linear and the bounds given by these papers will fail to hold.

Although there is a lack of heuristics with provable suboptimality bounds for precedence based scheduling, we have already seen that many heuristics for the scheduling of intra-query parallelism are in fact quite effective under varying system assumptions [107, 255, 334, 511, 569, 570]. One natural way, considered in [628], of extending any of these heuristics to deal with inter-query parallelism is to add one or more dummy nodes, representing operators with 0 execution time, so as to combine all the queries into one large query. This approach is shown in Figure 12.10. Unfortunately, as shown in [628], this approach does not seem to work very well. Relationships that hold among the operators in a single query do not always hold across different queries, and thus heuristics that are effective in optimizing the performance of a single query do not easily extend to several queries.

[‖]The condition requiring that the task times $t_j(\beta_j)$ be nonincreasing is not, in fact, restrictive. One can always set $t'_j(\beta_j) = \min_{1 \leq \beta \leq \beta_j}\{t_j(\beta)\}$. The task times $t'_j(\beta_j)$ *are* nonincreasing, and any extra, useless processors in the optimal solution can be left idle.

[**]In the absence of a specified maximum degree of parallelism for each task the problem with linear speedups becomes trivial. One merely needs to allocate m processors to each task and then schedule them in a batch fashion.

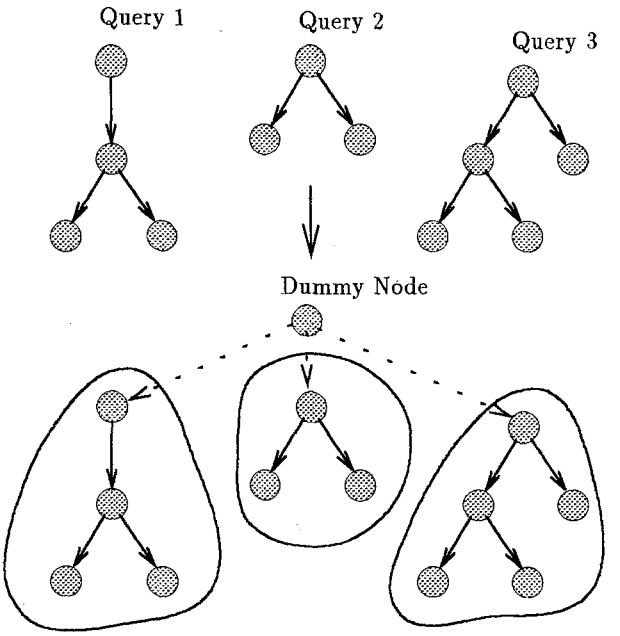


Figure 12.10: Combining Multiple Queries into One Query

What [628] did show, however, was that a *hierarchical* approach to scheduling the queries was effective. The hierarchical approach proceeds in two phases: In phase 1 one of the single query optimization techniques is applied to each of the individual queries to be executed, generating *composite* task execution time functions for the queries themselves. Thus, one generates a query time execution curve based on the different potential number of processors allocated to the query. In phase 2 a different scheduling algorithm is applied to the composite query tasks, which have no precedence. In the remainder of this chapter we will refer to the problem of malleable task scheduling when there are no precedence constraints as problem **MS**. This approach is outlined in Figure 12.11. Heuristics for handling the problem **MS** is the topic of the rest of this section.

12.7.2 Scheduling Independent Tasks

Krishnamurti and Ma [336] posed the problem **MS** and solve what amounts to the special case of packing parallelizable tasks onto a single shelf. (This is, of course, feasible only when the number of tasks does not exceed the number of processors.) The solution proposed turns out to be optimal over all solutions requiring that the all the tasks begin at time 0. Worst-case bounds on the makespan (with respect to the makespan of the optimal schedule) are given under the condition that the *work* performed by task T_j, given by $\beta_j t_j(\beta_j)$, is a nondecreasing as a function of β_j. Limiting the tasks to a single shelf, however, can be shown to yield arbitrarily bad schedules when the number of tasks approaches the number of processors. Furthermore, the algorithm does not address the question of how to handle the case where the number

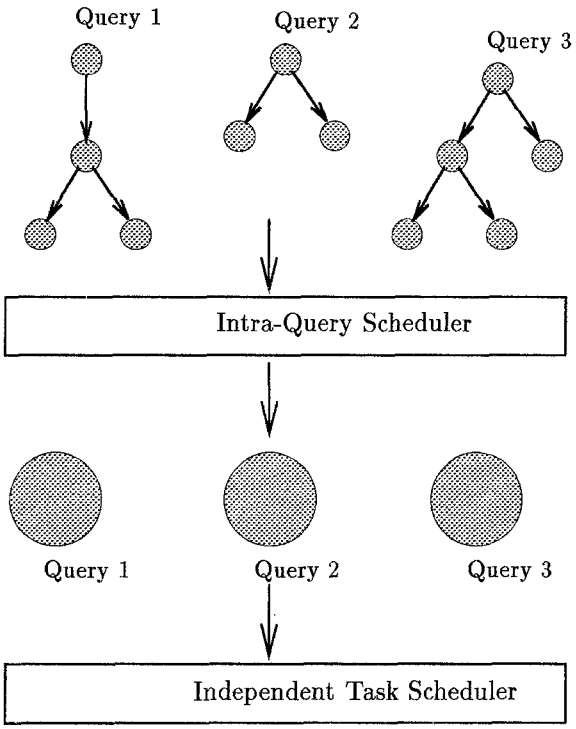


Figure 12.11: A Hierarchical Approach to Inter-Query Scheduling

of tasks is greater than the number of processors.

Turek, Wolf, Pattipati and Yu [588] relaxed the constraint that the parallelizable tasks all start at the same time as well as the condition that the work performed is a nondecreasing function, showing how to get an optimal K-shelf solution for any fixed number of shelves K. By varying K over all values from 1 to n one can show that the makespan of the schedule created using the algorithm in [588] will be no worse than 2.7 times the makespan of the optimal schedule. Unfortunately, the algorithm is exponential in K and, therefore, is not necessarily practical when large values of K need to be considered. (On the other hand, the algorithm *is* polynomial for each specific value of K, and an effective bounding argument given in [588] limits the number of values of K that need to be considered for many examples.)

Belkhale and Banerjee [41] gave an approximate algorithm with polynomial running time for the problem **MS** (again) under the additional constraint that the work done by the tasks is nondecreasing in the number of processors. Under the stated condition, the suboptimality bounds on the makespan of the schedule produced by their algorithm will be no worse than 2. Turek, Wolf, and Yu [589] presented a technique does not constrain the execution time of the tasks in any way and in addition allows the additional constraint that cooperating processors may need to be physically adjacent. The main contribution of their paper was a family of approximate solutions to the problem **MS**, using an extension of a technique frequently used to solve resource allocation problems [270]. In particular, they give an algorithm that selects candidate numbers of processors to be allocated to each of the parallelizable tasks, so that one can then use as a subroutine any algorithm A that "solves" the easier multiprocessor scheduling problem in which the number of processors allocated to a task is known a priori. (This problem has been extensively studied in the literature. See [27, 68, 127].) The algorithm has the property that for a large class of known algorithms their extension will guarantee the same worst case suboptimality bounds as A while increasing the running time of A by at most an additive factor of $O(nm)$. Applying their technique yields:

- Assuming that processor addresses assigned to a task need not be contiguous one can use the algorithm specified in [211] to get an algorithm that generates a schedule with makespan $w \leq 2w^{\star}$, where $w^{\star}$ is the makespan of the optimal schedule.

- Assuming that processor addresses assigned to a task need to be contiguous one can use the algorithm specified in [526] to get an algorithm that generates a schedule with makespan $w \leq 2.5w^{\star}$.

While our discussion has focused primarily on the execution of a set of known queries, we should not forget that the problem of scheduling the online arrival of tasks is an important component of any scheduling algorithm. One approach to this problem has been proposed by [520]. It consists effectively of scheduling all the tasks that have been submitted to the system at time τ. Any new tasks that arrive will have to wait until the time when the last task scheduled at time τ has started execution. This approach has the desirable qualities that (1) no task gets starved, and (2) it can be proven to affect the suboptimality bounds of batch scheduling algorithms by at most a factor of 2.

12.8 Summary

In this chapter, we have examined the various issues and techniques encountered in parallel query processing. The methods used to exploit query parallelism are divided into three categories: namely *intra-operator, inter-operator,* and *inter-query parallelism.* We have concentrated on join operations because they are the most expensive operations to execute, especially with the increases in database size and query complexity. In intra-operator parallelism, the major issue is task creation, where the objective is to split a operation into tasks so that the load can be spread evenly across a given number of processors. In the presence of data skew in the join attribute, naive range splitting or hashing does not suffice to balance the load. Inter-operator parallelism can be achieved either through parallel execution of independent operations or through pipelining. In either case, the major issues are the join sequence selection and processor allocation for each operation. Join sequence selection determines the precedence relations among the operations. For inter-query parallelism, the issue again is processor allocation, but now among the multiple queries. We explored query parallelism based on a hierarchical approach under a unified framework, so that potential integration of the techniques used to address each type of parallelism could be illustrated.

Chapter 13

Towards Flexible Distributed Information Retrieval

David W. Flater*, Yelena Yesha*†

13.1 Introduction

Information Retrieval is the process of locating the data which are the best answer to a user's query. In the case of text retrieval, it is "leading the user to those documents that will best enable him/her to satisfy his/her need for information [42, 471]." Information Retrieval is part of every database, every catalog, and every file system. It is such an integral part of all of these things that it is seldom thought of as an independent process.

This could soon change, however. The sharp increase in the amount of information available to the average user in recent years has drawn attention to the shortcomings of commonly used information retrieval systems. For years, very restrictive and inferior methods have been used to retrieve information. These methods often force the user to know the exact name or other identifier of the data to be retrieved. Sometimes one is lucky enough to be able to use regular expressions or keywords to locate data which cannot immediately be identified. Nevertheless, locating the desired information is often a frustrating, time-consuming, and even futile task.

Fortunately, information retrieval techniques have been under investigation for years, and a vast improvement could be realized in current information systems merely by implementing known algorithms. Most established techniques are designed for handling textual data, which is in fact where the greatest amount of difficulty is being experienced, but encouraging results for non-textual data are already being reported. In theory, the task of easing the general burden of information retrieval on the user has been well undertaken. In practice, the systems which are widely used every day are still relatively primitive.

If all information retrieval systems were upgraded overnight to use the best techniques available, finding information would be easier – but it would still not be trivial. There is another layer to the problem, which could humorously be called the *network* layer, but whose accepted name is the Resource Discovery Problem[503]. Proliferation of information is not the only difficulty. There is also a proliferation of information

*Computer Science Department, University of Maryland Baltimore County, Baltimore, MD 21228.

†Supported in part by the Information Systems Engineering Division of the National Institute of Standards and Technology, Gaithersburg, MD 20899.

bases. In an information retrieval system, one may start a serial search through the entire information base for a particular piece of information. One does not, however, have the option of starting a serial search over the entire Internet for a particular kind of database. In fact, even if one could connect to every site on the network, one would probably only be able to figure out how to access the available databases in the case of well-known archives and public catalogs.

At first glance, it would appear that the problem of finding the information base which would contain the desired information is just another form of information retrieval. One could establish a directory of information bases and use information retrieval techniques to find the correct information base. Later in this chapter, the reasons why this approach is inadequate will be presented.

This chapter is organized as follows. Section 13.2 provides a classification of information retrieval techniques. Section 13.3 describes the construction and use of thesauri and dictionaries for information retrieval. Section 13.4 discusses existing IR techniques and the use of these techniques to achieve fuzzy retrieval. Section 13.5 investigates the difficulties of extending IR into the distributed domain, and Section 13.6 describes an information system architecture which handles some of those problems.

13.2 Information Retrieval Techniques

IR techniques can be divided into three classes: tag-based retrieval, partial content-based retrieval, and full content-based retrieval. In tag-based retrieval, indices are painstakingly built by human beings, and the system uses more or less clever algorithms to search the index for relevant entries. (The term "index" is used loosely here to indicate any form of data keying, whether by means of inversion or signatures.) In partial content-based retrieval, indices are built automatically based on the content of data objects. In full content-based retrieval, queries are executed directly upon the data objects, and no index is required.

13.2.1 Tag-Based Retrieval

One of the simplest forms of IR is tag-based retrieval. When data objects are added to the information base, a human being has to provide meta-information, such as a subject classification or a set of keywords, for each data object. The information system then uses this meta-information as an index when it is called upon to retrieve data. The data objects themselves are never inspected by the information system; they are merely reproduced verbatim when their meta-information satisfies a user's query.

This form of information retrieval is used today in many applications. In cases where qualified personnel are available to perform the indexing for an information base of limited scope, such as a library of legal documents, these systems can provide extremely efficient information retrieval. However, consider the case of a public library where information of all kinds needs to be indexed. The library workers have the task of maintaining an internally consistent general classification system in the face of a constant influx of new documents. As a result, they will probably not distinguish between, say, distributed databases and distributed information systems, causing a degradation of retrieval precision. The library would require a staff of experts from every field to properly maintain their subject hierarchy.

13.2.2 Partial Content-Based Retrieval

Consider how much better the situation would be if the process of indexing an information base were automated. In place of the staff of experts needed to maintain a generalized subject hierarchy for a public library would be a library of software. Each program in the library would be maintained with assistance from field experts to perform accurate classification within one particular field of interest. In the best case one would also have a master program which could automatically determine the field of interest a document belongs to and pass it on to the "expert" in that field. The same software could then be distributed to all libraries, and the library workers would have a much simpler job.

It is on exactly this kind of technology that most current research into Information Retrieval focuses. Unfortunately, while there has been some success in the localized indexing problem for limited domains[163, 461], no one has yet succeeded in integrating many "experts" with a main program capable of solving the higher-level classification problem.

13.2.3 Full Content-Based Retrieval

An index must contain all the information which is required by the retrieval algorithm, including keyword frequencies or whatever else the algorithm might use. There is thus a conflict between the desire for thorough information retrieval and the desire for tight, efficient access structures. The more one restricts oneself to a fixed set of attributes, the more one is able to use optimized methods such as signature files[629]. The less brute-force searching is done or the less overhead is accepted, the less effective is the information retrieval.

The limits of information retrieval are attained by full content-based methods. These methods use no access structures whatsoever. The entirety of every data object in the information base is subject to scrutiny. No information whatsoever is omitted from the search. The benefits in terms of the system's ability to handle queries on obscure topics are phenomenal – but so are the costs. To perform such a thorough search on a large library in a reasonable amount of time would likely be beyond the ability of even the most powerful parallel processing engines. However, it is possible that these techniques could be used on small databases of limited scope, provided that there is still a global classifier capable of determining the appropriate domain for a data object or query.

13.3 Thesauri and Dictionaries

One of the most common problems with text retrieval systems is that the user must often express the same query in many different ways until he/she stumbles onto some keywords that are in the index. The information systems are not sophisticated enough to map an arbitrary natural language term onto the set of terms in their indices which have approximately the same semantics, even if the natural language term is an exact synonym of an index term. Since users do not know the set of index terms, they are forced to use trial and error to find keywords that trigger the retrieval of relevant documents. The usefulness of such a system will vary drastically from one user to the next, depending on how close that user's vocabulary is to the index vocabulary. To solve this problem, work is being done to find ways to build thesauri and dictionaries which will enable efficient retrieval of text from natural language queries.

13.3.1 Thesauri

A thesaurus is a mapping from terms to terms which are semantically close. It may be viewed as a graph where each node contains a term and has links to all of its synonyms. A text retrieval system which is faced with a query term which does not appear in its index should be able to use a thesaurus to try alternate terms in the order of decreasing semantic similarity. If a user specifies a synonym of an index term, the system should be able to determine the index term. On the other hand, if the information base has been indexed inconsistently, a thesaurus should help alleviate the problem by making some terms nearly equivalent.

The popular information retrieval technique of indexing by word stems or trigrams[398] can be thought of as a heuristic for detecting synonyms in English text. The thesaurus which it implicitly generates is neither complete nor accurate. A complete and accurate thesaurus would be sufficient to maximize the efficiency of simple keyword-based retrieval systems. Where more complex retrieval methods are used, a dictionary may be needed to provide semantic information to the retrieval or query-parsing algorithms.

The degree of closeness for each mapping from a term to a synonym must somehow be expressed by a thesaurus. To avoid the scenario where every possible combination of terms is assigned a similarity weight, thesauri should be built with the idea in mind that one may navigate more than one link away from the term one is examining if none of its immediate synonyms is satisfactory. To find the synonyms of a word in order of decreasing similarity, a breadth-first search of the graph is performed. If weights are assigned to the links, a more expensive best-first search must be used to insure perfect sorting, but this may be more trouble than it is worth.

Thesauri can be extended to convey more semantic information on how terms are related to each other[444]. For example, there can be a distinguished, directional link type which indicates that one term is a generalization of another. This link type allows an information system to generalize the terms of an overly specific query until some documents can be retrieved. However, it may be undecidable whether or not overspecification was responsible for a null response.

Early attempts at building thesauri automatically using purely statistical methods met with limited success. However, a recent attempt to improve the quality of automatic term association by using linguistic knowledge was successful[478]. Linguistic analysis could be used to improve the signal-to-noise ratio of statistical methods by transforming those sentence structures which cause unrelated terms to co-occur. While reliable parsing of English grammar is exceedingly difficult, a surprisingly high success rate can be achieved using simple methods if one only wishes to recognize a limited subset of the possible sentence structures[251]. It should therefore be possible to build sufficiently accurate thesauri using a combination of statistical methods and limited parsing. Another novel approach to automatic thesaurus construction uses the complete link clustering algorithm to cluster the document collection prior to the construction of the thesaurus[133]. This two-stage statistical approach, which has a strong theoretical basis, has also yielded promising results.

The uncertain accuracy of the resulting thesauri is not the only disadvantage of automatic thesaurus construction. While several different kinds of relations between terms can be distinguished by automatic methods[478], it is usually the case that statistical methods point out the existence of relationships without classifying them[444].

This places a restriction on the kinds of retrieval algorithms that can be used with automatically generated thesauri. However, for the retrieval methods which are most commonly used, automatic thesaurus generation provides a way to rapidly create many domain-specific thesauri until more accurate thesauri become available.

If one continues to build more and more semantic information into a thesaurus, at some point it ceases to be a thesaurus and becomes a dictionary. The essence of the distinction is that the possibilities for defining relationships between terms are exhausted and information related to the terms themselves, i.e. "definitions," begin to appear.

13.3.2 Dictionaries

When one crosses the line from text processing into text understanding, one also needs to trade in one's thesaurus for a dictionary. Any term-related information which would be useful during the retrieval process can be kept in a dictionary. This includes internal semantic representations that can be retrieved from the dictionary and used as building blocks for larger expressions.

As of now, dictionaries are primarily being used in expert systems and knowledge bases. However, for years it has been asserted that large amounts of "world knowledge" are needed for effective natural language understanding. Dictionaries might therefore be needed when thesauri become inadequate to support natural language information retrieval.

13.4 Fuzzy Retrieval

Thesauri are one of the tools that are being used to enable fuzzy retrieval, the probabilistic retrieval of data objects based on an estimate of their relevance to a query. Fuzzy retrieval is made necessary by the inherent limitations of non-fuzzy methods such as Boolean querying[482, 73]. Non-fuzzy methods are inflexible and require the user to have prior knowledge of the indexing scheme of the information base if efficient retrieval is to occur.

Most fuzzy retrieval is built upon the notion of a relevance or similarity function. Given a query which is not satisfied by any object in the information base, the similarity function provides an estimate of how similar each object is to the hypothetical object described by the query. The program can then select a small number of objects which are considered to be most similar and list them for the user, along with their similarity (or relevance) estimates.

Consider for example the case of retrieving abstracts with keyword queries. A direct keyword system returns only those abstracts in which every keyword appears at least once. In effect, a Boolean value is assigned (zero or one) to each term to indicate whether or not it appears in the abstract, and then the values are multiplied to determine the relevance of the abstract to the query. This system can be made nominally fuzzy by accumulating, rather than multiplying, the term values, so that the most relevant abstract is the one containing the largest number of query terms. While this nominally fuzzy approach is currently used in many on-line library catalogs, much better techniques are available.

Given a thesaurus, one can immediately improve on the above similarity function by assigning a relevance value which is less than one but greater than zero for any term with a synonym in the document, but which does not appear itself. The value

assigned will depend on the thesaural similarity of the term and its synonym. The relevance function will then be fuzzy along the semantic axis as well as the syntactic one, with the result that retrieval will be much more consistent across different query vocabularies.

Even this similarity function is crude compared to many which appear in the literature. For example, it makes no attempt to weight query terms based on how often they appear in a given document, or how often they appear in the document collection as a whole. The best index terms are those which do not appear in so many documents as to be useless for retrieval, yet are not so infrequent that many relevant documents do not contain them. The tf-idf (term frequency – inverse document frequency) term weighting scheme is one way of biasing the search towards those terms which are most useful for retrieval[133, 482, 481]. Furthermore, accumulation is only one possible way of combining term coefficients into a document coefficient. It is common practice to arrange term coefficients for queries and documents into vectors (the "vector space model"[482]) and use vector operations such as cosine to determine their similarity[133].

Similarity functions do not exist only for text retrieval. A great deal of work is currently being done to allow information retrieval using images. One recent work determines the similarity of images which are comprised of iconic objects[96]. Specialized techniques have also been developed for retrieving documented software[385]. However, it is the case that text retrieval has received vastly more attention than other kinds of retrieval, and until sufficient work has been done in this area, textual descriptions must be used to index non-textual data.

One class of fuzzy retrieval algorithms which does not depend exclusively on a similarity function (but which nevertheless may use one) is based on the inference net model of information retrieval[42, 590]. Information retrieval systems which use these algorithms can look like a combination of hypertext and expert systems. They use Bayesian logic to reason about the properties of data objects and then decide which of them are most likely to satisfy the query. A Bayesian inference network[42, 450] is built for the object collection and for the query. The two are then connected so that the probability that each object satisfies the query can be computed. When feedback from the user causes changes in the subnet representing the user's query, the probabilities are recomputed and a new set of pertinent objects is returned. Instead of simply accumulating values, the inference net model deals in detail with the case where the probability that a given object is relevant to a query depends on multiple factors. However, the benefits of such a thorough analysis of relevance may not outweigh the additional complexity for many information retrieval applications.

13.4.1 Partial Content-Based Methods

The most direct way to adapt existing information retrieval systems to be partially content-based is to automate the process of indexing the data. Using this approach, the same old fuzzy retrieval mechanisms can be used to search the same kind of index. The only difference is that nobody had to go through the effort to build the index manually.

Automatic indexing is most frequently done using methods such as tf-idf to decide which of the terms in a document or document collection would make the best index terms. An index can then be generated which contains only those terms, along with their frequencies in each document. Additional lexical association metrics are the signal-to-noise (concentration), the variance (concentration/diffusion), and the discrim-

ination values of terms[261]. The discrimination value of a term reflects the amount of difference it makes to the document similarity function if that term is removed from consideration.

If an information retrieval system uses summaries or abstracts to index data, all is not lost. Some researchers hope to find ways of piecing together entirely nonanaphoric sentences to form a reasonably informative "abstract" of a document using sophisticated linguistic analysis. An easier approach can be used if something is known about the structure of the documents, as is the case with an information system of limited scope. For example, if all the documents describe analytical experiments, one can search for phrases such as "we conclude" or "we have shown" which often co-occur with important bits of summarized information. However, the problem in general is insidiously difficult, and simple-minded heuristics are unlikely to provide a solution[443]. One approach to automatic keyword indexing[163] employs techniques which have been used for automatic abstracting, but suggests that the extracted phrases simply be used for keyword searches and not regarded as full summaries.

Automatic indexing of images[461] is also an immense challenge for researchers, but it is receiving more attention and is likely to reach maturity before automatic abstracting. The cited work points out that "image retrieval by content cannot be attempted in general," but explains that, given a sufficient body of domain-specific patterns and rules for image analysis, one can identify the different objects in an image and build a representation of the image in terms of its objects, their number, their compositions, and their positions. The images themselves can then be indexed using these representations, and retrieval can be achieved by specifying the qualities to look for in the representations.

Lastly, some similar techniques are being used in the realm of hypertext. Attention is now being given to the use of text processing techniques to automatically build semantic network representations of input texts to facilitate hypertext browsing[394]. It is more difficult to automatically build a semantic network than to automatically index a data set. For indexing, all the program has to do is extract the most significant terms from a data object, or, depending on what sort of indexing is being used, determine which of a pre-selected list of index terms are descriptive of the datum. To build a semantic net, the program also has to add links connecting the important components of the datum to related concepts. The possible uses of semantic networks to enhance information retrieval capabilities are worthy of investigation.

13.5 Distributed Approaches to Info Retrieval

There have as of yet been few attempts to expand information retrieval into the world of networked and distributed computing. This is puzzling, considering the vast amount of attention that has been given to distributed databases and client-server architectures in recent years. Somewhat more frequent are applications of parallel processing hardware to cut down on the amount of real time needed to perform searches[132]. While brute force can no doubt do wonders for the effectiveness of information retrieval algorithms, this is only really helpful when the information base is homogeneous. The larger problem of networked information retrieval remains.

The most notable incursion of information retrieval into the networked computing domain is WAIS[305]. WAIS (Wide-Area Information Server) is technically a client-server system. However, it is the first attempt to connect autonomous information bases over a wide-area network which has achieved a significant following.

server system. However, it is the first attempt to connect autonomous information bases over a wide-area network which has achieved a significant following.

When performing a search in WAIS, a user must select one or more servers from a list of servers which is provided. The user then enters a keyword query which is executed on the selected servers. A ranked list of matching data is shown when the search completes. The user may then select individual data to retrieve. WAIS has the ability to handle non-textual data, but this capacity is currently being used to a very limited extent. For example, some images can only be retrieved by providing the name of the file in which the image is stored as the search criteria[225].

Other distributed approaches to information retrieval[241] preserve the single homogeneous information base flavor of existing information retrieval methods. This is in conflict with the rising trend of heterogeneous computing which is evident in the database community.

13.5.1 Current Research Issues

The first problem which must be solved to acquire information using FTP is finding the address of an archive site which has the right type of data. Similarly, the first problem which presents itself to a new user of WAIS is that he/she must scroll through a long list of servers to locate a server which contains the desired kind of information. These are just two different manifestations of the Resource Discovery Problem[503], the fundamental problem which must be solved to truly achieve a global integration of information sources.

The Resource Discovery Problem

The problem of locating information on a large network is known as the Resource Discovery Problem. Before a user can interact with a database or archive to retrieve needed information, the user must find out *where* such a database or archive exists.

One approach to solving this problem is to accumulate information to help users find what they need. For this purpose, there currently are a number of on and off-line Internet directory services[149] which let skilled users perform keyword searches to find out which Internet sites have the information they want. Unfortunately, the off-line directories cannot remain up-to-date, and the on-line services have limited effectiveness since they can only handle so much information. They cannot keep track of enough attributes of the data being archived to support general resource discovery. In many cases, one needs the name of the file which contains the desired information to find out the name of the site which owns it. There are also a variety of browsing and information-gathering tools[505, 467] which try to help the inexperienced user, but a browser also can only go so far to make a fragmented and heterogeneous set of resources appear to be unified.

The University of Colorado's Networked Resource Discovery Project has concentrated mainly on developing better directory services for the Internet; however, an early technical report describes their attempt to design a networked resource discovery system based on probabilistic multicasts and resource brokers[504]. More recently, Matsushita Information Technology Laboratory has begun working on an information system as part of the Gold project[35]. Its main improvement over [504] is that the functionality of the resource directories has been specified in greater detail. The brokers, which are the maintainers of the resource directories, now use information retrieval techniques to classify queries and data in order to determine the location of

the database capable of answering a query. The system discussed in Section 13.6 is unique in its brokerless approach to query routing and resource discovery and in its use of cooperative caching.

Large Information Networks

The proliferation of databases in government and industry has focused attention on the problem of integrating these databases. It is no longer practical for data to be disseminated by manually exporting it from one database and importing it into another; neither is it practical to call someone at a remote site each time a piece of information is needed. Since experience has shown that a large, tightly-knit system is too inert and unmaintainable to be a permanent solution[610], work is currently being done to find ways of solving the integration problem by building a system on top of the many databases which are already established. The terms heterogeneous database, federated database, and mediated database all describe this kind of system. Different authors will assign different connotations to these terms, and precise definitions are difficult to achieve. As a group, however, they describe a particular class of systems whose goal is to unify existing databases.

The techniques which are being suggested to allow integration of heterogeneous databases may help to build an integrated information system. The notion of mediators, distinguished pieces of software whose job it is to allow a graceful interface between the outside world and an information base (or other mediators), can help solve the general heterogeneity problem in a modular fashion[610].

13.6 Architecture for an Integrated Information System

13.6.1 Preliminaries

The authors have been working on an information system architecture which will eliminate the need for directories of archives, servers, and databases and provide a truly general-purpose information retrieval service[191, 193, 194, 195, 192]. They assume that individual sites will host information bases of limited scope – the sorts of archives that existing information retrieval techniques can handle or will be able to handle with sufficient refinement. They then build a distributed information system architecture on top of these information bases to route general queries to the sites which can answer them and to cache replicas of frequently-used data objects.

A data object is a document, an image, a piece of software, or any other self-contained unit which can serve as the answer to a query. For most types of data it will be possible to assign a name, a summary, a list of keywords, or some other textual identifier which can be used by term similarity algorithms to determine whether cached objects are appropriate answers to queries which are received. Particularly strange types of data might require a specialized descriptor which only certain "specialist" sites can interpret; the generic textual descriptor of such data would merely indicate that the data required special processing. For classes of data which cannot be assigned any kind of identifier, all queries will have to be forwarded to the sites which host those classes of data, and replicas will not be created.

The information system is constructed by forming a virtual network of sites. The links in the virtual network are given time-varying costs which reflect the observed delay

on the underlying wide-area network, or any other costs which must be minimized. The sites in the virtual network do not all have to own archives; many of them can simply cache replicas of data whose authoritative copies are elsewhere. Each network site will:

- Answer queries on data which are in its cache by sending a copy back towards the querying site;
- Forward queries on other data over the link which is believed to lead to the nearest copy of the desired data;
- Forward response messages along the least costly path to the querying site;
- Cache replicas of data contained in response messages;
- Replace older versions of data with newer versions when a response message is forwarded which contains a newer version of a datum in cache, or vice-versa;
- Process queries on data for which this site owns an archive (if applicable).

13.6.2 Query Routing

In order to know where to send a query, the information system must be able to determine the class of data which would satisfy a query. For example, a query asking for PC software should be sent to a site which archives PC software, and a query asking about the Civil War should be sent to a site hosting historical documents. Thus, it is expected that the queries received will contain sufficient information to allow this classification. Having to type "US history Civil War" instead of just "Civil War" is the price of general-purpose information retrieval. "Civil War" is only really sufficient if the information base is already restricted to historical documents. Of course, one can always type "encyclopedia Civil War" to find out that the Civil War falls under the category of US history. This last command would access a site which contains the "encyclopedic" class of data, i.e. superficial text on a wide variety of topics.

Queries are routed in a point-to-data fashion, rather than a point-to-point one. Instead of maintaining a routing table which tells which way to send a message bound for a particular site, a table is kept which is indexed by classes of data. Each row represents one class of data; each column represents one outbound link which could be used to forward the query. The entries themselves are values indicating the believed usefulness of each link for locating the class of data in question. When a response message passes through which contains a particular class of data, the row in the routing table for that class of data is usually updated to reflect the fact that a source of that data class can be reached in the direction from which the message arrived. In some cases, the row will be updated to point in the direction in which the message is forwarded with the belief that a cached replica of the data may be found in that direction.

Each time a row in the routing table is updated, the existing values are "aged" so that the newer information takes precedence. However, if a query which is forwarded comes back "empty handed," the older information will be used as the next most likely alternative is tried. The old information also comes in handy if a query arrives over the link which was thought to lead to the desired information. Sites avoid sending queries back to sites which have already seen them.

13.6.3 Cooperative Caching

Cooperative caching is distinguished from "selfish" caching by the fact that sites will cache data which do not directly benefit those sites. If one particular site is generating

a huge volume of queries, the surrounding sites through which that site's queries are forwarded will accumulate copies of the data which that site is using. Idle sites donate their cache space to further the common good.

It is not easy to arrange this kind of behavior in a fully distributed manner without resorting to expensive measures. However, it can be done inexpensively using a heuristic decision function that determines whether or not a site will cache a copy of a datum which it receives. By maintaining counters in response messages, the number of hops taken and the amount of link cost incurred by a response message between the last site which had a replica of the datum in the response and the current site can be determined. Similarly, it is possible to keep track of the amount of cache space owned by the intervening sites. Using this information along with the information available locally, i.e. the costs of the links attached to this site, the following decision function can be implemented:

$$\text{rnd} < \frac{\text{Hops so far}}{\text{Hops so far} + \text{tcf}} \times \frac{\text{Running link cost}}{\text{tcf} \sum \text{neighboring link costs}} \times \frac{\text{Running cache sum}}{\text{tcf}}$$

"Rnd" represents a random number in the interval [0,1). Nondeterminism is used to prevent anomalies such as global looping behavior from persisting if they ever arise. "Tcf" is the Turnover Control Factor for the current site. Each site maintains its own tcf in an attempt to maintain a constant level of cache turnover. Turnover can be estimated as an incrementally smoothed function of the difference between the current time and the hint-adjusted timestamps of the items being replaced in cache. If necessary, a different tcf can be used in each of the three factors in the decision function to provide more control.

The first factor, hops over hops plus tcf, discourages the creating of replicas when the last observed replica is very close. The second factor, running link cost over tcf times the sum of the neighboring link costs, encourages replicas to be created when the cost of reaching existing replicas becomes great. The neighboring link costs are used to get an idea of how large the costs incurred by the response message are relative to an average link cost for the local area. The last factor, running cache sum over tcf, keeps a site from being stingy with its cache space when it has plenty of it.

Cache replacement is done using a modified Least Recently Used scheme which allows cached replicas to have hint values[222] which are added to their last access times to bias the replacement. This feature is used to make it less likely that a replica of a datum which is expensive to requery will be replaced than a replica whose source is nearby.

If the decision function is not triggered by a response message, all is not lost. If the site happens to have enough idle cache space to create a replica without replacing any existing replicas, it can create a replica with a low hint value. That way, all available cache space can be utilized without jeopardizing replicas which are more important. Naturally, if the new replica is reused, its hint value is upgraded. Note that idle cache space results from fragmentation since data objects are cached as atomic units.

13.6.4 Simulation Results

The query routing and caching methods have been studied and developed with the aid of a simulation. The simulation has gone through several generations as the methods have been revised and assumptions have been generalized. Overall, the results indicate that an efficient distributed information system is feasible and could achieve a level of

```
Query path from 16 (3.091741):  16 25 24 1 6 18 2
Shortest from 2 to 16 (1.286241):  2 18 20 15 16
Actual from 2 to 16 (1.350579):  2 18 20 4 12 26 5 7 16
Time is now 615
```

Figure 13.1: Early Algorithm Development

performance which averages around 40% of that which could be achieved optimally, where the distribution of data replicas would be perfect and every query would be routed directly to the nearest available replica.

To give meaning to the results which follow, several terms must be defined. The term "link delay" will indicate the time consumed by transmitting a single message from one site to another over a link. The term "response delay" will indicate the total time consumed in the transmittal of a query and its response through any number of links. The "connectivity" of a sub-network, or cluster, is defined as the number of links between nodes in the cluster divided by the number of links required for the cluster to be fully connected.

First Generation

In the first generation, a cluster of identical sites in which all links had costs in the same order of magnitude was simulated. A simple replica placement strategy was used which depended only on the accumulated link cost since the last observed replica on the return path. All sites had the same size cache, and the set of primary copies did not grow once it was created. Data were classified statically based on the sites at which their primary copies lay. As was the case with all the simulations, the system was started with empty caches and empty routing tables, allowing them to self-stabilize over time. Network topologies were generated at random, testing everything from lines and trees to fully connected clusters. What happened was that the caching and routing heuristics interacted in such a way that a stable arrangement of data replicas was created, and correspondingly stable routing tables guaranteed that queries were routed to nearby replicas. As link cost were perturbed, the system made minor adaptations to continue using the minimal cost paths.

The main focus of the first generation was to establish that routing and caching heuristics could be found which would achieve the authors' goals. Most of the data collected in this early simulation study consisted of traces of the routes taken by messages and descriptions of the topologies through which the messages were being routed. A representative excerpt from such a trace is shown in Figure 13.1. The failure of the point-to-point router to use the shortest path (the path of least delay, to be precise) is a "feature" which was added later to generate more realistic results; an implemented router is only likely to approximate the shortest path since the actual delays will be constantly changing.

Second Generation

In the second generation the study began to concentrate on networks in which there were two classes of links, namely intercluster and intracluster links. A number of clusters connected by low-cost links were generated, then pairs of clusters were connected

by choosing random sites within each cluster to host the high-cost links. This topology has been investigated in detail firstly because it simulates the structure of real-world networks, and secondly because the first generation algorithms had the weakness of making too much use of expensive intercluster links. The second generation algorithms made exceptions to deal with radically expensive links, and the result was that sites within a cluster succeeded in cooperatively caching all the information which was needed rather than repeatedly fetching it from overseas, as it were.

The second generation simulation took the number of data (primary copies), the minimum datum size, the maximum datum size, the mean datum size, the number of sites, the number of clusters, and the connectivity as inputs. The set of authoritative primary copies would be generated randomly at the start of each run, but remain fixed thereafter. The network topology was created in a similar manner, but intracluster link delays were perturbed during the run to simulate changing network conditions. Later simulations would perturb all link delays.

As has been the case with all the simulations, the sizes of data were generated using a Poisson distribution. The actual average datum size is usually higher than the specified average due to the removal of data of size 0 from the distribution. Since the simulation does not try to estimate the overhead involved in caching data, each site could cache all the null data without incurring any cost; it is therefore pointless to include them in the simulation.

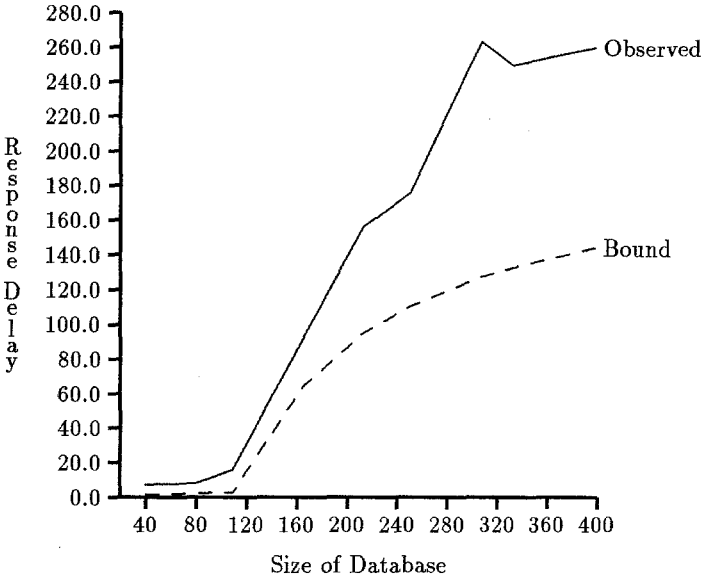
Figure 13.2 shows the results of increasing the number of data in the system while keeping the number of nodes constant. The simulation reported periodically on the mean of the response delays taken over just those queries which were issued since the previous report. The curve labeled "Bound" is an optimistic lower bound on response delay. The authors were encouraged by the resemblance between the curves formed by the observed data and the optimal bound. The response delays shown in the graph are the means of all the values which were reported after steady state was reached. A minimum connectivity of 0.4, an average datum size of 2, and a cache size of 10 were used. Intracluster links ranged in cost from 0 to 2; intercluster links ranged from 50 to 200[191].

Third Generation

The third generation algorithms improved on the second generation algorithms to the extent that most special case exceptions were removed. The simulation began growing the database dynamically over time and placing the primary copies of data at random sites within the cluster designated as the home cluster for that class of data. The static classification scheme induced by this arrangement is "lazier" than the previous scheme, which assigned a different class to data with different primary sites, since it is coarser and provides less precise information to the query router. Furthermore, sites were allowed to have arbitrary sized caches, including null caches. The realization that it is necessary maintain a consistent level of cache turnover across sites for cooperative algorithms to be at their best was the main inspiration for this generation and is primarily responsible for the generalization of the model. A series of tests was run to try to determine the best level of turnover to maintain, only to discover that the effect of varying turnover on efficiency is partially periodic, making it difficult to analyze numerically. A theoretical analysis of this effect is on the agenda for future work.

Using the new caching heuristic and the new "lazy" classification scheme, a study similar to the second generation study was performed. This time, the sizes of sites' caches were randomized as well. Data were collected for a series of scenarios in which

Figure 13.2: Response Delay vs. Database Size



an ever increasing number of sites was arranged into ten clusters. Ten basic data classes were used, but their home cluster assignments were allowed to overlap. A minimum connectivity of 0.8, an average datum size of 5, and an average cache size of 25 (varying uniformly from 0 to 50) were used throughout. Intracluster links ranged in cost from 0 to 2; intercluster links ranged from 50 to 200. All sites were configured to attempt to maintain turnover close to 600, meaning that the sites adjusted their caching to keep the average hint-adjusted age of data objects being uncached around 600 time units.

Figure 13.3 shows that response delay decreases as nodes are added. The delays used were those experienced with a constant influx of new data. Figure 13.4 shows the progress of response delay as the database is created, the system stabilizes, and then the long onslaught of new data begins. Note how delay is initially high since routing tables and caches start out empty, rapidly attains its steady state minimum, then as new data are constantly introduced, again begins to level off. The noise on the right side of the graph results from there being less simulation runs which took that long to complete. Finally, Figure 13.5 shows that the performance remains constant as the network becomes larger[193]. Performance is the quotient of the calculated optimal bound on response delay and the observed response delay; constant performance indicates that response delay changed at the same rate as the optimal bound.

13.6.5 Implementation

Using the methods developed during the simulation study, implementation of a networked information system will soon begin. The "prototype" will be constructed so that an incremental growth of the prototype will yield a fully implemented system for public use.

Figure 13.3: Sites vs. Response Delay

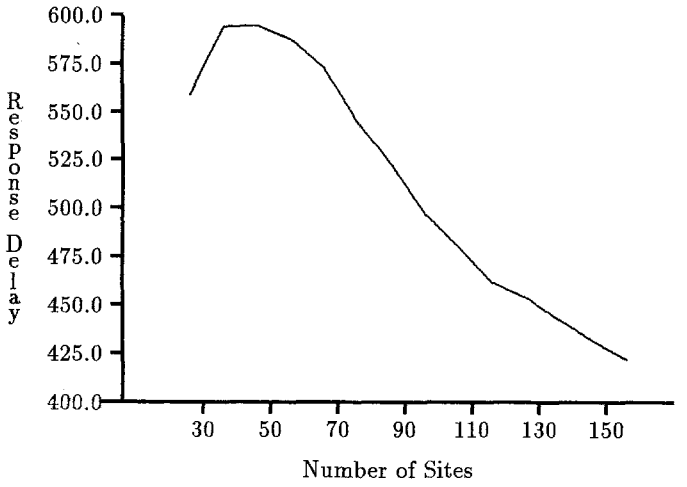


Figure 13.4: Time (DB Growth) vs. Response Delay

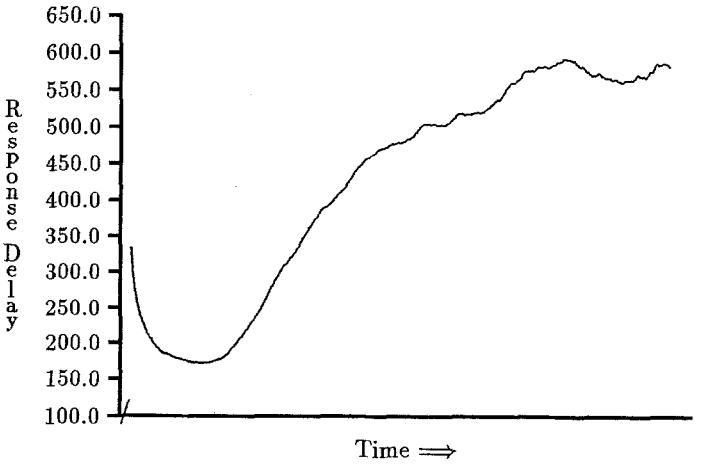
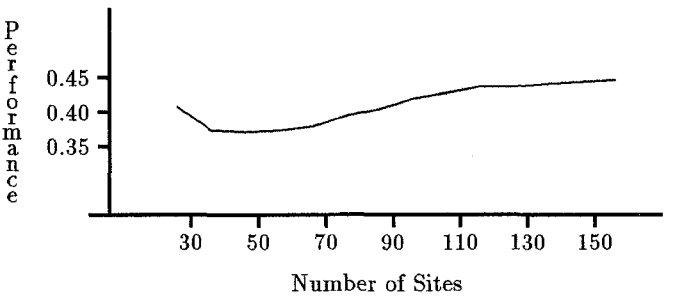


Figure 13.5: Sites vs. Performance



Although it will not take long to explain, the implementation of the system will require a significant amount of work. The heterogeneity of the information bases will necessitate the construction of several mediators, not the least of which will be a package capable of automatically utilizing anonymous FTP to access known Internet archives. Until such archives "join" the information system themselves, a member site must act as surrogate owner of each archive.

The information system will be built in two layers. The top layer will handle the routing of queries and the caching of data in a distributed manner. It will contain coarse grained classification software to determine the best way to route a query and general descriptor matching heuristics to decide when to try to answer a query with cached data. The bottom layer will consist of mediators and the local archive managers at each site owning an archive. It may be formed by an existing DBMS or by a specialized archive manager. Each will contain some combination of fine grained data classification, query processing, automatic indexing, and archive maintenance software. The bottom layer will be characterized more precisely in the following section.

Data Types and Classification Heuristics

When the prototype comes on line, it should include at least the following in order to provide a demonstration of its capacity to handle heterogeneity:

1. A corpus of ASCII text documents;
2. a collection of source code in several different programming languages;
3. a database of relational tables or other objects which is indirectly accessed through a DBMS;
4. a collection of bitmapped images with differing formats;
5. an FTP archive containing executables and other data.

Most of the workstations in the system should at least be able to determine which major data type a query requests. This coarse grained classification will be accomplished by inspection of the query for key terms. For example, source code would be flagged by the presence of the names of one or more programming languages or the word "source" in a favorable context. To avoid repetition of coarse grained classification, the general data type required by a query will be coded by the client at the querying site whenever possible. The most common data types will be assigned special codes.

Queries which cannot be parsed in this way will be forwarded without the special type coding.

Finer grained classification for each major data type will be accomplished as follows:

1. WAIS should serve as the main textual archive manager. Any additional text documents will be analyzed and indexed using an existing text processing algorithm, and matches between queries and documents will be made using an existing fuzzy retrieval method. Boolean querying may also be supported.

2. Source code will be classified by language and by purpose. The language is easily determined by scanning the file for constructs unique to different languages. The purpose of each source file will be kept in an index which is initially built automatically from comments found in the source, but modified as necessary by a human operator. Indexing solely by scanning comments is unreliable since commenting styles vary drastically from programmer to programmer, some of which may not comment at all. Matches between queries and index entries will be done as before.

3. When a DBMS is involved, an information system query might encapsulate a query to be presented to the DBMS or it might contain an unstructured question as usual. The encapsulated query case is trivial since it only needs to be identified as such; this is on the same level as determining the language of a source code file. However, unstructured questions must be translated into a query which is meaningful to the DBMS. Finding ways of translating arbitrary natural language queries into a database query language is beyond the scope of this work, as research is still actively being done in this area. However, a simple system may be implemented by which a user may ask how to access on line help for the specific DBMS and for the names and attributes of database objects which contain data relevant to some topic. It should also be possible to create a simple macroscopic mapping of database objects onto information system objects, such as to consider an entire relational table to be a single object[190, 35]. This simplifies the interface enough that the information system should be able to reliably construct queries for the desired objects and to cache replicas of them as usual. The drawback is that the user must accept the object in its entirety and send an encapsulated query to the DBMS if more selectivity is desired – but the simplified system is much more likely to succeed than an attempt to translate natural language into arbitrarily complex queries.

4. The format of an image can almost always be determined by scanning the beginning of the image file for signatures which are unique to different formats. As was discussed earlier, automatic determination of the content of images is just beginning to be actively researched and is in its extremely early stages. With the exception of automatically generated images such as satellite weather maps, the collections of images the system includes will almost certainly be manually indexed.

5. Many FTP archives provide informal indices for their available files. The system will interface with these archives by utilizing the existing textual indices and anonymous FTP in a fully automatic manner.

While it is small, this initial set of interfaces and data managers will provide a demonstration of the capabilities of the system. The FTP interface alone will provide a much needed service, removing the need to navigate unfamiliar file systems

and reducing the drain on central archive sites and the network through cooperative caching.

13.7 Concluding Remarks

While the information retrieval field was growing, the need for good information retrieval techniques was growing even faster. Some good information retrieval techniques are now available for small domains – but the current information domain is vastly larger than any of these techniques can handle. Just as small-scale information systems are becoming capable of dealing with natural language queries, the major information networks have grown so large that even a skilled user is sometimes unable to locate needed information.

To find a desirable solution to the Resource Discovery Problem, it is necessary to adapt and integrate information retrieval techniques for the large network environment. The fragmented and heterogeneous resources of the network must be used to support distributed information retrieval in a fair and efficient manner. Specialized information retrieval techniques must be allowed to operate within limited domains while researchers work to solve the greater problem of generalized retrieval. In this chapter, algorithms and architecture were presented for a distributed information system which can solve the following problems:

- The Resource Discovery Problem. By automatically routing queries to the sites which can answer them, the system removes the burden of resource discovery from the user.
- The heterogeneity problem. By treating all data as objects, it supports the integration of any information base whose data can be divided into units. By giving access to all these information bases simultaneously, a simplified global view is provided.
- The retrieval problem. Information retrieval techniques and data classification can easily be used by the system to handle queries expressed in natural language.

Acknowledgements

Information Retrieval is indeed a very rich field which has been under investigation for many years. The authors would like to recognize the very many researchers who worked long and hard to bring Information Retrieval to the level it has attained. They sincerely apologize to all those who could not be recognized due to space limitations.

Chapter 14

Efficient Parallel Recovery in Replicated Databases

Raj Tewari*

14.1 Introduction

Recovery in replicated databases is an important issues for fault-tolerant processing and non-stop computing. Recent database servers such as Oracle7 server are shipping with distributed and parallel options. The distributed option allows co-operative transaction processing by allowing selective replication of files in read-only and read-write modes on multiple servers and transactions that can execute across multiple servers are then co-ordinated by the two-phase commit operation. Moreover, the Oracle7 server allows parallel processing on multiprocessor machines such as the Sequent, to execute operations in parallel. It is of primary importance that the recovery algorithm for a distributed and parallel database server incorporate parallel operations to allow faster and efficient recovery. In this chapter we propose such an algorithm.

One way in which recovery can be facilitated is by using semantic knowledge of database operations to embed intelligence in the recovery algorithm. The idea of using semantic knowledge to increase concurrency was investigated by Garcia-Molina [208]. In an algorithm proposed in [208], transactions are divided into a collection of disjoint sets or classes, so that the transactions that belong to the same class are *compatible*. Transactions that are compatible can be allowed to interleave arbitrarily, whereas *incompatible transactions* (transactions that do not belong to the same class) are not allowed to interleave.

Commutativity of database operations is an important property that can be utilized for enhancing concurrency within transactions [24, 604]. In [24], a property known as *recoverability*, has been defined, which can be used to decrease the delay involved in processing non-commuting operations while avoiding cascading aborts. When an invoked operation is recoverable with respect to an uncommitted operation, the invoked operation can be executed by forcing a commit-dependency between the invoked operation and the uncommitted operation. In this way the transaction invoking the operation will not have to wait for the uncommitted operation to abort or commit. To enforce the serializability of transactions, the recoverability relationship between transactions is constrained to be acyclic.

A semantics based transaction management technique for managing replicated data was proposed in [337]. Conventional consistency control algorithms using one-copy serializability as the correctness criteria, are designed to deliver fast response times and

*Computer and Information Sciences Department, Temple University, Philadelphia, PA 19122

high availability for read-only transactions while sacrificing these goals for updates. They propose an approach that works well for both retrieval and update environments, by exploiting semantic properties of transactions. By subdividing transactions into various categories, and utilizing a commutativity property, they demonstrate that communication costs are minimized.

The *SHARD* (System for Highly Available Replicated Data) approach, described in [489], [490], [491] and an associated protocol called the log transformations protocol [67] are designed to support continued database operations in the event of site failures or network partitioning. To achieve this objective, SHARD sacrifices transaction serializability and uses timestamp ordering to ensure eventual mutual consistency. New algorithms for robust update propagation and a mutual consistency mechanism for complex update types such as increment and decrement was developed in [489]. SHARD also provides a basis for triggering application-specific compensating actions in the event of an inconsistency, and provides mechanisms for implementing concurrency controls that can be used selectively to reduce the probability of inconsistency at the expense of a controlled increase in response time. Basically, SHARD utilizes an optimistic approach for maintaining mutual consistency of copies of database objects.

We have previously proposed an approach for distributed databases on uniprocessor machines in [580, 581], that is intermediate between the mutual exclusion approach and the SHARD approach. Merging two partitions can be achieved by comparing only one copy of a data object in the first partition, with a copy in the second partition, together with the transaction history in both partitions containing all the transactions that performed operations on the data object. This information is sufficient to reconstruct the final value of the data object, since the classification of the transactions results in a sequence of operations that do not conflict with each other, hence they can be performed in parallel.

When updates or reads are made in a partitioned state, it is possible that nonserializable behavior may result. This can lead to inconsistent copies of the data object in different partitions. One suggested way to mitigate the effects of such inconsistency is through compensating transactions [490]. Compensating transactions depend on application semantics and would be determined by policy decisions for each organization.

The rest of the chapter is organized as follows. In the next section we present a high-level overview of the proposed *consistency control algorithm*. Details of the *basic merge protocol* are presented in the next section. We then present extensions to the basic merge protocol to include *parallelism* in the operations if possible on the machine architecture. The next section evaluates performance of the merge protocol using a simulation model, and a comparative performance analysis is presented, comparing the performance of the merge protocol to the previously proposed log transformation protocol.

14.2 Consistency Control Algorithm

The algorithms proposed by us in this chapter have the objective of attaining eventual consistency in the final database state, while sacrificing mutual consistency in intermediate states. We need to differentiate between *consistent* and *correct* data. A distributed database with multiple copies of data objects preserves mutual consistency by ensuring that all copies of a data object have the same value. If strict consistency is to be preserved queries and updates can be allowed in only one partition if network

failures occur, and disallowed in other partitions. If a database sacrifices consistency, it needs to ensure that the results of read and write operations are *correct*, in the sense that they are equivalent to the results that would have been obtained had consistency been preserved. If the results are not correct subsequent compensating mechanisms have to be used to ensure correctness in the final state.

A fully connected and operational distributed database system (DDBS) is non-partitioned. Due to site and/or communication link failures the system can be partitioned into two or more disjoint connected components. In general, a slow site or link is indistinguishable from a partition. A group of sites can determine autonomously whether there is a partitioning by polling the sites in the original configuration and using *timeouts* to determine any non-responding site(s). This process can be repeated for any further partitions, provided each site in every partition maintains a configuration vector.

The configuration vector maintains the set of sites that are in the same connected component and are reachable from the home site. The requirement of a configuration (or connection) vector is a common one in most systems with replicated data. The configuration vector may be updated when updates to data objects are propagated. For more details of the operation of the connection vector, please see [290, 291].

The rules for maintaining consistency of copies of data objects in a partition are read-one-write-all copies (ROWA). As long as all sites in the DDBS are operational and able to communicate with each other, the ROWA rule ensures data consistency. Notice that the ROWA rule by itself is not sufficient for maintaining serializability [136]. An underlying concurrency control mechanism has to operate to ensure serializability of transactions within partitions. We assume that a strict two-phase (2PL) concurrency control protocol is used within all partitions. The 2PL protocol does not operate across partitions because there is no communication between different partitions. The consistency of the DDBS is maintained by a *merge protocol* at the time of merging partitions.

If the database system partitions into two groups, then the effects of transactions in one partition will not be seen by the other partition. Within a partition, data objects will be synchronized by the ROWA rule, but there may be inconsistency across partitions. The problem we are addressing is to merge all data objects in the partitions optimally. Thus we seek to find a merging process that yields an optimal set of transactions for merging utilizing semantic knowledge.

14.3 Parallel Merge Protocol

In this section we will describe an efficient merge protocol that merges two partitions after a communication link repair has connected the two partitions. We consider the simplest possible case of one data object and two partitions. This case can be extended to handle the general case of many data objects and n partitions as shown later in Section 4.

We make the following assumptions:

1. Each site maintains a local clock that tags each transaction with a unique timestamp.
2. Each site maintains independent backup logs that record the *action* of each transaction on each data object (i.e. the backup log records each operation and its associated time stamp). This is in contrast to the more common backup logs that maintain *before* and *after* values of each data object.

3. The logical database schema remains unchanged during a partitioning.
4. The replicated database system has a mechanism for implementing parallel operations, either by using specialized parallel processing database machines, or by using software simulation of parallel operations. This assumption is easily realized in practice in modern high-performance databases that utilize parallelism for intra-query or inter-query operation speedup. For an excellent recent survey of parallelism in databases refer to [409].

The first assumption requires local clocks at each site (processor) which is common in distributed computing systems. A global order of transactions executed in each partition can then be determined, even when there is no globally synchronized clock as shown by Lamport in [348]. The second assumption says that the backup logs record operations performed by each transaction on every data object, instead of the before and after values of every data object. This can be accomplished without excessive additional processing. In fact, transactions were originally defined in the context of maintaining consistency by Eswaran et al. [182] as a sequence $\mathbf{T} = ((T_i, a_i, e_i))_{i=1}^{n}$ of n steps where T is the transaction name, a_i is the action at step i and e_i is the entity acted upon at step i. This is just the information that we maintain in the logs.

The following terminology will be used in the description of our protocol:

Definition 14.3.1 *A* **partition log** *is the sequence of locally ordered transactions that are executed site in a partition P_i.*

Definition 14.3.2 *An* **initial merge log** *is a sequence of transactions that must be executed in a partition P_i to bring the database to a globally consistent state.*

Definition 14.3.3 *A* **target log** *is a sequence of globally ordered transactions constructed from the locally ordered transactions in the partition logs of each site.*

Definition 14.3.4 *A* **merge_undo transaction** *is a transaction executed at merge time of partitions, to reverse the effects of an out of sequence transaction in the global ordering of transactions. The merge_undo transaction of a transaction T is denoted by T'. Further, the log $(T, T') = log\ (T', T) = null$ effect on the database. An* **merge_undo transaction log** *is a globally ordered sequence of merge_undo transactions.*

Definition 14.3.5 *A* **merge_redo transaction** *is a transaction executed at merge time to achieve the effects of a transaction that was rolled back for merging. Merge_redo transactions are the same as the original transactions, but they are scheduled for execution at merge time of partitions. A* **merge_redo transaction log** *is a globally ordered sequence of merge_redo transactions.*

Definition 14.3.6 *A* **semantically equivalent transaction** *is a transaction that has the same effect on the database as a sequence of transactions. If $(T_1, T_2, \ldots, T_n)$ is a sequence of transactions, then the semantically equivalent transaction of this sequence is denoted by $T_{1,2,\ldots,n}$.*

It should be noted that our definitions of *merge_undo and merge_redo transactions* and their associated transaction logs are different from the standard definitions of undo and redo[593]. The operation intended by the new merge_undo and merge_redo

transactions is very similar to that intended by the original undo and redo defined in[593], but has different semantics.

The initial state upon partitioning of the DDBS into two partitions P_i and P_j is a consistent state. PL_i and PL_j are partition logs that bring the two partitions into inconsistent intermediate states S_i' and S_j' (there could be finitely many intermediate states; however, we concentrate only on the last inconsistent states S_i' and S_j', which are the states just before the partitions are repaired).

Our objective is to find optimal merge logs ML_i and ML_j, which will run in partition i and j respectively, and will result in a consistent state S_f in both partitions. We define initial merge logs IML_i and IML_j as:

$$IML_i = (PL_i', T)$$

$$IML_j = (PL_j', T)$$

where T is the target log defined by the global transaction ordering mechanism we have assumed. T contains all the *merge_redo* transactions, hence it is called the *merge_redo partition log.* PL_i' and PL_j' are the *merge_undo partition logs*, consisting of the inverse transactions of all transactions in PL_i and PL_j.

We will propose a protocol that results in greater savings through more efficient processing than the *log transformation technique* [67]. The log transformation mechanism seeks to classify transactions into the following types:

- Overwrite pairs
- Commutative pairs
- Conflicting pairs

and then proceeds to simplify an initial merge log through successive log transformations that replace a pair of overwrite transactions by the overwriting transaction. Pairs of transactions that are inverses of each other, and occur successively are canceled.

We propose exploiting transaction semantics further to gain significant savings, that are not possible using the log transformation technique. Properties of arithmetic operations such as associativity, and distributive laws can be utilized to obtain further reductions in the initial merge log.

An example of a banking database will clarify our approach, and also point out situations where the log transformation approach will not ensure any savings, but our approach will achieve substantial savings in transaction processing.

Example: Assume that the database partitions into two partitions i and j. We will follow the transaction activity related to one data object, say a checking account with an initial balance of $200 before partitioning.

The transactions executed in the two partitions and the corresponding changes in the balances are shown in Table 14.1. For the duration of the partition transaction T_1, T_2, T_4, T_5, T_8 and T_{10} were executed in partition i and transactions T_3, T_6, T_7, T_9 were executed in partition j. The global balance is known to us from the history of the transactions but the global balance is not known to either of the two partitions. The objective of the merge protocol is to arrive at globally consistent (correct) database objects; in this case the account balance. Partitions i and j have inconsistent account balances. The balance in partition i is $650 before the merge, and the balance in partition j is $100, whereas, the globally consistent balance that we seek to arrive at is $545. This would be the balance if all the transactions were executed in their timestamp order. The global balance is shown in Table 14.1 as G(b).

Table 14.1: Partition Logs for Partitions i and j

Partition i		Partition j		G(b)
Transactions	Bal(i)	Transactions	Bal(j)	
T_1 = deposit(200)	400		200	400
T_2 = query account	400			400
		T_3 = withdraw(100)	100	300
T_4 = deposit(200)	600			500
T_5 = withdraw(100)	500			400
		T_6 = withdraw(50)	50	350
		T_7 = deposit(100)	150	450
T_8 = addinterest(10%)	550			495
		T_9 = withdraw(50)	100	445
T_{10} = deposit(100)	650			545

We classify the operations of the transactions into the following types:

- Collapsible
  - Associative
  - Distributive
  - Commutative
  - Other non-arithmetic
- Non-collapsible

Associative operations are those that can be combined with each other into one semantically equivalent operation. If X, Y, Z are operations then the notion of associativity that we are using is:

$$(X \odot (Y \odot Z)) = ((X \odot Y) \odot Z)$$

The operator $\odot$ stands for any associative operation. In the simplest case this operator could be the algebraic associative operation for arithmetic database objects. In more complex cases this operator could stand for associative operations on non-arithmetic database objects which could be combined using semantic information. If by the above manipulation, we can combine the operations X and Y into one operation, then we have achieved a reduction in the initial merge log.

Similarly, distributive operations on X, Y, Z are:

$$X \otimes (Y \oplus Z) = (X \otimes Y) \oplus (X \otimes Z)$$

The operators $\otimes$ and $\oplus$ could be algebraic in the simplest case or could be non-algebraic but distributive by other semantic knowledge. If we have database operations as depicted by the right hand side of the above equation which cannot be reduced in that form, it is possible that they could be reduced by converting them to the form of the left hand side.

One of the key points in the above discussion is that it is possible that not all operations involve arithmetic operations. There could be non-arithmetic operations on the database that could be combined by looking at their semantics. Thus, the category *other non-arithmetic* contains database operations that are of non-arithmetic nature,

yet can be combined knowing their semantics by one of the associative, distributive, commutative or other semantic operators (which are application specific). An example of a non-arithmetic operation is the location of a vehicle in a car rental system. The car may be rented at one location and returned at another location, which may, at some instant of time, be isolated from the originating location. To reconcile the return of the car, and its location in the two partitions (the originating partition and the returning partition), a non-arithmetic operation, namely updating the location variable, is required.

Non-collapsible operations are those that cannot be combined with other operations based on their mathematical properties *and* other available semantic information about them. The thrust of our protocol is that if by using available mathematical and semantic information about database operations, we can collapse a sequence of operations in the initial merge log into one equivalent operation, we can perform the merge process more efficiently, by substituting one equivalent write on the database in place of each individual write operation corresponding to each merge operation.

Operations of the type deposit, withdraw and query can be represented by functions of the form

$$f_1(x) = c_1 \pm x$$

where c_1 is a constant. Hence these are associative and any number of deposits, withdrawal and queries appearing in the initial merge log can be collapsed into one equivalent operation.

Operations of the type add interest can be represented by a function of the form

$$f_2(x) = c_2 + c_3 x$$

These operations cannot be combined with other operations since they are not associative, distributive or commutative with deposit, withdrawal and query.

The initial merge logs for the banking example will be defined for each partition i and j as:

$$IML_i = [T'_{10}, T'_8, T'_5, T'_4, T'_2, T_1, T_2, \ldots, T_{10}]$$

and

$$IML_j = [T'_9, T'_7, T'_6, T'_3, T_1, T_2, \ldots, T_{10}]$$

which reflects the brute force approach of *undoing* all transactions run in each partition, and then *redoing* all transactions in the globally serializable order, found by the global ordering algorithm.

We will first use the log transformation protocol to get a transformed merge log for each partition, as depicted below:

Partition i

$$\begin{aligned} ML_i &= [T'_{10}, T'_8, T'_5, T'_4, T'_2, T'_1, T_1, T_2, \ldots, T_{10}] \\ &= [T'_{10}, T'_8, T'_5, T'_4, T'_2, T_2, \ldots, T_{10}] \\ &= [T'_{10}, T'_8, T'_5, T'_4, T_3, \ldots, T_{10}] \end{aligned}$$

Partition j

$$ML_j = [T'_9, T'_7, T'_6, T'_3, T_1, T_2, \ldots, T_{10}]$$

The log transformation technique would result in 12 transactions in partition i, and 14 transactions in partition j to bring both the partitions to a consistent state, in the

first pass of the reduction process. In the first pass all reductions are made using the property of double node deletion. Further reductions are possible using the commutativity property and constructing merge graphs that identify further reductions. But this requires considerable computational effort.

Our proposed parallel merge protocol utilizes the semantic information provided by the complete history of all transactions available to in the logs, as follows:

1. Scan *merge_undo* transactions in reverse order till a non-collapsible transaction is encountered. Accumulate the effects of all the collapsible transactions (deposits and withdrawals) till this point in one *semantically equivalent transaction.* Continue scanning in this manner until the next non-collapsible transaction and repeat the process of constructing another semantically equivalent transaction. This process is repeated until there are no more *merge_undo* transactions.
2. Scan *merge_redo* transactions in order till a non-collapsible transaction is encountered, and accumulate the effects in semantically equivalent transactions as in (1) above. Repeat this process till the list of transactions in the merge_redo log is exhausted.

Continuing our example, we obtain the following reductions using our merge protocol:

Partition i

$$\begin{aligned} ML_i &= [T'_{10}, T'_8, T'_5, T'_4, T_3, T_4, \ldots, T_{10}] \\ &= [T'_{10}, T'_8, T'_{5,4}, T_{3,4,5,6,7}, T_8, T_{9,10}] \end{aligned}$$

Partition j

$$\begin{aligned} ML_j &= [T'_9, T'_7, T'_6, T'_3, T_1, T_2, \ldots, T_{10}] \\ &= [T'_{9,7,6,3}, T_{1,2,3,4,5,6,7}, T_8, T_{9,10}] \end{aligned}$$

Here $T'_{5,4}$ is the semantically equivalent transaction for T'_5 and T'_4, and its effect on the database is the same as a deposit of \$100. Similarly, $T_{3,4,5,6,7}$ is the semantically equivalent transaction corresponding to transactions T_3, T_4, T_5, T_6, T_7, and its effect on the database is the same as a deposit of \$50. We have reduced the IML_i with 16 transactions to a semantically equivalent log with 5 transactions, whereas the log transformation approach was able to achieve only a reduction to 12 transactions. Using our merge protocol in partition j, we obtained an optimal merge log containing only 4 transactions, as opposed to no possible reductions using log transformations for IML_j in one pass. Note that these results are based on all reductions achieved in the first pass. The log transformation protocol will discover all the reductions possible eventually, but this will be at the expense of constructing merge logs and then reducing them. Each of the merge log graphs will contain as many nodes as the number of transactions in the merge log, and conceivably the log transformation approach will result in large graphs.

The *advantage* of combining several transactions into one at merge time is that the final effect of the collapsed transactions is written and committed to the secondary storage only once. This is a significant benefit over writing and committing each traditional undo and redo transaction individually to the database, resulting in a large number of secondary storage accesses, that slow down database processing. Further, a sequence of semantically reduced transactions can be executed in parallel, since they are independent of each other.

It should be noted that our technique is different from the *group commit* technique. The group commit technique as described in [330] is performed at the time of regular transaction processing and not at the time of recovery. Moreover, the group commit technique does not take semantic knowledge of transactions into account when deciding which transactions make up the group to be committed. In contrast, our algorithm is executed at the time of recovery of sites or partitions, and takes semantic knowledge that is available in the merge_logs to decide which transactions can be grouped and then executed in parallel.

14.4 Extension of the Parallel Merge Protocol to the General Case

We have focused on one data object and two partitions for the sake of exposition. The general case of many objects and many partitions can be handled by our protocol. The extension of the merge protocol to the case of more than one data objects and n partitions is described below.

The final value of each data object can be efficiently reconstructed using the merge protocol from the history of transaction operations available in the *partition logs*. Recall that we maintain each operation on the data objects, instead of before and after values. We can view the merge protocol as operating sequentially on each data object in the database, taking into account only those relevant transactions that operated on that data object. Thus the set of relevant transactions for merging may be different for different objects, the only requirement being a global ordering of transactions across all partitions.

We consider only simple partitioning of the database at any instant of time, and exclude the possibility of byzantine failures. The merge protocol can then be applied successively to each binary partition to handle a general n-way partitioning. Thus any arbitrary partitioning of the database can be described in terms of a finite sequence of binary partitionings that can be linearly ordered in time.

One important question is that of failure during recovery. Since our algorithm does not use a voting technique to maintain consistency, the algorithm holds even if there is a failure during recovery. At the time of performing the merge algorithm, if there is a new failure at the site where the merge algorithm is running, we can view the situation de novo, in the sense that when recovery occurs again, the whole merge process starts again from scratch. In this respect, our algorithm is at least as good as other algorithms for recovery.

14.5 Incorporating Parallelism in the Merge Protocol

Once the operations in the initial merge logs have been classified into collapsible and non-collapsible, the new proposed way of handling the merge processing is to use parallel execution of the operations in the final merge log, since the operations in the final merge log do not interfere with each other. However, this is possible only if the machine architecture or a software emulation allows parallel execution of operations in the final reduced merge logs. With parallel concurrent writes, there is reason to

believe that ths speedup obtained in merging and consequently in the recovery process can reach speeds approximating real-time operation.

Parallelism is significantly faster in the case of operations involving multiple data objects, since frequently the processing involves join operations, which are costly in distributed databases. It is of great importance to utilize opportunities for parallel execution of recovery operations to speed up the recovery process.

14.6 Performance Analysis of the Parallel Merge Algorithm

We compare the performance of the parallel merge protocol proposed by us with the log transformations protocol proposed in [67] using a simulation model. The objective of the performance analysis is to obtain the efficiency measure for both protocols. The efficiency measure is defined as the percentage reduction obtained in the initial merge logs using the protocol whose performance is being evaluated.

Initial merge logs are generated containing on an average 10 to 90 percent non-collapsible transactions (correspondingly, 90 to 10 percent transactions are collapsible). Transactions are randomly classified into N (non-collapsible) or C (collapsible) and filed into the initial merge log. Once an initial merge log is generated, reductions are performed on it using our protocol and the log transformations protocol. Since our protocol is a one pass protocol, we simulate the reductions obtained by both the protocols in one pass.

The implementation of the log transformations (LT) protocol requires a graph representation and operations on a graph that will eventually discover most of the reductions obtained by us but at considerably greater expense, because of the complexity of their implementation. The implementation of the LT protocol results in a multi-pass approach, as is made evident by the examples presented in [136]. In order to perform a fair performance evaluation, we compare the performance of the two protocols using all reductions obtained in one pass only.

For each level of NC (percentage of non-collapsible transactions in the initial merge log), varying from 10 to 90, and for each initial merge length (IML) varying from 10 to 100, replications of the experiment are performed to achieve statistically significant results (95 % confidence intervals) Thus, each of our point estimates for the proportion represents 385 replications. The computer programs were written in the *C* programming language.

For any given initial merge log length, the efficiency of the both our merge protocol and the LT protocol increases as the value of NC decreases. This can be explained by the fact that as the value of NC decreases, there are less non-collapsible transactions on an average in the initial merge log, and consequently greater reductions are possible.

We performed the simulations for NC = 10 to NC = 90, since this spans the feasible region of transaction mixes. Here, we present the results for NC = 10 to NC = 40, since the reductions are more obvious. NC = 70 to NC = 90 follow the same pattern, but at a reduced level. The results from the simulation model indicate that the reductions obtained using the parallel merge protocol are significantly higher than those obtained by the log transformation technique for NC = 10 to 40 level.

The results obtained in the simulation were from randomly generated transaction streams. There are transaction streams that could lead to worst case performance results. If we have a transaction stream that alternates strictly between C and NC

transactions, then neither our merge protocol nor the LT protocol will be able to find any reductions thereby implying that the worst case behavior of both the algorithms is the same. This is, however, true for any protocol utilizing semantic knowledge of transactions, when faced with a transaction stream that strictly alternates between NC and C transactions.

14.7 Conclusion

In this chapter, we have proposed a new parallel recovery algorithm for replicated databases that are subject to site failures and network partitioning. The recovery algorithm utilizes available semantic information about the operations of transactions to classify the operations into collapsible and non-collapsible operations. This classification allows us to obtain reductions in merge logs to speed the recovery process when partitions are repaired and merged together by exploiting parallelism. The recovery algorithm proposed by us is also suitable for systems with long duration or nested transactions, such as CAD/CAM databases and emerging multi-media applications.

Our work is most closely related to the proposal for recovery in centralized databases in [331]. We have considered a replicated database environment and performed simulation experiments to test the efficiency of our recovery protocol. Results of this performance evaluation indicate that our recovery protocol provides improved performance than a previously proposed protocol based on the log transformation technique.

One of the important issues for future research is how to pre-classify operations of transactions and define compensating operations. Certainly an approach that automatically collects this information from the database system is preferable over one that requires users to specify this information. However, it may not be possible to obtain the complete semantics of all operations automatically through the database system, because knowledge outside the application domain may be required to infer the semantics.

Another significant research area is the investigation of parallel machine architectures and their influence on recovery strategies. It can be conjectured that the parallel recovery algorithm will be different for a connection machine type of massively parallel architecture, than an algorithm for a Cray Y-MP type of vector processor architecture. The influence of architectures and support for parallel operations in databases of the future is key to handling grand challenge applications such as the human genome project.

Chapter 15

Document Allocation In Multiprocessor Information Retrieval Systems

Hava T. Siegelmann* Ophir Frieder†

15.1 Introduction

The volume of data accessible online is steadfastly increasing. This growth will shortly render conventional information retrieval systems helpless in terms of responding to users' queries within an acceptable period of time. The increase in the volume of data available online stems from improvements in data communication and storage technologies which in turn spurred an increase in the number of information providers, and the volume of data provided by each information source.

To provide the computational demands needed to deliver acceptable response times in the wake of voluminous information retrieval databases, numerous large-scale (scalable) multiprocessor information retrieval systems [1, 23, 131, 455, 454, 513, 545, 546, 544, 550] have been proposed. A scalable multiprocessor information retrieval system generally necessitates the exploitation of a distributed memory architecture as a large number of processors is currently not possible in a shared-memory configuration. A distributed memory system, however, introduces the problem of mapping the data onto the given architecture. (A poor document mapping onto an architecture results in high access and retrieval times.) We refer to this problem as the Multiprocessor Document Allocation Problem (MDAP), and develop a heuristic approach based on Genetic Algorithms which yields a near optimal mapping.

Throughout this research, the term "document" is used generically to represent any datum to be accessed. That is, a "document" may take various forms, ranging from simple text to highly detailed photographs to voice and animated video. For example, in a hospital setting, patient data includes EKG readings, blood sample measurements, x-rays, patient daily charts annotated in handwriting by the physician, patient history, and possibly, accounting information. Such a system must incorporate pictures (x-rays), contiguous electronic signals (EKG readings), hand written free formatted annotations (doctors' notes), and formatted data such as billing. (For a detailed evaluation of the design issues of medical databases, see Allen and Frieder [14].)

*Department of Computer Science, Rutgers University, New Brunswick, NJ 08903, U.S.A., siegelma@yoko.rutgers.edu

†Department of Computer Science, George Mason University, Fairfax, VA 22030, U.S.A., ophir@cs.gmu.edu

We assume a clustered document database for the following reasons. First, in view of an appropriate data model, highly data intensive applications typically lend themselves to a clustered approach. In the medical arena, for example, a doctor may wish to capture:

- the entire medical history for a given patient (clustering based on patient identity);
- information concerning a given disease (clustering based on the type of disease);
- diseases exhibiting a given type of symptoms (clustering based on symptoms);
- all available data regarding common ailments for a particular region of the world (clustering based on geographical location).

Second from a standpoint of a compromise in terms of storage and processing times, a clustered approach reduces the storage overhead introduced by an indexed file organization. Haskin [243] proclaimed that an indexed file organization can introduce a storage overhead of up to roughly 300%. As compared to a full-text or signature analysis technique, a document clustering approach results in significantly lower search times.

In the context of a clustered document database, a proper solution to MDAP is any mapping of the document onto the processors such that the *average cluster diameter* is kept to a minimum while still providing for *an even document distribution across the nodes*. An even document distribution is needed to provide an equal workload across the processors. As shown in Frieder and Baru [198], a low average cluster diameter increases the inter-query parallelism. (The distribution of clusters along large diameters introduces high traffic in the communications network, increasing the contention and reducing the system throughput - see Frieder, et al. [199].) Returning to the medical database example described above, a solution to MDAP may involve the mapping of patient records across the nodes of a given distributed multicomputer such that efficient retrieval is supported for all queries that view the database as being clustered based on each of the above four clustering views.

The remainder of this chapter is organized as follows. Initially, a proof demonstrating that MDAP is NP-Complete is provided. The Mapping Problem and several of its derivatives are similar to MDAP, and hence, a brief review of solutions addressing these problems is provided in Section 3. Also provided in Section 3 is a brief discussion of prior relevant multiprocessor information retrieval systems. In Section 4, our genetic-based, document allocation algorithm is described. The theoretical foundation on which our approach is based, a proof of convergence of the derived allocations to a good mapping, is given in Section 5. An experimental evaluation of the proposed algorithm is presented in Section 6. We conclude in Section 7.

15.2 MDAP is NP-Complete

An instance of MDAP consists of a homogeneous distributed memory architecture with n nodes, $X_i, 0 \leq i \leq n-1$, and partitions of the documents $D_i, 0 \leq i \leq d-1$, called clusters, C. Each cluster $C_i, 0 \leq i \leq c-1$, represents a set of all the documents associated with it. The distance between a pair of nodes is defined as the cost of sending a packet from node i to node j and is represented by the internode communication cost matrix, $M_{ij}, 0 \leq i, j \leq n-1$. The diameter of a cluster is the maximum distance between any pair of nodes that contain documents belonging to the given cluster.

MDAP requires the documents to be evenly distributed across the nodes such that the sum over all cluster diameters is minimal. As the sum of the cluster diameters is reduced, the total communication traffic is minimized. An even distribution is required to achieve a balanced workload across the processors.

Here, we show that MDAP is inherently hard. We do so by proving that it belongs to the class NP-complete. Problems including in this class are considered too hard to be solved in practice, even when allowing for much computation resources. Instances of such problems, hence, can at best only be solved approximately.

(We provide in this paragraph a brief overview of the class NP-complete, a well studied issue in the field of computational complexity theory: NP is the class of decision problems, for which affirmative solutions can be verified polynomially. This class includes for example the Satisfiability problem; i.e., given a Boolean formula, decide whether there is an assignment of the variables so that the formula yields the True value. Here, an assignment of the variables can be verified as satisfying in polynomial time. NP-complete is a class of "hard" problems. A problem is in the class NP-complete if it is in NP and is at least as hard as any other NP problem (i.e., the existence of a polynomial time algorithm solving it implies that all other NP problems are solvable in polynomial time as well). Satisfiability is in this class, as well as the Traveling Salesman Problem, Multiprocessor Job Scheduling and many others. A detailed discussion about NP-completeness and many examples can be found in Gary and Johnson [212].)

Here, we concentrate on the decision form of MDAP: Instead of finding an allocation that satisfies some constraints, we ask if such an allocation may at all exist. Formally, **MDAP** is defined as follows:

INSTANCE:

- A distributed memory architecture with:
  - Nodes (PEs): $X = \{X_i | 0 \leq i \leq n-1\}$;
  - Communication cost: $M_{ij}, (0 \leq i, j \leq n-1)$;
- A clustered document domain with:
  - Documents: $D = \{D_i | 0 \leq i \leq d-1\}$;
  - Clusters: $C = \{C_i | 0 \leq i \leq c-1, C_i \subseteq D\}$;
- A real value bound: B.

QUESTION:

Decide whether there is an allocation map

$$\mathcal{A} : D \mapsto X$$

of the documents to the processors that satisfies the following conditions:

1. Let X_i be a node. Define the number of documents mapped onto this node by the allocation $\mathcal{A}$ as

$$\mu_{\mathcal{A}}(X_i) = |\{D_j \in D \mid \mathcal{A}(D_j) = X_i\}| .$$

Then, for all i $(0 \leq i \leq n-1)$, $\mu_{\mathcal{A}}(X_i) \leq \lceil \frac{d}{n} \rceil$.

2. Let C_j be a cluster of documents. Define the diameter of this cluster under a given allocation $\mathcal{A}$ as:

$$diameter_{\mathcal{A}}(C_j) = \max\{M_{\mathcal{A}(D_k)\mathcal{A}(D_l)} \mid D_k, D_l \in C_j\} .$$

Then,

$$\sum_{j=0}^{c-1} diameter_{\mathcal{A}}(C_j) \leq B .$$

Theorem 1. MDAP is NP-Complete.
Proof:

1. Assume an instance of MDAP together with an allocation map $\mathcal{A}$ that satisfies the required conditions. Then, $\mathcal{A}$ can be verified as an affirmative solution using a polynomial time algorithm. Therefore, MDAP is in NP.
2. To prove that MDAP is complete in NP, we reduce the NP-Complete problem, Binary Quadratic Assignment Problem (BQAP), defined in Garey and Johnson [212] to MDAP in polynomial time. We show that a solution to an instance of BQAP exists if and only if there is a solution to the associated MDAP instance. This means that a polynomial algorithm to MDAP implies one for BQAP and, hence, for all of NP. Thus, MDAP is complete in NP.

   Binary Quadratic Assignment Problem (BQAP):
   INSTANCE:

   - Non-negative costs: $b_{ij} \in \{0, 1\}$, $b_{ij} = b_{ji}$, $1 \leq i, j \leq g$;
   - Distances: m_{kl}, $1 \leq k, l \leq h$;
   - A bound: $Z \in Z^+$.

   QUESTION:

   Is there a one to one function $f : \{1, 2, 3, ..., g\} \rightarrow \{1, 2, 3, ..., h\}$ such that $\sum_{i \neq j} b_{ij} * m_{f(i)f(j)} \leq Z$?

   Given an instance of the BQAP, define an associated instance of MDAP as follows:

   - An architecture with h processors;
   - A cost matrix $M_{kl} = m_{kl}$, $1 \leq k, l \leq h$;
   - A set of g documents;
   - A set C of two-document clusters, such that $\{D_i, D_j\} \in C$ iff $b_{ij} = 1$;
   - A bound $B = \frac{Z}{2}$.

   This transformation takes polynomial time in the size of the input. It is easy to verify that an MDAP allocation exists to this instance if and only if there is a function f to the BQAP instance. Thus, MDAP is NP-Complete. ■

15.3 Related Efforts

A review of previous mapping algorithms (Section 3.1) and related multiprocessor information retrieval efforts (Section 3.2) is provided.

15.3.1 Previous Approximations of the Mapping Problem

The Mapping Problem and some of its derivatives are NP hard; therefore, heuristic algorithms that approximate optimal solutions were developed. Some of these approaches [69, 70, 352] dealt, in some manner, with the mapping of a communicating set of processes onto an architecture with a fixed interconnection topology. This problem is similar to MDAP in that both problems must map a set of tasks (items) onto a given architecture. However, the goals of the above efforts differ from MDAP as MDAP does not aim to maximize the amount of concurrent interprocess communication, but instead, aims at reducing the total communications diameter of its logical tasks (clusters).

Bokhari [69] introduced a pairwise-exchange heuristic algorithm that accepted as input two adjacency matrices representing graphs G (set of communicating processes) and G' (target architecture). Using the cardinality of the number of communicating pairs that directly communicated with their neighbor as the objective function, Bokhari developed and evaluated an algorithm that mapped graph G onto graph G'.

Lee and Agrawal [352] extended that Bokhari's effort by developing objective functions that more accurately quantified the communication overhead. Using a set of objective functions (parametric equations) that corresponded to the cost associated with the given mapping, Lee and Agrawal precisely measured the optimality of the derived mapping. The main limitation in their approach was that it only employed a fixed path routing scheme for the network traffic.

Bollinger and Midkiff [70] used a two-phase simulated annealing algorithm to map a logical system onto a physical architecture. The first phase, process annealing, assigned the processes onto the physical nodes. The connection annealing phase mapped the logical connections onto the network data links so as to minimize communication link conflicts. This effort improved upon Lee and Agrawal [352] in that it utilized the information concerning the actual routing rules.

Du and Maryanski [166] attacked a variation of the mapping problem. This variation concerned the allocation of data in a dynamically reconfigurable environment. The allocation algorithm used a set of "benefit" functions and a greedy search algorithm. The underlying execution architecture was based on a client/server model (a heterogeneous system). Although their problem more closely resembles MDAP, as the underlying architectural model significantly differs from the MDAP execution environment, their assumptions are not relevant to MDAP.

15.3.2 Related Information Retrieval Systems

Distributed-memory information retrieval systems have been investigated as a mean of providing short response times to users' requests. Some of these systems include various efforts on the Connection Machine [23, 545, 546, 544], on the Distributed Array Processor (DAP) [455, 454], on a network of Transputers [131], and on Hypercube systems [513]. Both the commentary on and the extensions of the Connection Machine efforts [23, 480, 546, 544, 550] as well as the hypercube [513], DAP [455, 454] and Transputer [131] efforts have all addressed the notion of data organization in the search and retrieval scheme employed.

Stone [550] demonstrated analytically that, by indexing keywords, a uniprocessor system with comparable memory to that of the Connection Machine employed in the Stanfill and Kahle effort [545] can achieve similar user retrieval response times to those times reported in [545]. Via keyword indexing, the volume of data that had to be

searched was reduced, significantly reducing the total I/O processing time. A parallel index-based retrieval effort on the Connection Machine was later reported in Stanfill, Thau, and Waltz [546] and in Asokan, Ranka, and Frieder [23]. Additional parallel text retrieval search methods are described in Salton and Buckley [480].

Various descriptions of efforts that focus on the organization of data for the DAP system, appear in the literature. In Pogue and Willett [455], an approach using text signatures is proposed and evaluated using three document databases comprising of roughly 11000, 17000, and 27000 documents. Pogue, Rasmussen, and Willett [454], describe several clustering techniques using the DAP. Both reports clearly demonstrate that if a proper document mapping onto the individual Processing Elements (PEs) is established, the DAP system readily achieves a high search rate. However, an improper mapping results in a poor search rate stemming from the inability of the DAP system to access the documents.

Cringean, et al., [131] describe early efforts aimed at developing a processor-pool based multicomputer system for information retrieval. The physical tested hardware consists of an Inmos Transputer network. To reduce the volume of data accessed and hence the total query processing times, a two phase retrieval algorithm is proposed. The initial phase acts as a filter to retrieve all potentially relevant documents. By using text signatures, the majority of the non-relevant documents are eliminated from further processing. This filtering of documents vastly reduces the volume of data processed in the compute-intensive second phase. (Some non-relevant documents are selected as a consequence of false-drops. False-drops are common to all text signature analysis approaches.) In the second phase, full text search is performed. The two phase algorithm is yet another example that emphasizes the need for intelligence in the organization and retrieval of documents.

Finally, Sharma [513] describes a hypercube-based information retrieval effort. The results presented are based on the timing equations provided. To reduce the volume of data read, Sharma relies on the fact that the documents are initially partitioned into clusters, and only documents that belong to "relevant" clusters are retrieved. As in our approach, Sharma does not address nor is dependent on any particular clustering technique. Sharma does require, however, that the cluster scheme employed yield non-hierarchical clusters, whereas we do not impose such a restriction. Thus, all the clustering schemes, including the numerous schemes described in Willett [613] can also be employed in our setting. Sharma partitions the clusters across the individual nodes according to an architectural topology-independent, best-fit heuristic. No evaluation of the document distribution algorithm is provided.

We also rely on clustering, but use a genetic algorithm approach that uses information about the underlying architecture to map the documents onto the nodes. As the actual dataset used in the [Sha89] evaluation is not described, it is not possible to directly compare the results of our algorithm to the algorithm described in [513]. For tutorials on clustering and other information retrieval related topics, the reader is referred to [78, 482, 613].

15.4 A Genetic Algorithm for MDAP

As MDAP is NP-Complete, obtaining an optimal allocation of documents onto the nodes is not computationally feasible. The heuristic algorithm proposed here is based on Genetic Algorithms [224]. In our representation, the set of documents is represented by a *document vector* which is a sequence of integers 0 to d-1. A permutation of this

sequence defines an allocation of the documents onto the nodes where a document D_i found at position j is stored at node j modulus n. This representation scheme results in all nodes containing an equal number of documents, with the possible difference of a single document. Each allocation is evaluated as the sum of the cluster diameters it defines. The lower the sum, the better is the allocation.

As with most genetic algorithms, the proposed algorithm comprises of initialization, reproduction, crossover, and mutation. In the initialization phase, a set—called *population*—of random permutations of the document vectors is generated. Each random permutation represents a possible allocation of the documents onto the nodes. By repetitively modifying the permutations, a near optimal allocation is generated. The number of simultaneous permutations (*population size*) p is an experimental parameter that is evaluated in Section 6.

The reproduction phase replaces permutations that represent *poor* mappings with those permutations that are viewed as *good*. Using the sum of the cluster diameters as an objective function, the merit of each permutation is evaluated. A biased roulette wheel favoring the better permutations (allocations) is created. A random sampled value is obtained. Using the biased roulette wheel and the sampled value, a corresponding allocation is determined. Each selection corresponds to the *birth* of a new allocation. The permutation that is replaced by this new birth is deemed as *deceased*. Probabilistically, in this phase the *poor* allocations are *killed*, while additional copies of the *good* allocations are reborn.

The crossover phase represents the *cross-fertilization* of permutations, similar to the composition of genes from both parents in a birth, and consists of a position-wise exchange of values between each randomly paired permutations. Two random numbers are chosen and serve as the bounds for the position-wise exchange. Each document of the first permutation that falls within the determined bounds is swapped with the corresponding document of the second permutation, and likewise the second permutation with the first.

Finally, to lower the probability of convergence of the allocation to local values that are not a global minima, a mutation phase is incorporated into the algorithm. Periodically, with a low probability, a permutation is randomly modified.

ALGORITHM:

Initialization Phase:

1. Create a permutation matrix, $P_{i,j}$ $(0 \leq i \leq p-1, 0 \leq j \leq d-1)$. Every row of P, P_i, $(0 \leq i \leq p-1)$ is a complete permutation of all documents D_j, $(0 \leq j \leq d-1)$. For example, if $p = 3$ and $d = 6$, a possible permutation matrix is P.

$P =$

	0	**1**	**2**	**3**	**4**	**5**
0	1	0	2	5	3	4
1	0	2	4	1	3	5
2	4	5	3	2	1	0

2. Define the document to node mapping function $\mathcal{A}_i : D \rightarrow X$ for any given row of P, P_i, $(0 \leq i \leq p-1)$ as $\mathcal{A}_i(D_k) = j \ mod \ n$, where j is the index in row P_i of document D_k, $(0 \leq k \leq d-1)$. If $n = 3$, row P_0 implies that documents 0 through 5 are mapped to nodes 1, 0, 2, 1, 2, 0, respectively.

Reproduction Phase:

3. Given the mapping function $\mathcal{A}_i$ for a given row P_i, $(0 \leq i \leq p-1)$, determine the cluster diameter, $R_{i,j}$, $(0 \leq j \leq c-1)$ for each cluster association list array entry, C_j. $R_{ij} = Max\{M_{\mathcal{A}_i(D_k),\mathcal{A}_i(D_l)}$, where $0 \leq k,l \leq d-1$ and $D_k, D_l \in C_j\}$. Then, if

$$M = \begin{array}{c|ccc} & \mathbf{0} & \mathbf{1} & \mathbf{2} \\ \hline \mathbf{0} & 0 & 2 & 4 \\ \mathbf{1} & 2 & 0 & 1 \\ \mathbf{2} & 4 & 1 & 0 \end{array} \qquad C = \begin{array}{c|cccc} \mathbf{0} & 1 & 3 & 4 & 5 \\ \mathbf{1} & 0 & 2 & & \end{array}$$

then R is

$$R = \begin{array}{c|cc} & \mathbf{0} & \mathbf{1} \\ \hline \mathbf{0} & 4 & 1 \\ \mathbf{1} & 4 & 2 \\ \mathbf{2} & 4 & 4 \end{array}$$

4. Define an evaluation function, E. This function measures the "goodness" of the allocation defined by a row P_i, $(0 \leq i \leq p-1)$, and the corresponding mapping function $\mathcal{A}_i$. In our case,

$$E(P_i) = \sum_{j=0}^{c-1} R_{ij} \quad 0 \leq i \leq p-1$$

5. Create a biased roulette. Compute the reciprocal of each $E(P_i)$, $(0 \leq i \leq p-1)$. Call them $E^{-1}(P_i)$. Bias the roulette proportionally to $E^{-1}(P_i)$. Assign each allocation an interval on the unit vector 0 to 1 based on the corresponding biased probability. In the above example, $E(P_0) = 5$, $E(P_1) = 6$, and $E(P_2) = 8$, resulting in the following roulette wheel.

Thus, permutations P_0, P_1, and P_2, are weighted at a probability of 0.4, 0.34, and 0.26, and are assigned the intervals [0.0, 0.4), [0.4, 0.74), [0.74, 1.0], respectively.

6. Replace the permutation matrix P. Randomly choose p numbers from within the interval [0.0, 1.0]. For each of the p random values obtained, copy the allocation permutation whose assigned interval corresponds to the random value generated into row P_i, $(0 \leq i \leq p-1)$. To insure the survival of successful document allocations (permutations), the lowest cost allocation is always kept. Therefore, if the permutation corresponding to the largest interval, say $P_j, (0 \leq j \leq p-1)$, is not selected within the first $p-1$ selections, P_j is assigned to row P_{p-1}. In the example, if 0.23, 0.92, and 0.36 were the random numbers obtained, then P would be

$P =$

	0	1	2	3	4	5
0	1	0	2	5	3	4
1	4	5	3	2	1	0
2	1	0	2	5	3	4

Crossover Phase:

7. While maintaining a copy of the lowest-cost permutation, say P_t', randomly pair up the rows in P. If p is odd, ignore the unpaired row. For each pair of rows in P, say A and B, randomly generate two integer values, i and j, such that $0 \leq i \leq j \leq d-1$. Position-wise exchange $A_i, A_{i+1}, A_{i+2}, ..., A_{j-1}, A_j$, with $B_i, B_{i+1}, B_{i+2}, ..., B_{j-1}, B_j$, respectively within the two strings. Replace the highest cost permutation with P_t'. The replacement of the resulting highest cost permutation by P_t' guarantees the survival of the "most-fit" parents. For example, $A = P_1$, $B = P_2$, $i = 3$, $j = 4$, mapping string A to string B exchanges the 2 and 5 and the 1 and 3 in row B while mapping string B to string A swaps the 5 and 2 and 3 and 1 in row A. In this example, P_0 is the minimum-cost permutation. The resulting P is

$P =$

	0	1	2	3	4	5
0	1	0	2	5	3	4
1	4	2	1	5	3	0
2	3	0	5	2	1	4

Mutation Phase:

8. Mutate the permutation periodically to *prevent premature loss of important notions* [224]. Randomly choose a number from the interval [0, 1]. If the number falls outside the interval $[1-q, 1]$, where q is the probability of mutation, then terminate the mutation step. Otherwise, select a random number between 1 and r, that designates the number of mutations that occur in the given step. For each of the mutations, select three random integer values i, j, k, such that $0 \leq i \leq p-1, 0 \leq j, k \leq d-1, j-k$, and position-wise exchange $P_{i,j}$ with $P_{i,k}$. Given $q = 0.01$ and $r = 1$, a randomly generated value of 0.006, $i = 0, j = 1$, and $k = 5$, then P would be

$P =$

	0	1	2	3	4	5
0	1	4	2	5	3	0
1	4	2	1	5	3	0
2	3	0	5	2	1	4

Control Structure:

9. Repeat steps 3 through 8. The precise number of iterations is dictated by an early termination condition (all allocations are identical) or by a maximum iteration count. Throughout the experimentation presented here, the maximum limit was set at 1000. In the future, an appropriate limit, possibly a percentage of the population size and/or the number of documents, will be determined experimentally. Upon termination, evaluate the "goodness" of the allocation defined by a row $P_i, (0 \leq i \leq p-1)$, and the corresponding mapping function $\mathcal{A}_i$. Choose the best allocation.

15.5 Theoretical Foundations

As any other heuristic algorithm, the above algorithms is not assured to yield an optimal solution. However, we can still characterize its behavior.

We define the function $b(t)$, where t is a natural number, to be the evaluation of the best string in the t^{th} population. This function is monotonically non-increasing. We say that the algorithm converges to a value l, if

$$\lim_{t \to \infty} b(t)$$

exists and is equal to l.

There have been a few independent efforts to analyze the behavior of the Genetic algorithm. All dealt with the classical Genetic Algorithm operating on binary strings. The most significant of them was conducted by Goldberg [224]. Goldberg has characterized the convergence property of different sets of strings (which he calls *schemata*).

Our algorithm operates on permutations rather than binary strings, and the operators are more complex. We observe that each of the phases has a specific role: The reproduction phase makes the population exponentially converge towards a minimal cluster diameter. The crossover and mutation phases guarantee that a wide search space is investigated. In particular, the crossover phase searches inside some subset ("span" [303]) of the permutation domain, while mutation allows for a search in the space of all possible permutations. The combination of these three results in a very efficient search procedure.

The "Fundamental Theorem of Genetic Algorithms" [224] characterizes the behavior of classical Genetic Algorithms. Towards the development of this theorem, Goldberg defines the schema $H \in \{0, 1, *\}^n$ as a set of binary strings $\omega \in \{0, 1\}^n$ that satisfy

$$H_i = 0 \Rightarrow \omega_i = 0,$$

and

$$H_i = 1 \Rightarrow \omega_i = 1 .$$

For example, the schema $H =$ '$01**11*$' includes both '0111110' and '0101111'. Non-starred elements within H are called fixed positions of the schema.

Goldberg observed the following two determining features for any schema H:

- $o(H)$, the schema *order*: the number of fixed positions in the schema.
- $\delta(H)$, the schema *length*: the difference between the first and last fixed positions in the schema H.

Let $m(H,t)$ be the number of strings in the population that are included in H at time t. Given $m(H,t)$, the Fundamental Theorem of Genetic Algorithms derives the expected value of $m(H,t+1)$. Goldberg shows that the appearance of a schema grows exponentially with its relative fitness, and decreases exponentially with $o(H)$ and $\delta(H)$.

The Fundamental Theorem For the Proposed Algorithm

We generalize the observations by Goldberg into our algorithm. Let the special symbol '*' represent any of the missing values in $(0 \ldots d-1)$; that is, any value not appearing in any of the fixed positions. Let

$r(H)$ be number of consecutive stars starting at the right most location.
(Starting with the left side is equally suitable.)

For example, if H_1 = '01 * *11*', H_2 = '01 * * * **', and H_3 = '01 * * * *1', then $r(H_1) = 1$, $r(H_2) = 5$, and $r(H_3) = 0$.

We consider the effects of the reproduction, crossover, and mutation phases on the schemata implied by the population of permutations. The effects of reproduction precisely mimic the case of binary strings, as described in [224]. Let $f_t(H)$ be the average fitness of the strings representing schema H at time t. If $\bar{f}_t$ is the average fitness over the population at time t, then $\frac{f_t(H)}{\bar{f}_t}$ is the relative fitness of the schema. After a reproduction step, roughly $m(H,t+1) = m(H,t) * (\frac{f_t(H)}{\bar{f}_t})$ copies of schema H exist. If for all time t, $f_t(H) \geq (1+c) * \bar{f}_t$, where c is a constant, then $m(H,t) = m(H,0) * (c+1)^t$. Thus, desirable schemata grow exponentially.

In the crossover phase, two random numbers are chosen as the boundaries. A schema can be destroyed if the boundaries bound some of its fixed values. Therefore, the probability of the survival of a schema H is:

$$(\frac{r(H)}{d})^2 + (\frac{d - r(H) - \delta(H)}{d})^2.$$

This probability results in an increase of the number of copies of short schemata. Rabinovich, Sinclair, and Wigderson [459] have showed that had the Genetic algorithm operated on an infinite population, repetitive steps of one-bit crossover (or mutation) would have caused the population to converge to some distribution, depending on the initial population. A similar assertion seems to hold in the case of permutation-based population.

The mutation phase modifies a schema H that includes the permutation $P_i (0 \leq i \leq p-1)$ if and only if:

1. At least one of the two positions chosen are of the fixed part of the schema; (This occurs with the probability $\frac{2o(H)}{d} - (\frac{o(H)}{d})^2$)
2. The positions do not designate the same processor. (This occurs with a probability of $\frac{n-1}{n}$)

During each mutation phase, i single mutations result, $1 \leq i \leq r$. The probability of surviving a mutation is approximated by (assuming the independence of (1) and (2) above):

$$(1-q) + \frac{q}{r} \sum_{i=1}^{r} (1 - \frac{1}{p}(\frac{n-1}{n})(\frac{2o(H)}{d} - (\frac{o(H)}{d})^2))^i$$

where q is the probability that a mutation occurs. A schema with a small number of fixed positions is more likely to survive.

Ignoring low order terms, we summarize the combined influence of the three operations as:

$$m(H,t+1) \geq m(H,t)(\frac{f(H)}{\bar{f}})(1 - \frac{2(r(H)+\delta(H))}{d} - q\frac{r-1}{r} - \frac{2o(H)q(n-1)}{rpnd})$$

This equation demonstrates that a schema grows exponentially according to its relative fitness in the current population and inversely to the number of fixed positions, the distance between the first and last fixed position within the schema, and the number of consecutive rightmost variable (starred) positions.

Representation Independent Analysis

We continue further and analyze the behavior of a general Genetic Algorithm based heuristic for the MDAP problem for any possible encoding of the allocations. Doing so, we gain a general theoretical understanding of such an algorithm.

Define Q as an equivalent, unique representation of the permutation matrix P as $Q_i[j] = k$ if and only if $P_i[k] = j$. The permutation $Q_i(0 \leq i \leq p-1)$ is an allocation of the documents onto the processors, such that document $D_j(0 \leq j \leq d-1)$ is allocated on processor $X_k(0 \leq k \leq n-1)$ if and only if $(Q_i[j]\ mod\ n) = k$. That is, position j in the permutation represents document j and the corresponding entry modulus n is the processor where the document is stored. For example, if P is:

$P =$

	0	**1**	**2**	**3**	**4**	**5**
0	1	0	2	5	3	4
1	0	2	4	1	3	5
2	4	5	3	2	1	0

then the equivalent Q is:

$Q =$

	0	**1**	**2**	**3**	**4**	**5**
0	1	0	2	4	5	3
1	0	3	1	4	2	5
2	5	4	3	2	0	1

Replacing each value $Q_{ij} \in Q_i$ with $(Q_{ij}\ \textit{modulus}\ n)$ results in a vector of length d with the values 0 to $n-1$. We refer to this vector as the *allocation vector s* (or S_i) of the permutation P_i and to Matrix S as the allocation matrix of P. Given $n = 3$, the equivalent allocation matrix for P is:

$S =$

	0	**1**	**2**	**3**	**4**	**5**
0	1	0	2	1	2	0
1	0	0	1	1	2	2
2	2	1	0	2	0	1

Vector S_i defines an allocation where each document D_j is allocated to processor $S_i[j]$. Every allocation is represented by a unique vector S_i. Notice that the function $AL: P \rightarrow S$ is not one to one. On the contrary, $(\frac{d}{n}!)^n$ different permutations map

to the same allocation. If the population size is p, then the probability that the initialization phase yields two or more permutations that define the same allocation is:

$$1 - \prod_{i=0}^{p-1} \frac{d! - ix}{d!}, \text{ where } x = (\frac{d}{n}!)^n.$$

Define an *s-schema* of an allocation vector s as the vector itself, however, in some of the positions in s, instead of the actual value appearing, a special symbol & is present. This symbol designates any of the valid orderings of the values 0 to $n-1$. For example, given the allocation vectors s_0 and s_1, s_0 = "1 0 2 1 2 0", s_1 = "0 0 1 1 2 2", s-schema H_0 may be "& 0 & 1 2 &". H_0 is an s-schema of both s_0 and s_1. Schema H_1 = "1 0 & 1 2 &", however, is an s-schema of s_0 but not of s_1. Let $\#(i)$ designates the number of the fixed positions in a schema in which the value i appears. Then, either $\lfloor \frac{d}{n} \rfloor - \#(i)$ or $\lceil \frac{d}{n} \rceil - \#(i)$ of the appearances of the special symbol &, designate the value i.

Similarly to the prior computation, we compute $m(H, t+1)$ given $m(H, t)$ for this representation-independent schema. The effects of the reproduction and mutation phases are similar to the case provided above. The main distinction between this representation-independent method and the representation-dependent scheme described above lies in the crossover phase. Here the effects of the crossover phase are exponentially decreasing in $o(H)$, rather than in $\delta(H)$ and $r(H)$. (These measurements are not defined for this case).

We conclude that reproduction converges the population towards the best string that have already been found. Both crossover and mutation diminish the appearance of individual strings (or generally, high order schemata) and intensify that of low order schemata. That is, these phases enlarge the domain of the explored solutions.

15.6 Experimental Evaluation of the Derived Algorithm

To evaluate the described algorithm, a simulation was developed. Given a particular multicomputer architecture (the number of nodes and a cost matrix specifying the internode communication topology) and a set of documents partitioned into clusters, the simulation derived a document allocation using the proposed genetic algorithm. The cost of the derived allocations over a magnitude of architectures and document distributions were used to evaluate the merit of the algorithm.

Various partitioning schemes of the documents into clusters were considered. Sharma [513] stated that d-document collections form from $\sqrt{d}$ to $d/constant$ clusters and assumed such a cluster organization in his evaluation. However, he did not mention what assumptions were made regarding the number of documents per cluster. In this study, we assumed $\sqrt{d}$ clusters and varied the number of documents per cluster. That is, the number of documents per cluster varied from a uniform distribution of documents to clusters to a partitioning in which 25 percent of the clusters contained 50 percent of the documents. The behavior of the proposed algorithm was observed in terms of these varied allocations.

Several multicomputer architectures were considered. These include a 16-node hypercube engine and three mesh configurations (1 by 16, 2 by 8, 4 by 4).

The effects of varying several parameters common to many genetic algorithms were studied. These parameters include the size of the population (number of permutations) and the probability of mutation. Five population sizes ranging from 10 to 50 permutations in increments of ten permutations were examined. The population sizes investigated were kept small to coincide with the size of the database modeled (64 documents). Our intent was to evaluate the algorithm (proof of concept) and not to derive an allocation for an actual database. We intentionally, therefore, selected a small database to better understand and verify our findings.

Figures 6.1 through 6.10 illustrate results for a 64 document database distributed over 16 node systems of varying interconnection topologies. The results for two different document to cluster partitioning are presented. Both document partitions employ 8 clusters but the distribution of documents to clusters is varied. That is, in the first distribution (figures 6.1, 6.3, 6.5, 6.7 and 6.9), all clusters contain an equal (8) number of documents.

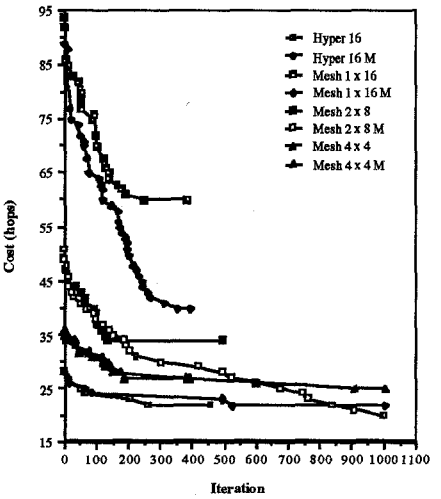


Figure 15.1: All architectures within an even document distribution

In the second distribution, (figures 6.2, 6.4, 6.6, 6.8 and 6.10), 25 percent of the clusters contain 50 percent of the number of documents. For notational convenience, we describe a document partitioning by a four-tuple (D, C, x, y), where D is the number documents, C is the number of clusters, and x and y represent the x percentage of clusters containing y percent of the documents. Therefore, $(64, 8, 25, 50)$ refers to the latter document distribution, while the even document partitioning is represented by $(64, 8, x, x)$, for all values of x, $0 < x \leq 100$.

Figures 6.1 and 6.2 present the results for all the architectures considered. A point on any curve represents an iteration in which a better allocation was derived. As shown, the number of points varies with the architecture considered. All runs terminated at either a point in which the entire population (document allocations), in this case 30, were identical or after 1000 iterations (premature termination), which ever came first.

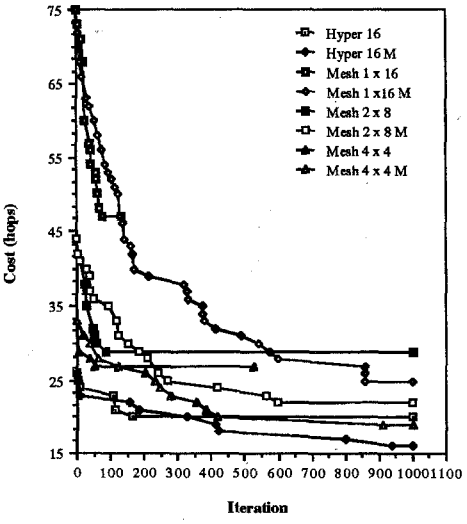


Figure 15.2: All architectures within a $(64, 8, 25, 50)$ document distribution

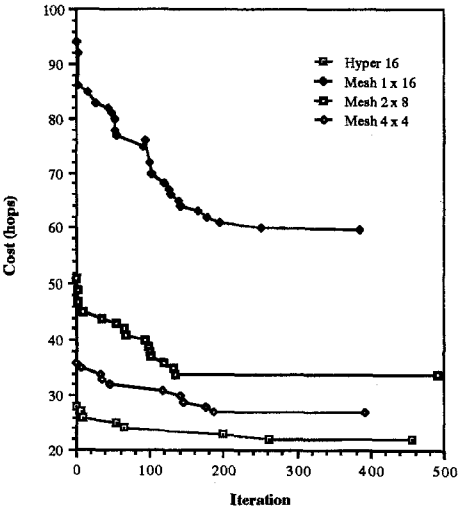


Figure 15.3: Without mutation on an even distribution

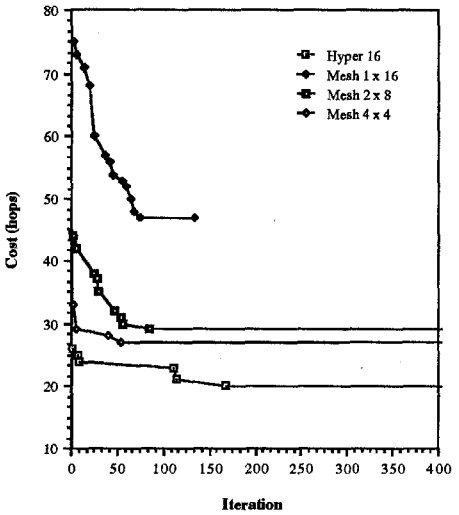


Figure 15.4: Without mutation on a $(64, 8, 25, 50)$ distribution

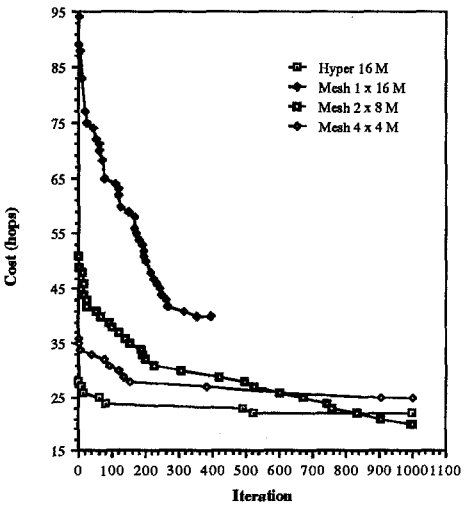


Figure 15.5: With mutation on an even distribution

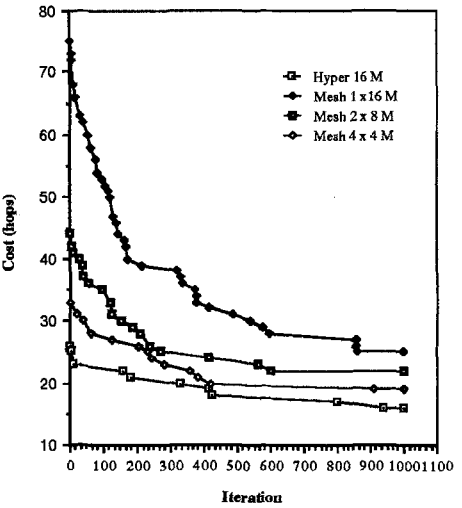


Figure 15.6: With mutation on a $(64, 8, 25, 50)$ distribution

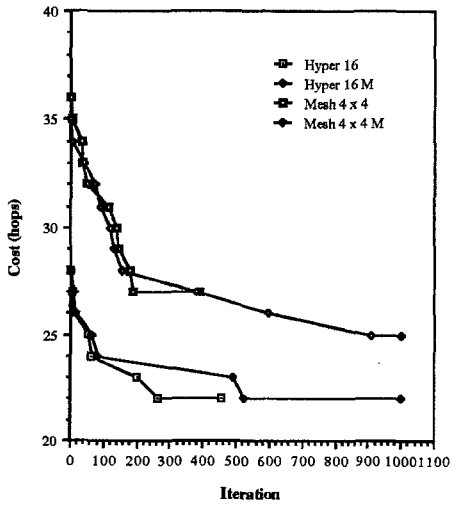


Figure 15.7: Effects of mutation with an even distribution

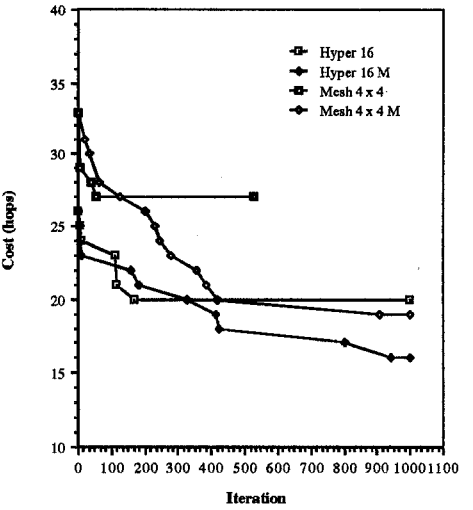


Figure 15.8: Effects of mutation with a $(64, 8, 25, 50)$ distribution

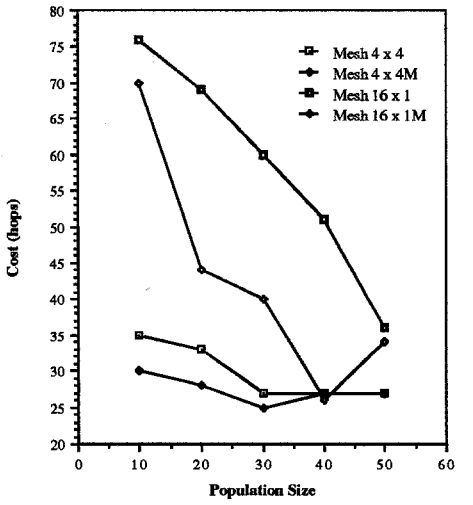


Figure 15.9: Effects of population size using an even distribution

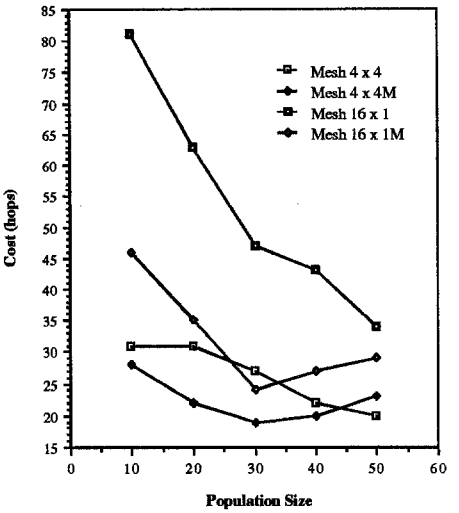


Figure 15.10: Effects of population size using a $(64, 8, 25, 50)$ distribution

A point at 1000 indicates that premature termination occurred. As expected, the higher the communication diameter of the architecture, the greater was the improvement in the derived allocation.

Figures 6.3 and 6.4 and 6.5 and 6.6 more clearly illustrate the behavior of the proposed algorithm in the case where no genetic mutations are possible and when a 0.5 probability of mutation exists, respectively. A 0.5 mutation implies that with a probability of 0.5, a random number of pairs ranging from 1 to 10, will be exchanged. That is, on average, 2.75 pairs will be exchanged per iteration. Figures 6.7 and 6.8 illustrate the effects of mutation on the allocations derived for a hypercube and a 4-by-4 mesh system. As seen, and in all runs performed, mutation results in better allocations. The better allocations result from the prevention of local minima interference.

Finally, figures 6.9 and 6.10 demonstrate the effects of varying the population size from 10 to 50 allocations in increments of 10. When mutations are not possible, the performance consistently improves with the increase in the population size. The improved allocations result from the greater number of possibilities explored during each iteration. In the case where mutation is possible however, initially the performance is improved and then eventually deteriorates. The improvement, as in the case where mutations are not possible, is caused by the increase in the number of possibilities explored. Since the maximum number of mutations per iteration is kept constant throughout, increasing the population size reduces the effects of mutation. Thus, the benefit derived from the mutation phase is diminished. Diminishing the effects of the mutation phase results in a derived allocation that more closely resembles the case in which no mutation is possible. Hence, the performance degrades.

An individual permutation p at a given generation t designates a sample point (p, t) within the total search space. Therefore, given a fixed number of generations (iterations), it is to be expected that the larger the population, and hence the search

	Mesh 4 × 4	*Mesh 4 × 4* (mutation)	*Mesh 16 × 1*	*Mesh 16 × 1* (mutation)
p = 10 (1000)	31	28	81	46
p = 20 (500)	31	22	63	35
p = 30 (333)	27	23	47	36
p = 40 (250)	27	27	47	50
p = 50 (200)	26	29	42	43

Table 15.1: Comparison of population size versus iteration count using a $(64, 8, 25, 50)$ document distribution

space, the better is the derived allocation. A more balanced evaluation compares the effects of varying the population size on the derived allocation if a constant explored search spaced is maintained. That is, an increase in the population size results in a corresponding decrease in the number of generations permitted. Experimental results from such a study are presented in tables 6.1 and 6.2. As seen, a clear preference regarding a larger population versus a longer search is not demonstrated.

	Mesh 4 × 4	*Mesh 4 × 4* (mutation)	*Mesh 16 × 1*	*Mesh 16 × 1* (mutation)
p = 10 (1000)	35	30	76	70
p = 20 (500)	33	28	69	44
p = 30 (333)	27	28	60	41
p = 40 (250)	32	31	62	52
p = 50 (200)	33	30	63	61

Table 15.2: Comparison of population size versus iteration count using a $(64, 8, x, x)$ document distribution

	Genetic Algorithm	*Random Algorithm*	*Greedy Algorithm*
Hypercube	19	23	29
Mesh 16 - by - 1	26	66	67
Mesh 8 - by - 2	22	37	39
Mesh 4 - by - 4	20	28	38

Table 15.3: Comparison of algorithms using a $(64, 8, 25, 50)$ document distribution

Ideally, the cost of the derived mapping should be compared against the cost of an optimal allocation. Since determining an optimal mapping, in the general case, is not computationally feasible (the problem was shown to be NP-Complete in Section

	Genetic Algorithm	*Random Algorithm*	*Greedy Algorithm*
Hypercube	23	25	24
Mesh 16 - by - 1	29	76	56
Mesh 8 - by - 2	23	43	32
Mesh 4 - by - 4	19	33	32

Table 15.4: Comparison of algorithms using a $(64, 8, x, x)$ document distribution

2), two alternative comparison metrics were developed. The first comprised of randomly drawing 100,000 valid (an even distribution across the nodes) permutations and choosing the best allocation among those drawn. The second solution involved the development of a greedy document allocation algorithm based on best fit mappings.

Tables 6.3 and 6.4 compare the derived mappings of the random, greedy, and genetic algorithm solutions. The genetically derived solutions are based on a population of size 30, ($p = 30$), and a 1000 iteration search space. In all cases, for both document distributions, the genetic algorithm approach yielded significantly better mappings than either the random or the greedy algorithms. The relationship between the greedy and random algorithms was not derived as neither algorithm yielded acceptable mappings. (The execution time, in seconds, for all genetic and random algorithms was roughly equivalent. The greedy algorithm took substantially less processing time.)

15.7 Conclusions and Future Directions

The performance of multiprocessor information retrieval systems depend not only on the underlying parallel technology employed but, at least as significantly, on the organization of the data to be retrieved. Poor data allocations result in minimal performance gains on a parallel engine as compared to a uniprocessor system. The problem addressing the derivation of document allocations that support efficient retrieval of documents from a distributed-memory multiprocessor is called MDAP. As an optimal solution to MDAP is not computationally feasible (MDAP is NP-Complete), we proposed a genetic algorithm for MDAP. A proof demonstrating the convergence of our derived mappings to an optimal mapping was provided. Via simulation, the derived document allocations were analyzed. The results obtained compared favorably with both a random and a greedy algorithm.

To complete the described effort, we are presently developing a parallel version of the described genetic algorithm. The parallel algorithm will be able to process significantly larger document sets within an acceptable amount of processing time. As demonstrated in the study by Blair and Maron [65], results obtained using a small document set do not always apply to significantly larger collections. Experimental results using an Intel iPSC/2 will be reported at a later date.

We are also presently developing RIME (Retrieval of Information from a Multicomputer Engine). Once developed, we will evaluate the actual response time difference resulting from the use of various standard document allocation schemes, e.g., round-robin, hashed, etc., and those allocations derived by the parallel version of the described genetic algorithm.

Acknowledgments

We graciously acknowledge the assistance and comments of Kenneth DeJong, Michael Fredman, and William Steiger.

Chapter 16

Amalgame*: A Tool for Creating Interoperating, Persistent, Heterogeneous Components†

Jean-Claude Franchitti‡, Roger King§

16.1 Introduction

Persistent applications are becoming more and more "open" in terms of topology, platform and evolution, thus creating a need for a component-based approach to their development. However, the integration of persistent software components becomes increasingly difficult as we move up through the low-level layers of hardware, operating systems, database schemas and data manipulation languages to the upper-layers of ontologies, application languages and application systems. Most previous approaches to component interoperability have provided support at the representation and specification levels with little or no concern to scalability and applicability of such support to layered persistent systems. The *Amalgame* system, being implemented at University of Colorado at Boulder, addresses higher semantic levels of interoperability, including languages, interfaces, and schemas, with a special focus on language/database interoperability. The Amalgame system is part of the Heterogeneous and persistent APPlications interoperabilitY project (called *L'Heureux*¶, from the acronym "HAPPY"). We shortly introduce the intent, organization and requirements of the larger project. We then focus on the goals and novelty of the Amalgame sub-project.

16.1.1 The Persistent and Heterogeneous Applications Interoperability Project

L'Heureux has as its primary goal the development of a toolkit for supporting incremental integration of diverse systems. It is interesting to notice that often, in modern

*We are using the French spelling of the equivalent English word "amalgam".

†This material is based on work sponsored by the Advanced Research Projects Agency under Grant Numbers MDA972-91-J-1012 and N00014-92-J-1862, and by the Office of Naval Research under Grant Number N00014-92-J-1917. The content of the information does not necessarily reflect the position or the policy of the U.S. Government, and no official endorsement should be inferred.

‡Department of Computer Science, Campus Box 430, University of Colorado, Boulder, CO 80309, U.S.A.

§Department of Computer Science, Campus Box 430, University of Colorado, Boulder, CO 80309, U.S.A.

¶French for "the happy one"; L'Heureux is Roger's real family name.

businesses, interoperability is provided via humans. The "single point of contact" is a person who, usually by telephone, is available to answer questions that involve multiple, heterogeneous systems. As an example, a telephone customer might call up the phone company and ask about a service problem. He or she doesn't know it, but the question might involve three separate systems. Rather than being referred to two other people, the single point of contact is able to view the three systems on three separate screens or windows and provide an integrated answer. The need for a software toolkit supporting integration of existing heterogeneous applications is clear from this example.

The L'Heureux toolkit currently consists of inter-related components that address a variety of semantic levels. The three that we have isolated for our experiments are: database schemas, database transaction protocols, and application programs. Clearly, higher levels of semantic richness result in tougher interoperability problems; for this reason, it is critical that the toolkit support incremental integration, so that heterogeneous systems can be gradually integrated, typically starting at lower levels of semantics. The various components are accessible through a unified interface language as illustrated in Figure 16.1 below.

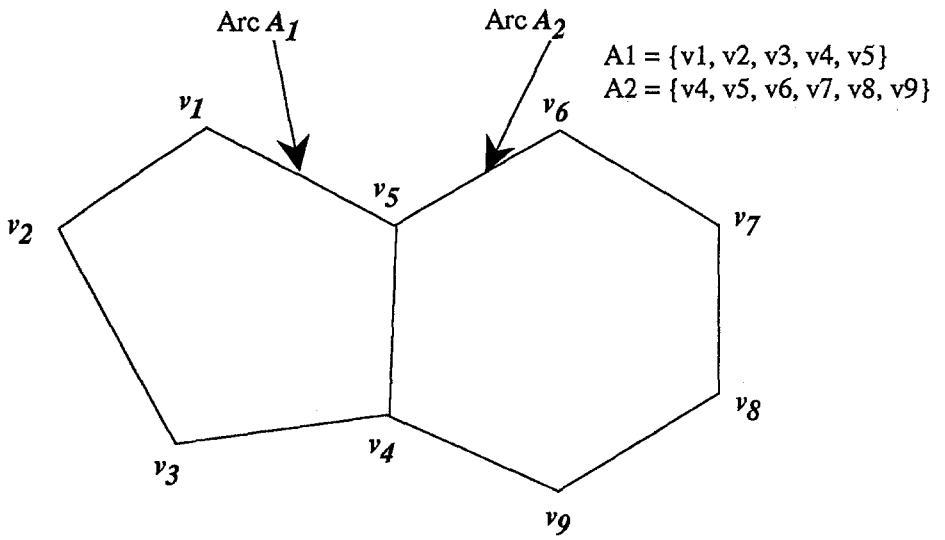


Figure 16.1: The various levels of interoperability encompassed by L Heureux

In this integrated architecture, *Findit* [76] documents, organizes, and helps users locate underlying application components on wide area networks, and *Pano-rama* [644, 645] provides reusable abstractions of application queries. These two components act at the level between database schemas and application programs by managing high level application descriptions and database queries. *A la carte* [161, 160, 162] acts at the level of database transactions by supporting heterogeneous transaction protocols. The *Heraclitus* [220, 265, 267, 286] active database manager provides a rule based repository for keeping track of component interconnections; this system is yet to be

fully integrated into L'Heureux. Amalgame, which focuses on application/database and application/application interoperability, can be used along with the other unified tools to build heterogeneous persistent applications that are truly "open" in the sense that they support interoperability at various levels and granularities of semantics.

The interoperability toolkit does not support any specific methodology for performing integration. For example, many existing "legacy" systems [81] are constructed in a "stove pipe" fashion, with separate application programs running on top of separate database transaction protocols and on top of separate database systems.

Thus, there might be a system written in C running on top of a relational system, and there might be a system written in Ada running on top of a home-made persistent store. It may be desirable to strip away all but the functional core of these systems and throw away the user interfaces and the database systems, and then integrate them with a centralized database system and a uniform user interface. Or, it might be desirable to gradually remove one database at a time and create a uniform database, while leaving the existing applications and user interfaces alone. Clearly, many methodologies can be imagined. Our toolkit leaves these decisions up to the user, and only provides the software modules necessary to integrate systems at a variety of levels of semantic richness.

Our toolkit does enforce, however, a couple guiding principles of interoperability. First, we assume that only rarely will complete, global, integrated database schemas be required, since such schemas are extremely difficult to maintain. Thus, our toolkit focuses on transaction and application program interoperability; we have not yet constructed schema-level components. (Panorama and Findit act closer to the transaction/application level and do not support schema integration.) Second, we assume that often, only pieces of legacy systems are to be integrated. Thus, our toolkit takes an encapsulation approach; our goal is to support the isolation of arbitrary code fragments, and not on building large scale, uniform system interfaces.

Our research[‖] is intended to serve as a vehicle for integrating technologies between the ARPA Arcadia software environment project [575, 576], the Prototech prototyping project [602], and the TI Persistent Object Base project (TI POB) [606]. *Arcadia* and *Prototech* are collaborative research programs encompassing groups at several universities and industrial organizations. The objective of Arcadia is to develop advanced software environment technology and to demonstrate it through prototype environments. On the other end, Prototech is investigating languages and infrastructure requirements for prototyping environments, and consists of several loosely-connected teams. TI POB research program focuses on the design and implementation of an open, next-generation object-oriented database system.

Due to the wide variety of inter-related persistent application components they produce, the Arcadia, Prototech and TI POB research programs provide an excellent framework for studying interoperability. In particular, three important goals of next generation software development environments, such as those envisioned by Arcadia, are broad scope, extensibility and integration. These goals require that Arcadia environments facilitate the addition, modification and replacement of any and all kinds of environment components. To help meet these goals, L'Heureux provides a set of integrated tools that support interoperability of highly heterogeneous components at various levels of semantics.

[‖]The development of L'Heureux is being funded by ARPA, ONR and USWest. In particular, research related to the design and implementation of Amalgame at University of Colorado (CU) is being funded under the ARPA CU-Arcadia and CU-Prototech efforts.

16.1.2 Goals and Novelty of the Amalgame System

The high level goal of the Amalgame system is to develop an integrated solution to address high semantic levels of interoperability, including languages, interfaces, and schemas, with a special focus on language/database interoperability.

In a nutshell, Amalgame provides a high-level specification language to help heterogeneous application designers encapsulate representations of arbitrary code fragments into classes of an object-oriented database framework. Designers can encapsulate entire applications or isolate components within monolithic applications. They can also specify mappings between the original application components and their encapsulated representation. The Amalgame encapsulated components may then be combined in various ways to create new heterogeneous applications. Upon request, Amalgame will validate a specified combination of components and generate a corresponding heterogeneous run-time program.

The Amalgame system provides a novel solution to the application interoperability problem. Indeed, persistent components produced in Amalgame to implement specific heterogeneous applications can be reused to support new designs. Since Amalgame designers incrementally unify underlying types in the context of particular applications, they are not required to conform to a universal type system. Since application subcomponents are abstracted and pieced together as needed, Amalgame ensures efficient "piecemeal" interoperability. Also, the integration mechanisms provided by Amalgame are applicable to a wide range of persistent applications interoperability requirements. Finally, Amalgame provides integrated support for accessing functionality implemented by other components of the L'Heureux toolkit.

In this chapter, we present the Amalgame approach and describe our prototype implementation. We first give a short presentation of current research work related to Amalgame. We then give a detailed description of the Amalgame architecture, present our initial prototype, and give a report of a working interoperability experiment involving the use of Amalgame to interoperate various Arcadia tools. To conclude this chapter, we summarize the experience acquired using Amalgame and outline our plans for extending both the approach and its realization.

16.2 Related Work

Several relatively disparate research areas are relevant to the Amalgame project. This related work is categorized below according to specific characteristics of the Amalgame framework.

16.2.1 Interoperability Support

A significant amount of research has been produced in the domain of application interoperability, an area of primary interest to Amalgame. Such interoperability must address two aspects of cooperation between programs. First, it must coordinate the execution of interoperating programs that may be based on very different execution models, including procedural, triggers or logic programming models. Second, it must achieve type correspondence at some level of granularity so that entities such as data objects or procedures, used in one application can be shared by another application that may be written in a different language or running on a different kind of processor.

Most current approaches to interoperability do not address the execution model interoperability issues. Instead, they rely on the assumption that the interoperating components share some notion of procedure or function call, or some concurrent communication mechanism [52, 221, 404]. Amalgame departs from these approaches by supporting the execution of interoperating programs through an extensible set of coordination mechanisms.

Many research projects have focused on the type model interoperability issues. These issues have been solved in most approaches by establishing correspondence of data types at the representation level (RLI) [19, 244, 366, 393, 392, 403]. The RLI approach provides a lower level support to interoperability than does Amalgame. Indeed, RLI is primarily focused on providing mappings between the different representations of low-level simple types or compound simple types. The lack of support for abstract types makes RLI awkward to use in conjunction with the rich and extensible typing mechanisms inherent to modern programming languages.

Specification level interoperability (SLI), which is much more closely related to Amalgame, overcomes the shortcomings associated to RLI. Rather than focusing on the mapping between different representations of a type, SLI focuses on support for common definitions of a type's properties. Of course SLI depends upon RLI mechanisms, essentially subsuming RLI in those cases involving simple types. Various approaches to achieving SLI are discussed in the literature [64, 115, 611]. A later approach proposed in [612] is most closely related to Amalgame. Amalgame shares this SLI approach by defining a unifying model that can be mapped into specific implementations in their respective domains. However, the Amalgame framework extends SLI mechanisms to the unification of classes in an integrated object-oriented database framework, thereby making it possible to capture higher semantic levels.

16.2.2 Architectural Representation Languages and Megaprogramming

Several research projects have defined architectural representation languages to specify arbitrary software components and support Megaprogramming activities. This work overlaps the domain of the Amalgame framework.

There is a large amount of literature related to specification languages that focus on various issues of module interoperability [44, 382, 425, 578, 579, 620, 621]. However, these approaches are usually restricted to a narrow application domain. More recently, Prototech [269, 322] has focused on defining general purpose prototyping languages which provide a way of expressing architectural and interface concerns in a highly parametrizable manner. To meet these requirements, various Prototech teams have implemented *Module Interconnection Formalisms* (MIFs) [30], such as *Polylith* [458, 457], and *Architectural Description Languages* (ADLs), such as *Griffin* [433, 534, 533].

The Amalgame framework shares some common goals with Prototech. In particular, the aspect of defining a specification language to express and relate application components. However, Amalgame does not intend to produce a language to address specific application prototyping needs. The Amalgame system is more concerned with making layered persistent application interoperable. As a result, Amalgame has focused on providing an extensible and well-integrated service layer infrastructure tailored to the support of persistent applications interoperability.

16.2.3 Extensible Reusable Heterogeneous Frameworks

A lot of related research on heterogeneous and extensible systems has influenced the design of Amalgame [2, 32, 75, 82, 86, 114, 321, 430, 495, 643]. There have been several approaches to provide frameworks for tool fragments interoperability [242, 246] which provide little or no support for extensibility. However, there are several more general purpose approaches which, like the Amalgame framework, use some form of meta-information as a basis of analysis or integration. Among these, the *ARPA TI Open Object-Oriented Database* (TI OOODB) [606], and the A la carte framework [161, 160, 162] are most closely related to Amalgame.

The TI OOODB introduces a computational model of extended events which constitutes a formal conceptual framework to operationally characterize openness and achieve seamless extensibility. The Amalgame internal architecture implements a similar model to transparently extend the behavior of normal operations on its component classes. Amalgame also intends to use an operational version of the TI OOODB as its own underlying component database. At the difference of the TI OOODB, Amalgame primary focus is to support tailorable persistent heterogeneous systems rather than autonomous database systems. To reach this divergent goal, Amalgame provides a unified language to manage heterogeneous components, allowing multiple persistent applications accessing various underlying POBs to interoperate, share and potentially complement their capabilities.

The A la carte system is a toolkit for the rapid construction of heterogeneous, persistent object stores. This system concentrates on providing extensibility and tailorability for internal heterogeneous database management system software. Amalgame uses some of the design concepts of A la carte to implement its software component class management capabilities. At the difference of A la carte, Amalgame provides a persistent heterogeneous application design environment not forcibly limited to the design of heterogeneous object stores. Amalgame also provides a natural extension to A la carte by supplying support for evolving architecture specifications through a unified specification language.

16.3 An Overview of Amalgame

The Amalgame system provides a toolkit designed to assist the construction of heterogeneous and persistent programs. The various components of the Amalgame system are fully extensible; they allow designers to interoperate arbitrary code fragments at selected granularities of semantics, and produce reasonably efficient heterogeneous applications. Using concrete examples, we give a high-level overview of the Amalgame architecture and show how it is used to implement heterogeneous and persistent programs. Our architectural overview describes Amalgame from the designer's point of view, the specification language which guides the designer through the construction process, and an architectural view of the Amalgame framework itself.

16.3.1 A Motivating Example

To experiment with Amalgame, we implemented a demonstration scenario involving two persistent software development support systems used in the Arcadia consortium. A detailed report of this experiment is given in section 16.4 of this chapter. The two systems involved in the experiment are *Rebus* [302] and *PGraphite* [114, 611].

Rebus is a requirement specification process program written in a Software Process Programming Language (SPPL) called *APPL/A* [301, 547]. SPPLs support software-process programming, a way to formalize the representation of software product and processes. APPL/A is an SPPL which specifically addresses the problems related to change management in process centered environments. APPL/A is an extension to Ada which provides abstract and persistent relations to represent inter-object relationships explicitly, and maintain derivation dependencies automatically. APPL/A uses *Triton* [247] to store the relations existing between the different phases of a software process. Triton is an object management system built on top of the *Exodus* [87] database system. The current implementation of Rebus consists of a menu based DAG editor client and a Rebus server. The Rebus server maintains underlying APPL/A relations that are managed by Triton and stored in Exodus.

PGraphite is a meta-tool which automatically generates persistent Ada implementations of graph abstract data types from concise declarative specifications supplied in a Graph Description Language (GDL). These generated Ada implementations access the *Mneme* Database System [418] through a *Storage Manager Interface* (SMI) to store and retrieve graph node instances.

To illustrate our presentation of the Amalgame architecture, we will use concrete examples derived from our experience of using Amalgame to interoperate Rebus and a Rebus-like PGraphite application.

16.3.2 The Designer's View of the Amalgame Toolkit

The Amalgame toolkit is built on top of the *Eiffel* environment [279] and relies extensively on the tools and class management facilities provided by this system. The Amalgame toolkit consists of two main components that help automate the design of heterogeneous and persistent applications. These components include the Amalgame framework which provides a reusable and extensible architecture to integrate persistent and heterogeneous applications, and the Amalgame toolset which provides design components to operate on the framework. The toolset includes a language processor complemented with various tools implementing basic and extended functionality.

The toolset language processor performs lexical analysis and parsing of Amalgame scripts. Amalgame scripts are source texts specified by designers and written using the Amalgame Specification Language (ASL). The ASL is an extension of the Eiffel [298, 401, 402] object-oriented language. The design of the ASL extensions to Eiffel was strongly influenced and guided by the textual version of the Better Object Notation (BON) [431, 432].

The ASL is a unified and extensible language which provides constructs to implement the full range of capabilities offered by Amalgame. Some of these ASL constructs are used to define, refine, test, and assemble framework components. Other ASL constructs implement access methods to the Amalgame toolset basic and extended functionality.

The basic functionality supported by the Amalgame toolset consists of a browser, a run-time generator, and a heterogeneous program execution driver. The extended functionality provides a channel to communicate with other components of the L'Heureux toolkit such as Findit, Panorama, and A la carte. Figure 16.2 below illustrates the various application design components supplied by Amalgame.

The Amalgame browser allows designers to peruse, edit and query existing framework components. Amalgame also provides a run-time generator to package the various framework components and support infrastructure involved in specified heterogeneous

programs. Finally, Amalgame provides an execution driver which can be used as an option to spawn, control and terminate the execution of heterogeneous programs.

Access to the toolset functionality is serverized to promote cooperative heterogeneous application development, and ease the integration of Amalgame in existing development environments. An Amalgame toolset server can handle concurrent requests from multiple client interfaces. Each client interface provides access to the basic and extended Amalgame toolset functionality. Amalgame supports a combination of menu driven and programmatic client interfaces.

Menu driven client interfaces provide interactive implementations of the basic and extended set of Amalgame functionalities. These interactive interfaces are intended to support human designers in the process of defining new heterogeneous and persistent applications. Programmatic interface bindings are currently provided for C and Ada. These bindings implement ASL interfaces for the supported languages and are intended to support direct access to the ASL language from existing applications. Heterogeneous programs specified using the ASL can be built and executed directly through Amalgame client interfaces. Experienced designers may also elect to build and/or execute their Amalgame heterogeneous programs manually.

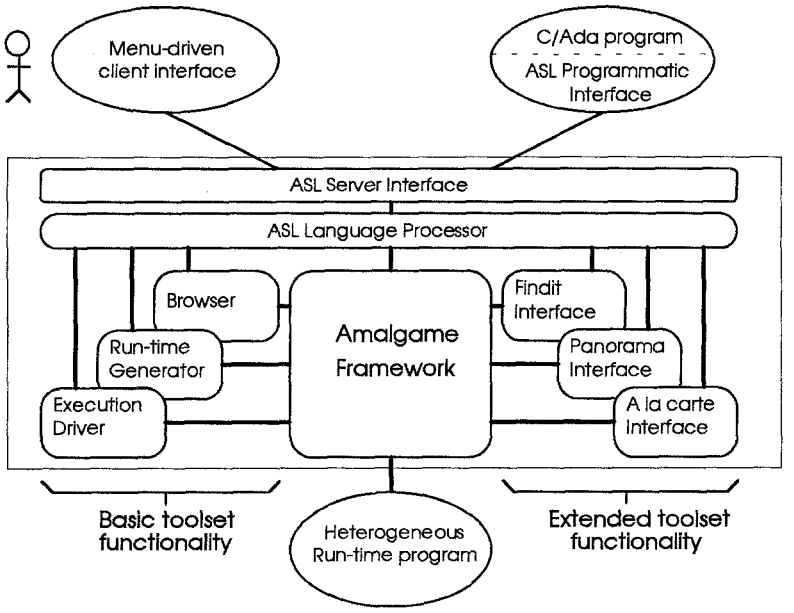


Figure 16.2: A designer's view of the Amalgame design components

16.3.3 Practical Use of the Amalgame Toolkit

To describe the construction of a heterogeneous application using the Amalgame toolkit, we step through the various phases of the Rebus and PGraphite interconnection process mentioned earlier. We assume that a fictitious human designer interacts

with a running Amalgame server through a menu driven client interface. For reasons described in section 16.4 of this chapter, our fictitious designer would like to inter-operate Rebus with a PGraphite Rebus-like application. As a result, he is wondering whether Amalgame has been previously exposed to similar applications and can supply any reusable components.

To answer this question, our designer selects the Amalgame browse menu option to invoke the Eiffel graphical browser and starts exploring the existing Amalgame clusters. In Eiffel terminology, a cluster is a group of related component classes. In Amalgame, any defined application is attached to an environment which itself corresponds to an Eiffel cluster. To simplify our presentation we assume that the cluster list displayed by the browser does not show any evidence of applications related to Rebus or PGraphite. At this stage, our designer can select existing clusters one by one and browse through them. Another alternative is to select the language processor option and submit a small ASL script to query the Amalgame framework using the "Find" construct. Figure 16.3 below shows an example of such a script devised to identify any Amalgame environment containing components related to the keywords "Rebus" or "Requirement Specification".

```
Find    ENVIRONMENT
With    Rebus, Requirement Specification
```

Figure 16.3: A simple ASL script to locate component classes

The ASL "Find" construct is implemented through Findit which is part of the Amalgame extended toolset functionality provided by the L'Heureux toolkit. Other ASL constructs are supplied to assist designers in locating related Amalgame framework class components. In particular, the ASL "Browse" and "Show" constructs provide an alternative to using the graphical browser.

At this point our designer is in the worst case situation where Amalgame cannot provide reusable framework component classes to build upon. He is now ready to start from scratch and step through the various phases of the Amalgame heterogeneous application construction process. These various phases include the encapsulation of selected Rebus and Pgraphite functionality into framework component classes of specified Amalgame environments, the creation and refinement of the infrastructure required to relate these encapsulated components to the selected application functionality, the interconnection of the encapsulated components, and the generation of a run-time executable heterogeneous application.

To start with, our designer creates a new Amalgame environment for these applications. The ASL constructs provided to assist the designer in this process are illustrated in Figure 16.4 below.

"A_APPL/A" and "A_PGRAPHITE" are the names of the Amalgame environments selected to respectively contain Rebus and PGraphite Rebus-like components. This naming scheme provides a good basis for clustering as it suggests grouping applications written in APPL/A in the A_APPLA clusters and applications generated using PGraphite in the A_PGRAPHITE cluster.

As they attach a new environment to Amalgame, designers may be aware of a subset of rules for interchanging basic and/or complex types between the type system of the

```
Attach Environment   APPL/A
With                 NONE
To                   A_APPL/A

Attach Environment   PGRAPHITE
With                 NONE
To                   A_PGRAPHITE
```

Figure 16.4: Creating Amalgame environments for Rebus and PGraphite

environment they are attaching and the Eiffel type system. Amalgame data dictionary files provide a specific syntax for expressing these rules and communicate them to the Amalgame system. In particular, the "With" field, in the "Attach Environment" ASL construct, is used to specify an optional Amalgame data dictionary file name. However, by selecting "NONE" in the "With" fields, our designer chose to implement the type conversions as part of the component specification.

Our designer is now ready to encapsulate selected functionality from the Rebus and PGraphite Rebus-like applications into framework component classes of the respective "A_APPL/A" and "A_PGRAPHITE" Amalgame environments. As explained in section 16.4, his goal is to store persistent Rebus objects interchangeably using Triton or PGraphite. He is therefore trying to isolate the Rebus functionality that manipulates these persistent components. Careful examination of the Rebus source code reveals that the nine underlying APPL/A relations which compose the Rebus process program are mapped into a single Triton relation which provides a better compromise for encapsulation. Figure 16.5 above illustrates the portion of the ASL script which encapsulates the Triton relation.

Now that appropriate Amalgame class components have been created, our designer is ready to use the Amalgame extension library to help specify the necessary infrastructure to relate his class components to the original Rebus and PGraphite Rebus-like applications. Amalgame provides library clusters containing various extension components, which are generic implementations of a specific type of support mechanism. Amalgame currently provides data dictionary, communication and translation library clusters. The set of support mechanisms can easily be extended by adding new clusters and populating them with extension components implementing the adequate support mechanisms. Extension components package functionality from existing Arcadia tools such as Q [392], Triton [247], Chiron [314], and the client/server based generic interconnection scheme supplied by A la carte.

At this point our designer chooses to implement a client/server infrastructure. While browsing through the Amalgame communication library, he notices that the Q-based communication extension components. "aserver.e" and "aclient.e" provide the functionality required to solve his problem. These two components also require the translation extension "adr.e" and the data dictionary extension "add.e". Figure 16.6 above illustrates the partial contents of the "aserver.e" extension component selected to support the Rebus communication server mechanism.

To make the selected library generic support mechanisms available to his framework component environments, our designer uses the ASL "Attach Extension" construct as

```
Encapsulate From    APPL/A.triton.triton_relation
Type Is             SUB_COMPONENT
In                  A_APPL/A.TRITON
As                  RELATION
With Specification {
class       TRITON_RELATION export
            insert, delete, update
feature
Create(name:  STRING);
            do
            (...)
            end; -- Create
insert(relation:  STRING, name:  NAME_STRING) is
            deferred
            end; -- insert
delete(relation:  STRING, instance:  NODE_NAME_TUPLE) is
            deferred
            end; -- delete
update(relation:  STRING,
            instance:  NODE_NAME_TUPLE,
            update_name:  BOOLEAN is false,
            name:  NAME_STRING is ''noname'') is
            deferred
            end; -- update
end -- class TRITON_RELATION
```

Figure 16.5: Encapsulation of Rebus selected functionality

illustrated in Figure 16.7 below.

To complete the implementation of his Rebus framework component infrastructure mechanisms using the generic client/server mechanisms provided by the Amalgame library, our designer creates extension mapping components to handle messages which are specific to his applications. On the Rebus side, his server mapping component inherits the generic functionality provided by the extension component "qserver.e" and provides an implementation for the deferred "execute" method of that extension component. The implementation uses the data dictionary and data translation mechanisms provided by the "adr.e" and "add.e" extension components mentioned earlier. A partial snapshot of the ASL script implementing this phase is given in figure 16.8 below.

Our designer provides similar ASL script specifications to accommodate his PGraphite Rebus-like component encapsulation. The ASL can also be used to define new extension components or check whether a framework component is an extension component. This functionality is available through the ASL constructs "New Extension Component" and "Is Extension Component".

Our designer is now ready to interconnect his Rebus and PGraphite Rebus-like framework components and implement the internal features of his serverized Re-

```
class   ASERVER export
        initialize, execute, shutdown
feature
        initialize is
            do (...)
            end; -- initialize
        begin is deferred
            end; -- begin
        commit is deferred
            end; -- commit
        abort is deferred
            end; -- abort
        execute(func:  STRING,
            input:  ADR_HANDLE,
            return:  ADR_HANDLE)is
            deferred
            end; -- execute
        shutdown is
            do (...)
            end; -- shutdown
end -- class ASERVER
```

Figure 16.6: Amalgame class which supplies server mechanisms

bus framework component. To properly support the demonstration scenario described in section 16.4 of this chapter, the serverized Rebus framework component should properly react to requests of the type "Execute(APPL/A.TRITON.-RELATION.nodename.insert, input, ...)", where the "input" buffer supplies the parameters "(tuple, PGRAPHITE)". Such a request should insert the specified "tuple" in the "nodename" relation of APPL/A.TRITON.RELATION in the context of the "PGRAPHITE" Rebus-like application.

The context based mechanism just described allows designers to provide different internal implementations of their encapsulated components according to the context in which they are being accessed. As a result, our designer requests the interconnection and provides the necessary implementations using the ASL constructs partially illustrated in Figure 16.9 below.

To emphasize the result of this context-based interconnection process, it is important to point out the differences between the Rebus encapsulated component "A_APPL/A.TRITON.RELATION" and the original Ada package "APPL/A.triton.-triton_relation". Although both have the same interface in terms of methods and attribute declarations, the internal implementations of these methods and attributes are completely different. For example, the "insert" method of "A_APPLA.TRITON.-RELATION" in the context of PGRAPHITE shown in Figure 16.9 uses the "Put-Attribute" method provided by the encapsulated component "PGRAPHITE.REBUS" to store persistent objects in PGraphite.

In a nutshell, the interconnection process we just described specifies a list of com-

```
Attach Extension To A_APPL/A
Of Type             DATA_DICTIONARY
Specified In        ./library/dictionary/add.e

Attach Extension To A_APPL/A
Of Type             DATA_COMMUNICATION
Specified In        ./library/communication/aserver.e

Attach Extension To A_APPL/A
Of Type             DATA_TRANSLATION
Specified In        ./library/translation/adr.e
```

Figure 16.7: Attaching component extensions to the A_APPL/A environment

ponents to interoperate and supplies modifiers that implement appropriate support features. Through this process, Amalgame environments can dynamically capture meta-model information that describes the requirements and conditions under which specified components can be interconnected. This interconnection process is supported by Amalgame context-sensitive modifiers and a heterogeneous constraint mechanism [90], which is part of the extended functionality provided by the L'Heureux toolkit.

At this stage, our designer has successfully created the Rebus and PGraphite Rebus-like framework class components and derived specializations of these components using Amalgame modifiers. He has also derived and interfaced proper mapping components as specialized instances of generic Amalgame framework extension components. It is now possible to generate a run-time heterogeneous program that will implement the interconnection between the Rebus and Pgraphite Rebus-like encapsulated components.

Our designer can use the ASL construct "Megacompile" to validate framework components, generate corresponding Eiffel classes, and Eiffel compile the resulting classes. He can then use the ASL construct "Megalink" to assemble specified framework components. The ASL construct "Generate Runtime Component", which is also available as an option of the menu driven client interface, combines the "Megacompile" and "Megalink" phases and generates runtime programs by operating directly on a specified root class. Once successfully built, the heterogeneous program can be invoked through the "Execute" menu option or the equivalent ASL construct. Figure 16.10 below shows the script portion that builds and executes the Rebus/Pgraphite heterogeneous program.

16.3.4 The Internal Architecture of the Amalgame Toolkit

The Amalgame internal architecture implements support mechanisms for creating, maintaining, and interconnecting heterogeneous framework components. These mechanisms are handled by various Amalgame underlying components including the persistent object base (POB), the ASL lexical analyzer and parser, the framework configuration manager, the framework component context handler, and additional support facilities for transaction, correctness and constraint management.

The Amalgame POB is implemented using the Eiffel environment support facilities. In Eiffel, an environment represents a set of persistent objects that can be individually

```
Add Class To          A_APPL/A
Of Type               EXTENSION
As                    TRITON_MAPPING
With Specification    {
class TRITON_MAPPING export

        initialize, execute, shutdown
inherit
        ASERVER, ADR, ADD
feature
        -- Execute method
        execute(func:  STRING,
             input:  ADR_HANDLE,
             return:  ADR_HANDLE) is
        local
             adr:  ADR;
             tuple:  TUPLETYPE;
             relation:  TRITON_RELATION
        do
             if func = APPL/A.
             TRITON.RELATION.nodename.insert then
             adr.Create;
             adr.set_read;
             adr.tupletype (input, tupletype); (...)
             relation.Create (``nodename'');
             -- perform ``insert'' on tuple
             relation.insert(tupletype); (...)
             end
        end -- execute
end -- class TRITON_MAPPING
```

Figure 16.8: Partial snapshot of the Rebus extension mapping component

identified through keys. Amalgame maps its heterogeneous application environments onto Eiffel environments, and uses keys to identify and selectively manipulate the various framework components associated with each environment. Amalgame currently uses the Eiffel facilities to store its heterogeneous application environments in the UNIX** file system. In the near future, Amalgame will provide specializations of the Eiffel environment support classes to integrate the TI OOODB and Triton as possible underlying framework object repositories.

The Amalgame lexical analyzer and parser provide the language processing support required to check the syntax and semantics of ASL scripts. This extensible component is implemented using the Eiffel lexical and parsing library classes. The parser uses the lexical analyzer to identify ASL tokens as it scans through the ASL constructs found

**UNIX is a registered trademark of AT&T

```
Interconnect     APPL/A.TRITON.RELATION, PGRAPHITE.REBUS

Modifier For     APPL/A.TRITON.RELATION.insert
In Context Of    PGRAPHITE.REBUS
With Type        DEFINE
With Method      {

        insert(relation:  STRING, name:  namestring) is
        local
                node:  REBUS;
                id:  integer;
        do
                (...)
                if relation = ''nodename'' then
                   id = GetUniqueId(name);
                   node.Create(''RebusNode'');
                   node.AssociateNodeWithId(''RebusNode'');
                   node.PutAttribute(node.id,relation,name);
                   node.PutAttribute(node.id, ''nodeid'', id);
                end
        end -- insert
```

Figure 16.9: Interconnection and specification of a context sensitive feature

in the submitted scripts. Once scripts are successfully parsed, the Amalgame language processor applies the Amalgame semantics, which implement the specified ASL constructs through the Amalgame framework configuration and component context managers.

The Amalgame framework configuration manager maintains a table of configuration connector components to keep track of the inter-relations between the various Amalgame environments. The configuration connectors also keep track of component classes and extensions attached to each particular environment. Finally the configuration connectors associate and maintain the keys associated to the Amalgame framework components in their corresponding environments. As an example, during the encapsulation process described in section 16.3.3, the configuration manager defines two separate configuration connectors for the "A_APPL/A" and "A_PGRAPHITE" environments. A third configuration connector is defined for to the Rebus/PGraphite environment.

The framework component context manager interacts with the framework configuration manager to create new configuration connectors or check the consistency of ASL constructs according to existing configuration information. The context manager also creates context-based mapping connector components which are used to seamlessly adapt the features of encapsulated components through modifiers specified using the ASL. In the case of the example given in section 16.3.3, the Amalgame context manager creates a connector component which inherits the deferred insert, delete and update features of the "A_APPL/A.TRITON.RELATION" (see section 16.3.3, Figure 16.5)

```
--
-- Build and execute run-time heterogeneous program
-- The root class is TRITON_MAPPING
--
Generate Runtime Component    APPL/A.TRITON_MAPPING

Execute                       APPL/A.TRITON.RELATION
```

Figure 16.10: Building and executing the Amalgame heterogeneous program

and defines the insert method using the specification provided in the modifier (as shown in section 16.3.3, Figure 16.9). The functionality of the resulting context-based connector component is illustrated in Figure 16.11 below.

To generate a run-time version of "A_APPL/A.TRITON.RELATION", the context manager combines all the context-based connectors defined for "A_APPL/A.TRITON.-RELATION" into a single object and supplies an access method to allow this generated component to sense the context in which it is accessed at run-time. This explains how the "PGRAPHITE" context parameter is used at run-time by "A_APPL/A.TRITON.-RELATION" to select the insert method shown in Figure 16.11.

This component context sensitivity plays an important role in supporting concurrency control mechanisms and transaction management facilities within heterogeneous applications created by Amalgame. Let us assume that we are implementing an heterogeneous application which is somewhat more complicated than the one we describe in the Arcadia experiment. This application still involves a "A_PGRAPHITE.REBUS" component but this component is now accessible in the context of REBUS as well as PGRAPHITE. We also assume that the implementations of the modifier methods defined for the PGRAPHITE context access "A_PGRAPHITE.REBUS" in the context of REBUS. In this case, accessing "A_PGRAPHITE.REBUS" in the context of PGRAPHITE triggers a begin transaction for this particular component/context combination. When "A_PGRAPHITE.REBUS" is accessed in the context of "REBUS" Amalgame senses the context parameter, finds out that it is different from the current transaction context and triggers a nested begin transaction. This example illustrates the nested transaction model implemented by Amalgame on the basis of defined context parameters.

To implement the nested transaction model just described, the run-time version of "A_APPLA.TRITON.RELATION" is created by Amalgame as a multi-threaded object which supports concurrency control and a nested transaction model to switch between its defined access contexts. As a result, if PGraphite were to use a version of Mneme which supports transaction management and concurrency control, we could create a heterogeneous application on top of Mneme that supports a context-based nested transaction model. A similar approach that combines the definition of active threaded objects in an object-oriented model is described in [82]. As an alternative to the nested transaction model just described, Amalgame also provides ASL constructs to reuse existing transaction management and recovery mechanisms supplied by A la carte through the extended toolset functionality provided by the L'Heureux toolkit.

The context-sensitive mechanism we just described was largely inspired by the TI

```
(...)
export
    insert, delete, update
feature
insert(relation:  STRING, name:  namestring) is
    local
          node:  REBUS;
          id:  integer;
    do
          (...)
          if relation = ''nodename'' then
               id = GetUniqueId(name);
               node.Create(''RebusNode'');
               node.AssociateNodeWithId(''RebusNode'');
               node.PutAttribute(node.id,relation,name);
               node.PutAttribute(node.id, ''nodeid'', id);
          end
    end -- insert
delete(instance:  NODE_NAME_TUPLE) is
    deferred
    end; -- delete
update(instance:  NODE_NAME_TUPLE,
    update_name:  BOOLEAN is false,
    name:  NAME_STRING is ''noname'') is
    deferred
    end; -- update
end
(...)
```

Figure 16.11: Functionality of the Rebus context-based connector component

OOODB computational model of event extensions described in section 16.2.3 of this chapter. The model seamlessly adds database functionality to existing object-oriented programming languages as a collection of event extensions. An event is the application of an operation to a particular set of objects. An extension to a programming language is modeled as enhancements to the behavior of events in the language. Extensions can be modeled in various ways and invariants that must be met by the corresponding operations can be specified. For example, if an extension defines the ability to operate on remote objects, an invariant could state that the operator and the operands need to be instantiated in the same physical address space. Each way of meeting such an invariant is defined in the TI model as a policy which is implemented by a policy performer. We refer the reader to [606] for a complete description of the TI OOODB computational model of event extensions.

To model its run-time context sensitive mechanism, Amalgame defines as an event the operation that consists in applying a set of modifier methods to a framework class component. This event is extended by defining modifier methods using the ASL

"Modifier For" construct. The corresponding Amalgame invariant rules that the set of modifiers associated to a class component in a certain context must match the set of modifiers defined for this context. To meet this invariant, Amalgame currently implements a single policy as an internal policy performer. This performer selectively retrieves the proper set of modifiers which are applicable to a specified framework class component in a certain context.

To complete the description of the Amalgame internal architecture, we briefly describe its additional support facilities for transaction, correctness and constraint management. Amalgame provides transaction management support as part of its generic extension components. Indeed, the begin, commit and abort methods provided in an extension mapping such as "TRITON_MAPPING" can be used to group multiple ASL constructs in a single transaction. Within this transaction, Amalgame automatically implements a nested transaction model using the context parameter as a basis for locking components. An example of a heterogeneous application making use of this nested transaction model was presented earlier in this section.

Amalgame also inherits correctness enforcement mechanisms from the Eiffel language. Semantical correctness ensures that a class implementation conforms to its specification and other requirements. Eiffel provides language constructs which can be checked dynamically to ensure partial correctness of run-time programs. In particular, Eiffel supports the definition of class invariants, method pre/post conditions, and loop variants/invariants. We refer heterogeneous application designers to the Eiffel language documentation [279] for a complete description of correctness mechanisms.

Finally, the ASL supports constraint specification, detection, and enforcement, as part of the extended functionality provided by the L'Heureux toolkit through its heterogeneous constraint package. Using this package through special ASL constructs, designers can ensure correct heterogeneous component integration and maintain consistency in heterogeneous environments.

16.4 An Arcadia Demonstration Scenario

The interoperability experiment we have implemented using Amalgame consists in interoperating Rebus with an Ada Rebus-like application generated by PGraphite from a simple GDL specification. We quickly describe the Arcadia experiment from a high-level standpoint and expose the simple modifications required to interconnect the Rebus and Ada Rebus-like applications to the heterogeneous Amalgame server generated in section 16.3.3. Finally we summarize the various benefits of using Amalgame in the scope of this experiment.

16.4.1 High-level Description of the Arcadia Experiment

There are numerous benefits to implementing mechanisms that transparently store requirement specifications produced by Rebus using Triton and PGraphite interchangeably. For example, such mechanisms would allow a design specification tool written using PGraphite to directly access and modify a Rebus requirement specification for a particular design. This would definitely simplify the specification and maintenance of the inter-dependencies between software requirement and design specifications. Moreover, the above described mechanisms address the interoperability of applications based on heterogeneous graph specification languages. As a result, they provide an interesting testbed for experimenting Amalgame interoperability mechanisms at the levels of

protocols and languages.

These various considerations motivated our experiment which precisely consists in interoperating Rebus, Triton and a PGraphite Rebus-like application to transparently store Rebus requirement specifications using either Triton or PGraphite. The basic scenario using Amalgame to interoperate Rebus with Triton and PGraphite is shown in Figure 16.12 below. On the PGraphite side, a GDL specification is devised to implement the Rebus underlying APPL/A relations in terms of a PGraphite node kind. The corresponding Ada application generated by PGraphite is then made interoperable with Rebus and Triton using Amalgame supplied extension components. Figure 16.12 also illustrates the encapsulated Rebus and PGraphite components required to support this experiment and the access contexts supported by these components.

16.4.2 Implementation Details Related to The Arcadia Interoperability Experiment

A detailed representation of the Arcadia experiment showing the nature of the actual components required to support the demonstration is provided in Figure 16.13 above. We then briefly describe the different components required to support the interconnection of the Rebus and Ada Rebus-like applications to the heterogeneous Amalgame server generated in section 16.3.3.

Modifications and Amalgame Support for Rebus

Some small modifications are inserted in the Rebus DAG editor server at the level of its Triton client interface. These modifications involve initialization and shutdown of the Amalgame heterogeneous server, and execute method calls to invoke the Amalgame encapsulated Rebus application component. Actual modifications added to the Triton relation insert procedure are shown in Figure 16.14 below.

The execute method calls inserted in the Rebus DAG server Triton interface are processed by a generic Ada client interface obtained from the Amalgame communication library. This client interface is specifically designed to communicate with the server extension component used to create the extension mapping "TRITON_MAPPING" in section 16.3.3. The generic Ada client interface obtained from the Amalgame library does not require any modification, it is simply compiled and linked to the Rebus server/client component. Figure 16.15 below shows the Ada specification for the Amalgame supplied client package.

Modifications and Amalgame Support for PGraphite

Modifications and Amalgame support provided for the PGraphite generated Ada application are of similar simplicity as the ones provided for Rebus. Therefore, we omit the details concerning the Amalgame supplied server interface and the associated modifications to the PGraphite application. Figure 16.4.2 below illustrates the GDL script implemented to specify the Rebus relations in terms of a PGraphite node kind.

```
class RebusDAG is
package Rebus;
with     Unix_Time.(Time_T);
subtype  uniqueid is INTEGER;
subtype  timestamp is Unix_Time.Time_T;
```

```
subtype  namestring is STRING;
subtype  accessrights is INTEGER;
subtype  text is STRING;
type coordstate is (Posted, Locked, Completed);
type FieldType is (NAME, AUTHOR, DATE, DESCRIPTION, TYPES,
                   DEFINITIONS,INPUT_SPEC, OUTPUT_SPEC,
                   ACCURACY, ROBUSTNESS, TIMING, FLEXIBILITY,
                   SECURITY, PERFORMANCE);
type FieldSet is array(FieldType) of Boolean;
type FieldValueTuple is record
         typename:  namestring;
         value:  text;
     end record;
type FieldValues is array (FieldType) of text;
node RebusNode;   -- complete definition given below
type NodeSequence is sequence of RebusNode;
node RebusNode is
     nodeid:       uniqueid;          -- nodename
     nodename:     String;
     parents:      NodeSequence;      -- parents/children
     children:     NodeSequence;
     created:      timestamp;         -- status of node
     modified:     timestamp;
     owner:        namestring;
     rights:       accessrights;
     state:        coordstate;        -- state of node
     author:       namestring;
     stamp:        timestamp;         -- trace of node
     to:           coordstate;
     traceauthor:  namestring;        -- trace author
     lockowner:    namestring;        -- lock
     fields:       FieldSet;          -- field set
     fieldvalue:   FieldValues;       -- field values
     end node;
end Rebus end RebusDAG;
```

Figure 0.16 GDL class representing the Rebus DAG

16.4.3 Benefits of The Amalgame Approach

The object-oriented paradigm promotes the reusability of Amalgame components and past modifications to these components. The encapsulation of general purpose application code fragments provides a way to organize existing components by decomposing them into cooperating objects. As a result, Amalgame framework components usually support a basic set of reusable functionality that applies to various contexts without requiring extensive modifications. These modifications are also extremely simplified by the constructs supported by the Amalgame ASL and the underlying Eiffel object-oriented language. Moreover, inheritance provides a flexible mechanism to isolate,

maintain, adapt and reuse past modifications to Amalgame components implemented through ASL modifiers.

There are other benefits to the Amalgame approach which are not directly related to its object-oriented approach. In particular, the implementation of the communication infrastructure supporting our Arcadia experiment is greatly simplified by the Amalgame extension library which provides the necessary support to quickly implement the required client/server interfaces. We also find the amount of modifications required to interoperate the programs to be quite minimal in comparison with the coding of the Amalgame specification script. The Amalgame lexical analyzer and parser are fast and efficient and Amalgame provides useful assistance in the debugging of ASL scripts.

We can foresee that the type of automated assistance provided by Amalgame becomes increasingly useful as more interconnection contexts populate the Amalgame framework. Since the implementation details are hidden from the interoperating programs by the Amalgame framework components, we can quickly experiment alternative modifier implementations without affecting the interoperating programs. Finally, the use of a single unified specification language is a definite benefit. Indeed, in the case of our Arcadia experiment, no extensive knowledge of APPL/A or GDL is required to interoperate the Rebus and PGraphite Rebus-like persistent programs.

16.5 Future Directions

16.5.1 A Joint Arcadia and Prototech Demonstration Scenario

We are currently implementing a joint Arcadia and Prototech demonstration scenario to enable application developers to use variations of Rebus with either Triton or PGraphite as their storage manager. These variations of Rebus are obtained by modifying the Rebus Process Program Control Structures using a MIF [30] prototyping language. For example, a variant of Rebus could capture bottom-up instead of top-down functional requirements. Amalgame provides the support components for this interoperability experiment between Rebus, Triton, PGraphite and the MIF prototyping language. The demonstration accomplishes four major goals. It shows how Prototech languages can interoperate with Arcadia tools, it makes Rebus more usable by making it more flexible, it also validates the usefulness of a Prototech language by using it to quickly prototype a variation of Rebus. This experiment finally illustrates how Amalgame can be used to allow a tool to run on a different database platform, and allow a prototype system to replace an existing version of an outside tool.

16.5.2 An Interoperability Experiment with Chiron

There is a growing interest among the Arcadia researchers at University of California at Irvine to put Chiron on top of multiple database managers. Chiron is a serverized user-interface system capable of supporting multiple simultaneous views of abstract data types (ADTs). Chiron provides an Abstract Depiction Language (ADL), which is object-oriented and used to program new "widgets". Multiple client programs called "artists" can concurrently send requests to the Chiron server to display different views of the Chiron server ADTs. We are currently investigating the requirements of the Chiron Object Manager interface which corresponds to a heterogeneous interface between

Chiron and multiple underlying databases. This will help us evaluate the support provided by Amalgame to place Chiron on top of multiple object managers.

16.5.3 An International "Library" of Deployed "Wrapped" Persistent Applications

Research performed under the L'Heureux and Discovery projects at University of Colorado at Boulder suggests the idea to extend the scope of interoperating persistent applications to develop an international "library" of deployed encapsulated persistent applications. Possible interested sites to support this experiment have been identified in Europe (Esprit, GMD), Japan, Australia (CITRI), and the U.S.A (USC, CU). Although this project is in its very preliminary stages, it constitutes an interesting experiment for Amalgame in terms of providing interoperability mechanisms for large scale persistent systems.

16.6 Conclusion

The Amalgame system provides an integrated solution to the persistent heterogeneous component interoperability problem by addressing various levels of interoperability among languages, interfaces, and schemas. In this chapter, we have thoroughly described the architecture of Amalgame and discussed an interoperability experiment involving Arcadia tools.

The existing Amalgame prototype could be improved in many ways. Constraints between heterogeneous components could be managed in a more flexible fashion. Interfaces to existing or upcoming components of the L'Heureux toolkit should be created or perfected to improve the level of automated assistance. Finally, new components should populate the extension library to further enhance existing communication, translation, data dictionaries, or transaction management schemes, and support the various types of execution models found in diverse application domains.

Amalgame contribution to the "open" database research world is to provide a framework for analyzing the components that should be exposed to ease interoperability among future database systems. Amalgame provides an important enabling technology which applies to a large number of application domains ranging from wide integrated, extensible, broad-scope persistent environments to larger, evolving, heterogeneous persistent software systems.

16.7 Acknowledgments

We appreciate the comments and suggestions provided by Dennis Heimbigner, Stan Sutton, Mark Maybee, Shenxue Zhou, Athman Bouguettaya and Jim Dalrymple who all contributed to the design and implementation of the Amalgame system.

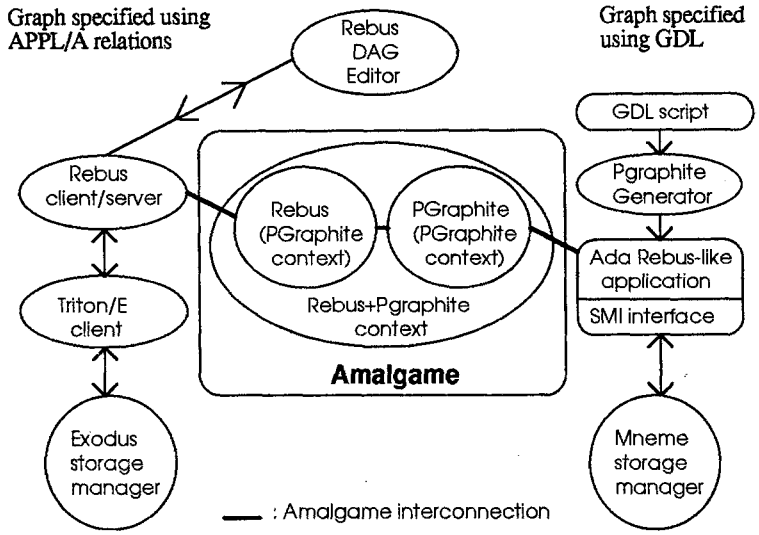


Figure 16.12: High level view of the Amalgame experiment

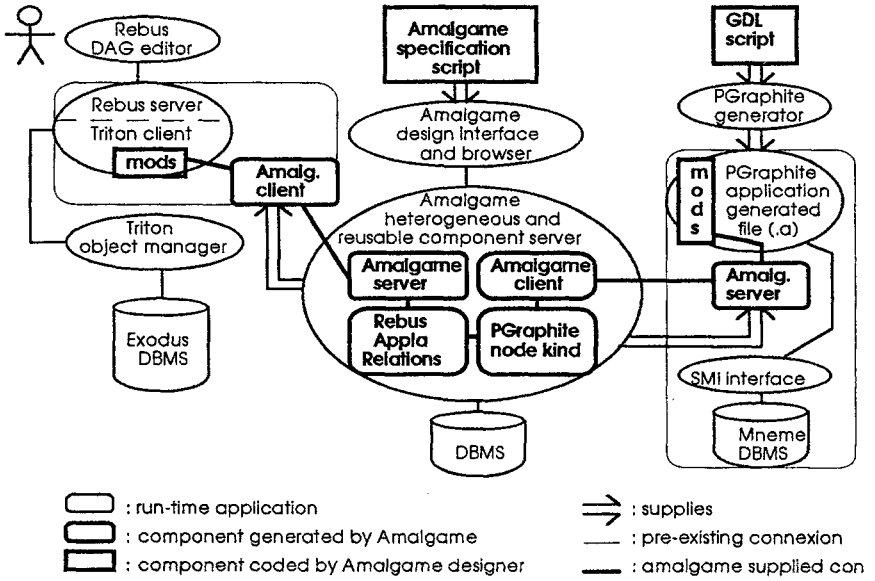


Figure 16.13: Amalgame interoperability experiment using Arcadia components

```
procedure insert(tup:  in out tupletype)
     is
     begin
     (...)
     if A_initialized = true then
          qdr.set_write(A_qargs);
          qdr_tupletype(A_qargs,tup);
          qdr_string_slice (A_qargs,A_context.all);
          status := execute(''APPL/A.Rebus.'' & relation_name &
                              ''.insert'', A_qargs,A_qresult);
          if status <= 0 then raise amalgame_access_error;
          end if;
     end if;
end insert;
```

Figure 16.14: Sample modification of the Rebus insert procedure

```
with system;
with q, qpc, qdr, qpc_clnt;
package amalgame is

     amalgame_access_error:  exception;
     amalgame_constraint_error:  exception;

     function initialize_amalgame (hostname :  string)
          return integer;
     function shutdown_amalgame(exitcode :  in integer:=0)
          return integer;
     function execute (amalgame_component_method :  in string;
          input:  in qdr.handle := qdr.null_handle;
          output:  in qdr.handle := qdr.null_handle)
     (...)
end amalgame
```

Figure 16.15: Rebus Ada client interface supplied by Amalgame

Chapter 17

Correctness and Enforcement of Multidatabase Interdependencies

G. Karabatis*, M. Rusinkiewicz†, A. Sheth‡

17.1 Introduction

Many industrial computing environments consist of multiple data processing systems developed along functional or organizational divisions. Each such system usually automates a part of company operations and consists of an application and a centralized DBMS. While the systems are frequently interconnected, they typically are not integrated and provide a limited support for interoperability. An important problem in such environments is to maintain a desired level of consistency of data across these systems, in the presence of concurrent update operations.

The concept of interdependent data has been introduced in [518] to provide a framework for studying data consistency in multidatabase environments. Interdependent data are data objects related by consistency requirements and possibly managed by different systems. These objects could be quite different structurally and semantically. *Data dependency descriptors* (D^3s) [479] are used to specify the dependency between related data objects, the levels of permitted inconsistency, and methods that can be used to restore the consistency if it is violated beyond the specified limits.

In addition to the specification of data consistency requirements, two additional issues need to be addressed to manage interdependent data: (a) correctness of the specifications and (b) enforcement of the specifications. In this chapter, we address these two issues for a wide range of interdependency specifications. However, given the complexity of handling relaxed constraints consisting of both data state and temporal components in an environment consisting of heterogeneous and autonomous systems, significant additional work will be needed in the future.

An update to an interdependent data object, may violate the consistency requirements among interdependent data objects. The mutual consistency requirements may not require immediate restoration of full consistency. Instead, a promise from the system that the consistency will be restored eventually, may be sufficient. Various relaxed consistency criteria have been proposed in literature. The relaxed criteria of consistency for replica control include k-completeness [489], ϵ-serializability [456, 631]

*Department of Computer Science, University of Houston, Houston, TX 77204-3475, U.S.A., george@cs.uh.edu

†Department of Computer Science, University of Houston, Houston, TX 77204-3475, U.S.A., marek@cs.uh.edu

‡Bellcore, 444 Hoes Lane, Piscataway NJ 08854-4182, U.S.A., amit@ctt.bellcore.com

and N-ignorance [333]. The above three criteria allow limited inconsistency among replicas specified in terms of the number of updates or other countable events. D^3s allow specifying more general criteria of *lagging consistency* and *eventual consistency* to capture relaxed consistency requirements involving a variety of temporal and data state based parameters among interdependent data [519].

When the consistency limits of a dependency descriptor are exceeded, an additional transaction may be created automatically to restore the consistency of the interdependent data. The execution of this transaction may affect consistency requirements specified by some other D^3 and another transaction may be activated. We refer to a tree of transactions, initiated by an update to an interdependent data, as a *polytransaction* [519].

Polytransactions allow automatic maintenance of multidatabase consistency, based on the weak consistency requirements. When immediate consistency is not required, an update to a data item can be decoupled from the actions that would need to accompany it to restore the mutual consistency. These actions can be executed later, within the limits specified by the dependency descriptor. Such an approach also allows to reduce the number of remote transactions that are needed to maintain mutual consistency. These consistency-restoring transactions may need to be executed only when the consistency requirements are violated rather than after every update.

The weak mutual consistency criteria may allow a situation in which an object does not reflect all the changes to the data items to which it is related, but the inconsistency remains within the allowed limits. We discuss the conflicts between polytransactions and the correctness of concurrent execution of polytransactions. While the serializability can be used when no temporal terms are involved, we introduce the concept of temporal serializability that considers serializable schedules for nontemporal constraints as well as temporal precedence for temporal constraints.

This chapter is organized as follows. Section 17.2 reviews our framework for specifying interdependent data and presents a conceptual architecture in which the specifications can be enforced. Section 17.3 discusses the correctness of dependency specifications. Section 17.4 defines the polytransaction mechanism. Section 17.5 discusses the consistency states of interdependent data objects and various operations/events that lead to the state transitions. We also discuss the need to control updates by local transactions. Section 17.6 discusses the correctness of the execution of a single polytransaction and the issues related to concurrent execution of polytransactions. Finally Section 17.7 provides conclusions.

17.2 Background

In this section we briefly review our framework for the specification of interdependent data. A more detailed discussion can be found in [479, 519]. We then discuss a conceptual architecture that could support maintenance and enforcement of specifications.

17.2.1 Specification of Interdependent Data

Our framework for specifying interdatabase dependencies consists of three components: dependency information, mutual consistency requirements, and consistency restoration procedures. While these components have been addressed in the literature separately, in our opinion they represent facets of a single problem that should be considered together. Data dependency conditions are similar to Unlike integrity connstraints

in distributed DBMSs [525], full integrity between interdependent data in different databases may be necessary at all times or not possible in many environments. We use *Data Dependency Descriptors* (D^3) to specify the interdatabase dependencies. Each D^3 consists of an identification of related objects and a directional relationship defined in terms of the three components just mentioned. A D^3 is a 5-tuple:

$$D^3 =< \mathcal{S}, U, P, C, \mathcal{A} >$$

where:

- $\mathcal{S}$ is the set of *source data objects*,
- U is the *target data object*,
- P is a boolean-valued predicate called *interdatabase dependency predicate* (dependency component). It specifies a relationship between the source and target data objects, and evaluates to true if this relationship is satisfied.
- C is a boolean-valued predicate, called *mutual consistency predicate* (consistency component). It specifies consistency requirements and defines when P must be satisfied.
- $\mathcal{A}$ is called *action* component and contains information about how the consistency between the source and the target data object may be restored.

The objects specified in $\mathcal{S}$ and U may reside either in the same or in different centralized or distributed databases, located in the same or different sites. We are particularly interested in those dependencies in which the objects are stored in different databases managed by a local database management system (LDBS).

The dependency predicate P is a boolean-valued expression specifying the relationship that should hold between the source and target data objects.

The consistency predicate C, contains mutual consistency requirements specified along two dimensions – the data state dimension s, and the temporal dimension t. The specification of the consistency predicate can involve multiple boolean valued conditions, referred to as *consistency terms* and denoted by c_i. Each consistency term refers to a mutual consistency requirement involving either time or the state of a data object.

The action component $\mathcal{A}$, is a collection of *consistency restoration procedures*. They specify actions that may be taken to maintain or restore consistency. There can be multiple restoration procedures, and the one to be invoked, may depend on which conditions lead to the inconsistency between interdependent data. The execution mode can be defined for each restoration procedure to specify the degree of coupling between the action procedure and its parent transaction (i.e., the transaction that invokes it).

The set of all D^3s together constitutes the *Interdatabase Dependency Schema, (IDS)* [519]. It is conceptually related to the Dependency Schema presented in [373].

Alternative ways to specify consistency requirements among related data have been also discussed. *Identity connections* [609] introduced a time based relaxation of mutual consistency requirements among similarly structured data items. Relaxed criteria based on numerical relationships between data items have been proposed in [34]. Quasi-copies support relaxed consistency between primary copies and quasi-copies, based on several parameters [16]. E-C-A rules can be used to specify the C and A components of D^3s [138]. In [90, 91] interdatabase constraints are translated to production rules in a semi-automatic way, using a language based on SQL, to specify consistency between

interrelated data objects. The derived production rules enforce consistency by generating operations automatically. However, tolerated inconsistencies are not allowed in that approach.

In the following example, we illustrate the use of D^3s, for the specification of consistency requirements between a primary and a secondary copy, as a special case of replicated data. We assume that copies of data are stored in two or more databases. The dependency between all copies requires that changes performed to any copy are reflected in other copies, possibly within some predefined time. Let us consider the relation *D1.EMP* (i.e., relation *EMP* stored in database *D1*) and its replica *D3.EMP_COPY*. We assume that *EMP* must always be up-to-date, but we can tolerate inconsistencies in the *EMP_COPY* relation for no more than one day. The following pair of dependency descriptors represents this special type of replication:

S: D1.EMP	*S* : *D3.EMP_COPY*
U: D3.EMP_COPY	*U* : *D1.EMP*
P: EMP = EMP_COPY	*P* : *EMP = EMP_COPY*
C: ε*(day)*	*C* : 1 *update on S*
A: Duplicate_EMP	*A* : *Propagate_Update_To_EMP*
(EMP is copied to EMP_COPY).	as *coupled & vital*
	(The update on EMP_COPY
	is repeated on EMP.)

The two descriptors above, represent a case of a bi-directional dependency between two database objects. The target object in one descriptor is the source object in the other descriptor. The consistency predicate *P* is exactly the same in both D^3s. The consistency between the two objects is specified as follows: whenever an update is performed on *EMP_COPY*, it must be reflected immediately in the *EMP* relation. On the other hand, consistency will be restored in the *EMP_COPY* with respect to the updates on *EMP* only at the end of the day (although there may be a number of updates performed to the *EMP* during that day).

17.2.2 Conceptual System Architecture

In this chapter we are concerned with the effects that an update operation on a data object may have on the related data managed by other systems. We assume that a system involved in the management of interdependent data consists of a data manager (*DM*), the database(s) it manages, and a new component called a dependency subsystem that is introduced below. A *DM* can be a DBMS that manages data in the database(s). With different types of *DM*s, interfaces vary. However the basic issues of managing interdependent data that are discussed here apply in all cases.

A possible conceptual system architecture that can be used to maintain interdependent data objects is illustrated in Figure 17.1. Every database participating in a multidatabase environment is augmented with a *Dependency Subsystem (DS)*, that serves a dual purpose: it acts as an interface between different databases where incoming transactions (updates and queries) are to be executed, and also monitors the consistency of interdependent data. The evaluation of the consistency between interdependent data implies knowledge of all events and operations in the system. To facilitate monitoring of relevant events and operations, we assume the existence of a monitor [470], as an internal part of each *DS*. To identify all consistency terms that may be violated due to updates on interdependent data, the monitor in the *DS* is informed of all updates of the data objects, which may require adding appropriate

commands to a transaction (e.g., see [517]) or linking the data manipulation routines with procedures that inform the monitor. In addition, the monitor needs to know the changes to interdependent data before and/or after the updates are performed. If the consistency requirements are violated, the *DS* would invoke appropriate actions to execute restoration procedures on the data managed by the *DM*s. Under the proposed architecture each site can be monitored independently. *DS*s at different sites can communicate with each other, exchanging information between monitors.

A transaction submitted to a *DS* is analyzed before being executed by the *DM*. In particular, the *DS* consults the *IDS* to determine whether the data accessed by the transaction are dependent on data controlled by other *DM*s. Then, a series of related transactions may be scheduled for execution to preserve mutual consistency of related data. The initially submitted transaction, and related transactions corresponding to restoration procedures, are submitted to the *DM*s that manage the databases where data to be updated are stored. After the execution of a restoration procedure, the values of the various components of the dependency descriptors that are maintained by the monitor, including the consistency terms c_i, are updated. Special precautions must be taken to properly serialize the execution of these monitoring transactions with respect to the updates, to assure that the values observed by the monitor correspond to consistent snapshots of data. Once it is determined that the inconsistency among the interdependent data has exceeded the limits specified in the *IDS*, either in the terms of the data state, or temporal constraints, appropriate procedures are invoked to restore the consistency.

The *IDS* itself can be either centralized or distributed over multiple systems. In the latter case, only those dependency descriptors that have their source or target objects stored locally might be kept in the *IDS* partition associated with the *DS*.

17.3 Correctness of Dependency Specifications

In this section we discuss the issue of correctness of interdependent data specifications, i.e., the D^3s in the *IDS*. We first introduce a graphical representation of the *IDS* and then discuss the correctness issues that can be determined by static examination of the *IDS*. We call interdependency specifications to be incorrect when they specify contradictory requirements, or when there are no potential schedules to enforce them. In this section we investigate two cases that lead to incorrect specifications in the *IDS*: first, we identify incorrect specifications due to potential conflicts among the C components of D^3s. Then, we show incorrect specifications due to conflicts among the P components of the D^3s.

17.3.1 Dependency Graph

Each descriptor D^3, identifies a relationship between the source and the target data objects. The set of D^3s that comprise the *IDS* can be represented as a directed graph that we call the *dependency graph*. Figure 17.2 illustrates an example of such a graph.

A database object participating in an *IDS* is called a *Data Object Vertex* and represented by a circle. A dependency descriptor D^3 between two database objects, called a *Dependency Vertex*, is represented by a square. The edges of the graph represent the directionality of the descriptors between the objects. An edge of the graph originates from a data object vertex (source data object), passes through a dependency vertex (representing the D^3 itself), and terminates at another data object vertex (the target

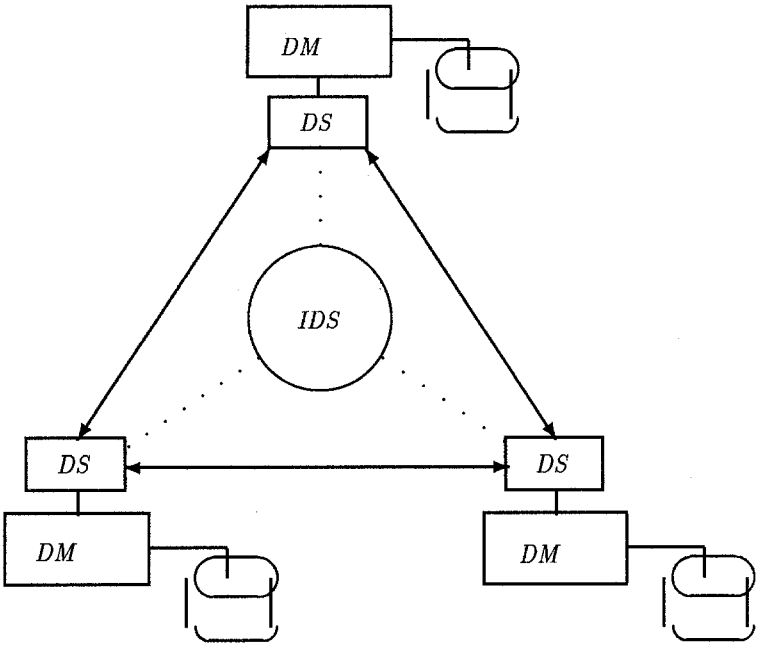


Figure 17.1: Architecture of a system for managing interdependent data

data object). If a D^3 has multiple source data objects, we create edges from all participating source objects to the descriptor vertex. From that descriptor vertex, another edge is directed to the target object vertex of the D^3. In Figure 17.2 the descriptor D_1^3 specifies the relationship between the source object o_1 and the target object o_3. Descriptor D_2^3 connects two source objects o_1 and o_2, with the target object o_4. An object vertex with no incoming edges is called a *top vertex*. In Figure 17.2 objects o_1 and o_2 are top vertices.

Every data object vertex o_i has outgoing edges to all dependency vertices d_i in which it participates as a source object. A dependency vertex d_i, has incoming edges

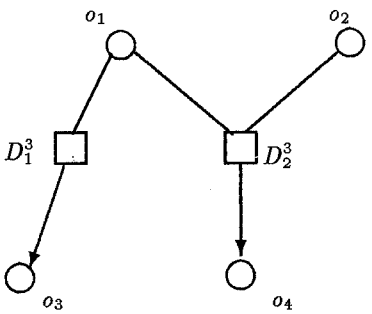


Figure 17.2: Interdependent data objects and their descriptors

from all the data object vertices that are sources in the dependency descriptor it represents, and an outgoing edge to the vertex that represents its target data object. Hence, we have

$$\forall d_i,\ \textit{in-degree}(d_i) \in \{1, n\}$$
$$\forall d_i,\ \textit{out-degree}(d_i) = 1, \text{ and}$$
$$\forall o_j,\ \textit{out-degree}(o_j) \in \{0, n\}$$

Now, we examine correctness of IDS specifications.

17.3.2 Correctness Requirements Involving Consistency Predicates

A possible case of conflict arises when a new D^3 has a target object that is also the target object of an existing D^3. This case introduces the notion of conflict between data descriptors in the *IDS*. One possibility is to characterize two dependency descriptors as conflicting if they have the same data object as their targets (see Figure 17.3). The problem with this specification is as follows. Suppose that o_1 and o_3 are data objects

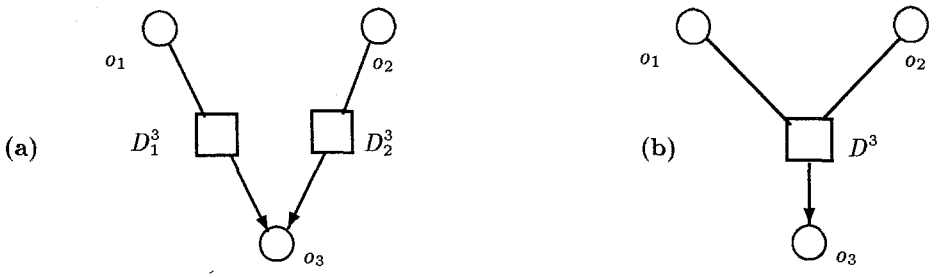


Figure 17.3: Conflicting Data Descriptors

related through descriptor D_1^3 and o_2 is related with object o_3 through descriptor D_2^3. The dependency predicates P_1 of D_1^3, and P_2 of D_2^2 are given below:

$$P_1 : o_3 = o_1 + 3$$
$$P_2 : o_3 = o_2 * 2$$

Additionally, the consistency predicates C of both D^3s specify that the updates on the sources should be propagated to the targets immediately. If the initial values are $o_1 = 1, o_2 = 2$ and $o_3 = 4$, both D^3s are satisfied. Suppose that o_1 is updated to 5, which is propagated to o_3 through descriptor D_1^3, updating o_3 to 8 (5+3). Then, P_2 predicate in D_2^3 is violated. If o_2 is now updated, the result of this update is propagated to o_3 invalidating the effects of the previous update on o_3 resulting from D_1^3. In general, there is no way the system can ensure that both dependency predicates are satisfied simultaneously. The problem is due to the existence of two D^3s targeting the same object, whose consistency predicates overlap (in this particular example they are exactly the same). In general, it is not clear what semantics these two D^3s convey, and situations like this may lead to uncertainty about the D^3 that updates the target. It is similar to the case where two rules have the same right hand side and both are triggered. Stonebraker et. al, refer to such updates as *nonfunctional* [553]. One way

to avoid this problem is not to allow specification of a D^3 that targets a data object which is already a target of another D^3.

However, this approach is rather restrictive, since there may be cases that a target object must be mutually consistent with more than one source data objects. Our framework is flexible enough to allow in a single D^3 the specification of consistency between multiple sources and one target. For this reason, we recommend incorporating all source objects having the same target into a single D^3. The collection of all consistency information regarding the same target object into a single D^3 gives a uniform structure to the *IDS*. The advantages of this approach include a better description of the IDS, a natural criterion for fragmenting it, and also a safeguard against the creation of new D^3s that may potentially undo actions that other D^3s enforce. Schematically we illustrate this by re-directing all edges terminating at the same target object, to the same descriptor vertex containing the D^3, as shown in Figure 17.3b. The target object always belongs to one D^3. Whenever a new D^3 is to be created targeting an object that is already the target of an existing D^3, the two descriptors must be merged to one with the P, C, and A predicates, appropriately augmented.[§] Therefore, we impose the following restriction to the dependency graph: each data object vertex has at most one incoming edge, i.e., its $\textit{in-degree}(o_i) \in \{0, 1\}$.

17.3.3 Correctness Requirements Involving Dependency Predicates

In this subsection, we discuss the correctness of dependency predicates, first limiting our discussion to D^3s involving singleton source sets.

Example: Let us consider the following pair of D^3s:

D_1^3::		D_2^3::	
	$S_1 : o_i$		$S_2 : o_j$
	$U_1 : o_j$		$U_2 : o_i$
	$P_1 : o_j = o_i + 2$		$P_2 : o_i = o_j - 3$
	$C_1 :$ *immediately*		$C_2 :$ *immediately*
	$A_1 :$ *Update_*o_j		$A_2 :$ *Update_*o_i

In this example, a cycle exists between objects o_i and o_j in the dependency graph. If an update occurs on o_i so that P_1 no longer holds, o_j becomes inconsistent, and must be updated by executing the procedure *Update_*o_j. After o_j has been updated, the P_2 predicate is violated. That means, o_i is inconsistent, and requires immediate execution of the *Update_*o_i procedure. This could be repeated indefinitely.

The above abnormal behavior is caused by the fact that each P_i predicate of the corresponding D^3s is not the *inverse* of the other. If P_2 were $o_i = o_j - 2$ instead of $o_i = o_j - 3$ the cycle would be acceptable since after performing A_2, P_1 and P_2 are both satisfied. Therefore, no more updates will be performed due to restoration procedures. Cycles that do not cause infinite number of updates are harmless and are referred to as *stable*.

One way to avoid unstable cycles is to disallow cycles in the IDS. However, this may be too restrictive, since there may be applications that require cyclic dependencies. Therefore, we will require that all cycles in the IDS are stable.

To generalize the previous discussion let us assume a cyclic dependency graph as illustrated in Figure 17.4. All updates resulting from an update to o_1 can be propagated

[§]If the designer of the D^3 is knowledgeable about the semantics of the existing D^3 targeting the same object, it is easy to merge the two D^3s into one.

further, up to o_k. The last dependency D_k^3 introduces a cycle by linking objects o_k and o_1. Let P_i be the dependency predicate of $D_i^3, i = 1, 2 \ldots k$. In order for the cycle to be stable, the composition of all predicates involved in the cycle must be equal to identity, i.e., $(P_1 \circ P_2 \circ \cdots \circ P_{k-1} \circ P_k) = I$. For example, if

$P_1 : o_2 = o_1 + 3,$
$P_2 : o_3 = o_2 - 1,$
then for a stable cycle we must have
$P_3 : o_1 = o_3 - 2.$

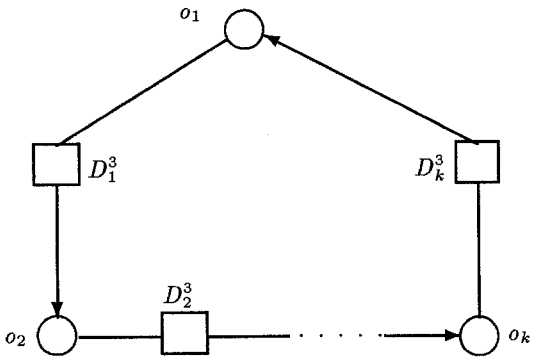


Figure 17.4: A cyclic dependency graph

The correctness of dependency specifications in the *IDS*, can be determined by a static analysis of the dependency graph. Conflicting descriptors and unstable cycles can be identified, every time a new D^3 is added, and appropriate actions can be taken to maintain an *IDS* with correct specifications. The above notion of correctness can be extended to the cases that involve multiple source objects.

So far, we investigated some cases of correctness of specifications that can be identified by a static analysis of the dependency graph. In the remainder of the chapter we will investigate the issues of maintenance and enforcement of consistency between interdependent data that are being updated by transactions.

17.4 Polytransactions

Transaction management technologies have been developed to ensure proper interleaving of concurrent activities, and to maintain database consistency. Most of the concurrency control methods proposed for distributed databases use the concept of one-copy serializability to support mutual consistency of related data. In most cases, its use is limited to replicas. In the context of semantically related data maintained in multiple databases, one-copy serializability, and the corresponding replica-control mechanisms, may be unnecessarily stringent, expensive or difficult to implement. A possible approach to this problem is to use application/operation semantics to allow harmless non-serializable conflicts [209, 185].

In an environment consisting of multiple autonomous systems, the concept of global (multidatabase) transaction that is composed of a well-defined set of subtransactions

may be too restrictive. The need to relax atomicity, isolation and durability are discussed in various papers in [172]. The transactions we are interested in may not have all the *ACID* properties [240]. We require that a transaction is correct in a sense defined by the semantics of the application. Depending on the application, requirements weaker than absolute consistency may optionally be imposed on updates performed on interdependent data.

Some of the earlier efforts that support enforcement of weaker consistency criteria, but more limited that those that can be expressed using D^3s, or in a limited system environment, are as follows. Mechanisms to enforce ϵ-serializability [631] and N-ignorance [333] have been proposed. Demarcation protocols allow maintenance of arithmetic constrains [34]. An enforcement mechanism for some types of interdependent data specifications was proposed by extending a distributed transaction management approach in [517]. Actions to restore consistency between interdependent data were discussed in the context of active databases with nontemporal constraints are discussed in [138, 257]. An idea comparable to triggers for the management of interdependent data was discussed in [407]. The authors used a table driven approach to schedule complementary updates (or invoke a contract) whenever a data item involved in a multi-system constraint was updated. The parent transaction would then terminate, without waiting for a chain of complementary actions to take place.

To support enforcement of data consistency requirements as specified by D^3s, we use the more flexible notion of a *polytransaction* to describe a sequence of related update activities. An important difference between polytransactions and the above mentioned extended transaction models is that polytransactions do not assume that a set of component (sub-)transactions is known in advance. In this respect, polytransactions are closest to the transactional model for long running activities proposed in [140]. Polytransactions are dynamically generated when an update or other event could result in violation of the consistency specification given in D^3. Additionally, polytransactions allow selective and controlled relaxation of atomicity and isolation criteria, as discussed later.

A polytransaction P is a "transitive closure" of a transaction T submitted to an interdependent data management system. The transitive closure is computed with respect to the IDS. A polytransaction can be represented by a tree in which the nodes correspond to its component transactions and the edges define the "coupling" between the parent and children transactions. Given a transaction T, the tree representing its polytransaction P can be determined as follows. For every data dependency descriptor D^3 such that a data item updated by T is among the source objects of the D^3, we look at the dependency and consistency predicates P and C. If they are satisfied, no further transaction will be scheduled. If they are violated, we create a new node corresponding to a (system generated) new transaction T' (child of T) to update the target object of the D^3. T' will restore the consistency of the target object. Specification of weaker mutual consistency criteria will result in less frequent violations of consistency. Hence, the restoration procedures (and the corresponding children transactions) will be scheduled less often.

When a user submits a transaction that updates a data item that is related to other data items through a D^3, this transaction may become the root of a polytransaction. Subsequently, the system responsible for the management of interdependent data uses the IDS to determine what descendent transactions should be generated and scheduled in order to preserve interdatabase consistency. Execution of a descendent transaction, in turn, can result in generating additional descendent transactions. This process continues until the consistency of the system is restored as specified in the IDS.

The ways by which a child transaction is related to its parent transaction within a polytransaction are specified in D^3 by the *execution mode* of the action component. This relationship is indicated as a label of the edge between each parent and its child in the polytransaction tree. A child transaction is *coupled* if the parent transaction must wait until the child transaction completes before proceeding further. It is *decoupled* if the parent transaction may schedule the execution of a child transaction and proceed without waiting for the child transaction to complete.

If the dependency schema requires immediate consistency, the nested transaction model may be used, in which the descendent transactions are treated as subtransactions which must complete before the parent transaction can commit. A two-phase commit protocol may be used in this case. A coupled transaction can be *vital* in which case the parent transaction must fail if the child fails, or *non-vital* in which case the parent transaction may survive the failure of a child [210].

Traditional transactions are characterized by the ACID properties. Polytransactions provide a mechanism to support these properties if needed, using appropriate specification of the consistency predicate and the execution mode action predicate of the dependency descriptors that are used to create a polytransaction. It is the responsibility of the D^3 designer to specify which of the ACID properties a polytransaction may have, as follows:

Atomicity A polytransaction supports atomicity if all of its transactions are executed in *vital* mode. Atomicity is relaxed if at least one of its transactions is executed in *non-vital* mode. ¶

Consistency We discuss issues of consistency of polytransactions later in Section 17.6 of this chapter.

Isolation A polytransaction supports isolation if all of its transactions are executed in *coupled* mode, and the D^3 consistency requirements specify *immediate* consistency. Otherwise, isolation is relaxed since values of a data object that is within a consistent state, can be seen by other transactions.

Durability As in other extended transaction models, if every transaction of a polytransaction is durable, then the whole polytransaction supports durability.

Several new transaction paradigms have been proposed recently in the literature that are based on various degrees of decoupling of the spawned activities from the creator (e.g., [327]). Triggers used in active databases [139] are probably the best known mechanism in this group. The main problem with asynchronous triggers is that the parent transaction has no guarantee that the activity that was triggered will, in fact, complete in time to assure the consistency of the data.

To allow the parent transaction some degree of control over the execution of a child transaction, the concept of a VMS mailbox has been generalized in [210]. Similar ideas have been presented in [48], and in [258], where the notion of a "persistent pipe" has been introduced. Both generalized mailboxes and persistent pipes allow the parent transaction to send a message to a child process and know that the message will be eventually delivered. If such a guarantee is sufficient, the parent transaction may then commit, without waiting for the completion of the actions that were requested. The parent or its descendant may check later if the message has been indeed received and take a complementary or compensating action.

¶By definition only *coupled* transactions can be *vital* or *non-vital*.

17.5 Consistency of Interdependent Data

In this section we present the concept of consistency of interdependent data. First we discuss about the states, the events, and the transitions that affect the consistency of interdependent data objects. Then we define the measures that can be used to quantify the consistency of interdependent data. Finally we discuss the issue of when a target data object can be directly updated outside of the polytransaction mechanism without violating consistency requirements of interdependent data.

17.5.1 States of Interdependent Data Objects

For every dependency descriptor, its source and target data objects must be consistent according to the specified degree. However, the source and the target objects go through various states of consistency: Initially the target object is fully consistent with its source objects. Then, updates on the source object may violate the relationship between them, but the discrepancy may still be within the limits specified in the D^3 (partial consistency). Finally, the source and target will diverge beyond the tolerable limits (inconsistency). Then a restoration procedure will update the target, restoring full consistency. This scenario is repeated in a cyclic manner. In this section we classify the various states of consistency a data object can be in, and we introduce metrics to identify in detail how consistent source and target objects can be at any point in time, with respect to a given D^3.

Definition 17.5.1 *At any instant of time, the target data object within a given D^3 is defined to be* current *with respect to a given D^3, if the dependency predicate P is satisfied. The data item is said to be* consistent *if the dependency predicate P is violated, but the consistency predicate C is not. Hence, if a target item is current then its consistency is implied, but the opposite is not true: an object may be consistent and not current. If both P and C are violated, the target data object is* inconsistent *and the corresponding D^3 is* violated. *A multidatabase with an IDS is defined to be in a* consistent state *if every target data object, is either current or consistent, i.e., $\forall D^3 \in IDS, D^3.C = true$.*

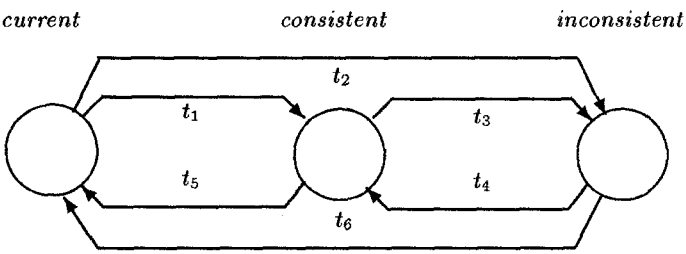


Figure 17.5: Transitions of a target data object

Figure 17.5 illustrates the transitions among the three states (current, consistent and inconsistent) of a target data object. These transitions occur as a result of the changes in values of the P and C predicates. The transitions are explained below:

- Transition t_1: the state of the target data object changes from current to consistent, as a result of an update to a data item belonging to the source set $\mathcal{S}$. The consistency predicate C continues to be satisfied, although the values of the temporal or data state consistency terms may be changed.
- Transition t_2: a current target data object becomes inconsistent as a result of an update on an object in the source set $\mathcal{S}$. Either the data state and/or the temporal terms violate the consistency predicate C.
- Transition t_3: a target data object is transformed from the consistent to the inconsistent state, due to a change in the terms of the C predicate.
- Transition t_4: the purpose of a restoration procedure is to change the state of a target data object, to either consistent or current. Transition t_4 occurs when we perform a *partial restoration*, so that an inconsistent data object becomes consistent, but not current. Various cost policies may indicate that a partial restoration to a consistent state is more appropriate than a (sometimes more expensive) restoration to a current state.
- Transition t_5: this transition occurs when the state of the target data object changes from consistent to current. When the target object is consistent, we have the choice of either doing nothing, or executing a restoration procedure to make the data object current. This choice can be made considering performance parameters, load balancing, etc. Execution of restoration procedures in this manner will be referred to as *eager restoration of current state.*
- Transition t_6: an inconsistent data object becomes current, by invoking a restoration procedure in one of the following ways:
  (a) the restoration procedure is executed when it is discovered that C is violated. This is referred to as *late restoration of current state.*
  (b) the particular data object is marked as inconsistent but no action is taken until an access to the target object is attempted. Then, the restoration procedure is activated before the access is granted. This method is referred to as a *lazy restoration of current state.* This strategy is used in [517].

External events, such as source updates or time restrictions, change the state of consistency of a data object in the *IDS*. In order to estimate the degree of inconsistency, we need to precisely identify how "far away" a target object is from its source, or how inconsistent it is with respect to the consistency requirements specified in the D^3s. However, it is not enough, to say that an object is current, consistent or inconsistent. The monitor must be able to identify the relative discrepancy between the source and the target at any instant of time. Below, we present definitions that are used as metrics to calculate the relative consistency between the source and the target objects. From now on, the word consistent will be used to mean either current or consistent, unless we explicitly specify otherwise.

17.5.2 Measures of Consistency

We are interested in the semantics of consistency requirements of interdependent data. We estimate the degree of inconsistency between two objects connected by a D^3. We will view each consistency predicate C of a D^3, as a pair of the two dimensions: time and data state.

As an example, let us consider the following D^3:

$\mathcal{S}$: a
$\mathcal{U}$: b
$\mathcal{P}$: $a = b$
$\mathcal{C}$: $c_1 \vee c_2$
$c_1 = 5$ *versions of* a
$c_2 = e(48$ *hours*$)$
A : Update_Target

This example identifies two interdependent data objects a and b. The target data object b is a replica of the source data object a, as specified by the dependency predicate P. If the source object is updated the target object becomes inconsistent. We specify that we can tolerate inconsistencies between the source and the target up to 5 versions of a or until a 48 hour period ends.

Definition 17.5.2 *A state-time-pair (stp) is a pair $\langle s, t \rangle$, where s is a value in the data state dimension, and t is a value in the time dimension.*

A value along the state dimension identifies the data state of a database object, and a value in the time dimension specifies time. We use the syntax described in a previous section for data state and temporal consistency terms. The pair $\langle 5$ *versions*$, 48$ *hours*$\rangle$ is an example of an *stp*. In general, s and t, can be logical formulae consisting of various types of consistency terms. For example, if $C = 5\%(Employee)$, then the data state dimension of this term is $s = 5\%(Employee)$. A detailed presentation of the different consistency terms can be found in [519]. For this chapter, we assume that both s and t can be represented as linear functions.

Definition 17.5.3 *The limit of discrepancy along a D^3, $L(D^3)$, is an stp, $\langle d_s, d_t \rangle$, where d_s and d_t specify the maximum allowed discrepancy along the data state and temporal dimensions, between the source and the target database objects of a dependency descriptor.*

For example, the limit of discrepancy between the source and the target objects of the above D^3, is either 5 versions or 48 hours specified as $L(D^3) = \langle 5$ *versions*$, 2$ *days*$\rangle$. The limit of discrepancy of every D^3 is constant, and can be extracted from the C predicate of the D^3 itself.

Definition 17.5.4 *The consistency restoration point of a D^3, $I(D^3)$, is an stp, $\langle i_s, i_t \rangle$, where i_s and i_t specify the values along the data state and time dimension when consistency between source and target objects was restored.*

The value of I changes every time we restore consistency between the source and the target data objects. It is the initial point of reference, used in calculation of discrepancy between source and target data objects. In the above D^3, assuming that the value of the 30th version of the source object a was propagated to the target object b, at 10 a.m on 26th of February 1992, then $I(D^3) = \langle 30, 26 - 02 - 1992@10 \rangle$.

Definition 17.5.5 *The Current Value C of discrepancy along a D^3, $C(D^3)$, is an stp, $\langle c_s, c_t \rangle$, where c_s and c_t identify the distance between current state of the source and target objects measured in terms of data state and time.*

The data state dimension c_s of the current value changes every time an update is performed on the source object, and the temporal dimension c_t of the current value changes constantly with time. If an update has been performed on object a, 15 hours after the last restoration of consistency the current value of our D^3 is $C(D^3) = \langle 1\ versions, 15\ hours \rangle$.

Definition 17.5.6 *The Final Value F of a D^3 is an stp, $\langle f_s, f_t \rangle$, defining a point when the consistency between source and target objects must be restored.*

The value of the final state is calculated as the sum of the consistency restoration point plus the specified limit of discrepancy, i.e., $f_s = i_s \oplus d_s$. The operator $\oplus$ denotes summation on data states and carries a broader meaning than the regular arithmetic operator "+", since we have different types of data state terms that must be "added" together. The f_t is calculated as the sum of the time of consistency restoration plus the specified limit of discrepancy, i.e., $f_t = i_t + d_t$. In our D^3, we have $F = \langle 35\ versions, 28-02-1992@10 \rangle$.

Definition 17.5.7 *The source $\mathcal{S}$ and the target $\mathcal{U}$ data objects of a D^3 become inconsistent with respect to that D^3, when the value of the restoration point $I(D^3)$ plus the current value $C(D^3)$ exceed the final value $F(D^3)$, i.e., when $I(D^3) + C(D^3) > F(D^3)$.*

Hence, D^3 is violated (the state of the target object is inconsistent) when at least one of the following cases occur:

1. the value of the data state at restoration point i_s plus the value of the current data state c_s, exceed the value of the final state f_s, i.e., $c_s \oplus i_s > f_s$, or
2. the value of the temporal dimension at restoration point i_t, plus the value of the current temporal dimension c_t, exceed the final deadline f_t, i.e., $c_t + i_t > f_t$.

As a direct consequence of the above, the source and target data objects are current or consistent if the restoration point of the descriptor, $I(D^3)$ plus the current value of the descriptor $C(D^3)$ has not reached the final value $F(D^3)$ of the same D^3, i.e., $I(D^3) + C(D^3) \leq F(D^3) \Leftrightarrow \mathcal{S}$ is consistent with $\mathcal{U}$.

17.5.3 Updatability of Objects

The IDS encapsulates the information about data objects and dependency descriptors and can be used to maintain the consistency of related data through polytransactions. One of our primary concerns in the management of interdependent data is to assure that applications will always access interdependent data in a consistent state. Updates performed on a data object represented by one of the top vertices of the IDS are guaranteed to propagate to all the dependent objects, to maintain mutual consistency. However, if an external update (i.e., an update not resulting from the polytransaction mechanism) is performed directly on a data object that is a target of a dependency descriptor, then we may introduce inconsistencies, which cannot be corrected by polytransactions. Such updates occur outside of the consistency maintenance framework, specified by the set of dependency descriptors. The results of such updates can be invalidated by subsequent invocations of polytransactions. Also, it is not clear whether such updates should be propagated along the dependency descriptors.

One possible solution to the above problems that guarantees consistency of interdependent data read by applications, is to disallow external updates on target data

objects. Therefore, external updates to the interdependent data are allowed only on the data objects represented by top vertices. All data objects managed by the IDS can be read by applications at any time.

Although the above mechanism could guarantee mutual consistency of interdependent data of the entire IDS, it is quite restrictive. Frequently, we would like to perform (possibly limited) external updates on target data objects in addition to the updates propagated from source data objects by polytransactions. At the same time we do want interdependent data to still be mutually consistent as specified in the IDS. We can achieve this, by allowing external updates on target data objects only if they do not violate consistency requirements specified by any D^3 in IDS. Since we realize that maintaining consistent data using polytransactions and allowing updates outside the polytransaction mechanism, are two contradictory goals, we discuss below a compromise solution, based on restricting external updates.

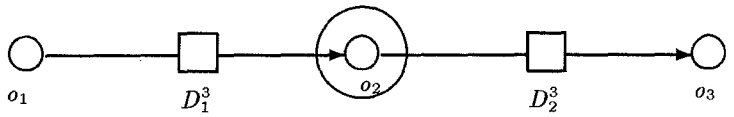


Figure 17.6: Consistency zone for object o_2

Let us consider an object (e.g. o_2) which participates in two D^3s: as a target in D_1^3, and as a source in D_2^3 (Figure 17.6). We will examine the effects of an external update to o_2 on other objects to which o_2 is related. If o_2 is externally updated, the dependency P_1 and consistency C_1 predicates of D_1^3 may be violated. In this case, D_1^3 would be violated, which is not acceptable according to our criterion of mutual consistency. However, if the consistency predicate C_1 is still satisfied, the external update was within consistency limits. This kind of update does not violate the consistency requirements between o_1 and o_2. On the other hand, if we examine the relationship between o_2 and o_3 specified by descriptor D_2^3, we may find that the update on o_2, may have been propagated to o_3, which is not desirable. The propagation of external updates on target objects through the polytransaction mechanism is undesirable, if the updates do not originate from a top object, because we will face the problems of inconsistency, described earlier. Therefore, an external update of object o_2 affects all D^3s that are adjacent.

The consistency predicates of these descriptors define a *zone of consistency* around a data object. In general, the data objects that are targets of a dependency descriptors represent derived data and, hence, should not be directly updated unless a complementary dependency descriptor to a source data item exists. However, we may allow direct updates to a data object, if they would not violate any mutual consistency requirements specified in the D^3s it participates, either as a source object or as the target object. Such updates maybe useful for temporarily changing the value of a data object to a new value for the purpose of running a local application. The updated value represents a non-permanent patch and would be overwritten by the "correct" value, by the polytransaction originating from the source data item. However, such direct updates on target data objects should also be directly sent to the top object. The top object update will flow through the polytransaction mechanism (and may be combined with other updates) to overwrite this target object.

Definition 17.5.8 *The zone of consistency of a data object is specified by an stp,*

$\langle c_s, c_t \rangle$ that is the intersection of the limits of discrepancy $L_i(D^3)$ of all the descriptors in which the object participates (as source or target).

If an external update changes the value of the target data object in such a way that the object remains within its zone of consistency, then the update is allowed. The zone of consistency of an object can be adjusted by changing the limits of discrepancy allowed by the relevant dependency descriptors. However, the stricter the consistency specification is, the smaller zone of consistency we have, which results in limiting external updates that are allowed. We also see that the updates are performed according to semantic criteria of consistency specified in D^3s, as opposed to a fixed number of updates [631], or a predetermined number of transactions [333]. This is because we believe that even a single update can irrecoverably destroy the consistency between related data objects, if semantic information regarding affected data objects is not taken into account.

In addition to the updates we mentioned above, we can also allow updates on target objects if there is another D^3, directed from the target object to its source with execution mode marked as *vital and coupled* and *immediate* consistency specification. We allow such updates, since they invoke the polytransaction mechanism to propagate them immediately. Such example was previously specified using a pair of D^3s identifying a case of replicated data with primary and secondary copies.

17.6 Concurrent Execution of Polytransactions

In this section we discuss issues concerning the consistency of a system of interdependent data in the presence of concurrent polytransactions. A polytransaction starts at a site and its (sub)transactions may propagate to various other sites to maintain or restore mutual consistency of related data. A number of polytransactions may be active at the same time, updating related data objects. We assume that initially the system contains consistent interdependent data objects, i.e., the set of all D^3s in the IDS is satisfied. In this section we will investigate the effects of concurrent polytransactions on the consistency of the interdependent data objects. We first present the correctness criterion for the execution of a single polytransaction. Then we define when a concurrent execution of polytransactions is correct. In the discussion below, we use P_i to denote a polytransaction and P_i^j to specify a transaction of polytransaction P_i that executes at site j.

17.6.1 Correctness of the Execution of a Single Polytransaction

First, consider the execution of a single polytransaction. As we described earlier a polytransaction starts at a particular site, originated by an external update or a temporal event, and propagates dynamically to other sites where interdependent data are stored.

Definition 17.6.1 *The projection of operations from different polytransactions at a site over time represent the local history at that site.*

A polytransaction accesses various objects in different sites, by means of its transactions. We identify the set of data objects a transaction reads as the read-set of the transaction, and the set of the data objects it writes as the write-set of the transaction.

Definition 17.6.2 *A D^3 is incident to a (poly)transaction at a site, if the set of its source objects intersects the write-set of a transaction, or its target object belongs to the read set of the transaction.*

In the traditional transaction model, it is assumed that each transaction when executed alone on a consistent database will execute correctly, transforming the database to another consistent state [49]. The equivalent requirement for polytransactions is that a polytransaction when executed alone on a consistent system of interdependent data, will terminate and leave the system consistent. The consistency is determined by the IDS. The above can be stated more formally as:

Definition 17.6.3 *The execution of a single polytransaction is correct if and only if:*

- *Every transaction of a polytransaction obeys intra-polytransaction precedences in all local histories.*
- *After the execution of every transaction of a polytransaction the temporal and data state predicates of all D^3s incident to the polytransaction at all sites are satisfied.*

17.6.2 Conflicts in Polytransactions

Now, we consider concurrent execution of polytransactions. In our model we incorporate a two level approach to the concurrency control. At the lower level are transactions that belong to polytransactions. At this level we rely on traditional concurrency control protocols supported by the DMs and we assume that these transactions are executed correctly by the local systems, with regards to the $ACID$ properties. Thus, we avoid problems such as lost updates, non-atomic behavior of transactions, etc. In this chapter we will not address the lower level, and we will concentrate on the polytransaction level. We discuss consistency problems due to concurrent execution of different transactions that belong to separate polytransactions. These transactions are executed under the control of local systems, with only limited global coordination.

We start by examining the notion of conflict for polytransactions. We assume that the IDS composed of D^3s, is correct, i.e., it does not include cycles in the dependency graph.

Definition 17.6.4 *Two transactions T_i and T_j are in conflict on a dependency descriptor D^3 if and only if they perform conflicting operations on data objects that belong to the source set S of D^3. Two operations are conflicting if at least one of them is a write operation.*

Definition 17.6.5 *Two polytransactions P_i and P_j are in conflict with respect to the* IDS *if and only if they contain transactions $T_i \in P_i$ and $T_j \in P_j$ that conflict on any D^3 in the* IDS.

Figure 17.7 shows a dependency graph with source objects a, b and targets c, d interconnected through descriptors D_1^3 and D_2^3. If the source object a is updated by polytransaction P_a, and the source object b is updated by polytransaction P_b, then P_a and P_b are examples of conflicting polytransactions.

Since uncontrolled updates from conflicting concurrent polytransactions may lead to distortion of data and violation of mutual consistency, we need to control the execution of concurrent polytransactions so that consistency of the data is preserved. One

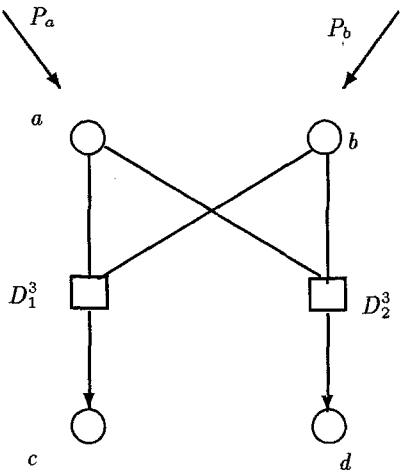


Figure 17.7: A dependency graph and conflicting polytransactions P_a, P_b

way of achieving this goal is to use serializability for polytransactions, which is analogous to global serializability. However, in the *IDS*, the problem of *indirect conflicts* caused by local transactions serialized between polytransactions and possibly changing their serialization order [218] does not arise. This is because we either completely disallow external (local) updates or limit them to stay within the zone of consistency, thus assuring that they are insignificant.

The correctness of concurrent execution of polytransactions relies not only on the execution order and precedence of their transactions, but also on the specification of the D^3s. On one hand, we have a more precise specification of what is considered consistent, so we can exploit semantic information to preserve consistency. On the other hand, we may have different actions and restoration procedures and various timing intervals during which a transaction may run, which impose additional restrictions.

Since we know the code of the conflicting transactions we can use this information to customize serializability for polytransactions. If a descriptor D^3 contains only a single restoration procedure, then two conflicting polytransactions will execute the same transaction twice. Then under certain conditions we can demonstrate that the execution of a non serializable schedule, may still be correct, regardless of the serialization order. The basic assumption is the following: If the calculation of the new value of the target object d (Figure 17.7), is a function of both sources a and b, and each transaction reads the same latest version of the sources when it starts executing, then the two conflicting transactions will execute the same code with identical input (a, b) and produce the same output d. The order of their operations may violate serializability but the final value of the target object will be the same regardless of their relative order. If an edge appears in the serialization graph, due to events of this type, that introduces a cycle, we can safely remove this edge from the serialization graph, eliminating the cycle, thus making the schedule correct.

In this case, by examining the specification of the D^3s we may allow some relaxation of serializability. This, in turn, may lead to the reduction of concurrency control overhead involved in processing of polytransactions. A more conservative approach

where serializability is preserved can be found in [207]. The authors introduce two concurrency control mechanisms for concurrent execution of polytransactions. The first is a deadlock free graph locking mechanism and the second is a variant of multiversion timestamps with rollback, that never rejects operations arriving out of timestamp order. However, this conservative approach assumes D^3s without temporal predicates. Concurrent execution of polytransactions including temporal predicates is examined next.

17.6.3 Polytransactions with Temporal Constraints

So far, we discussed the issue of concurrent execution of polytransactions triggered by D^3s that do not contain temporal predicates. In the absence of temporal constraints, it is sufficient to define the notion of "conflict" and then derive a protocol that resolves conflicts. Usually "preservation of precedence order" is an acceptable criterion of correctness. However, time itself implies an order, an absolute precedence between events (e.g. *read* and *write* operations). The temporal order of *read* and *write* operations on an object in a database, may not necessarily be the same as the precedence order imposed by the concurrency control mechanism. In this subsection we address the issue of correctness of concurrent execution of polytransactions under temporal constraints specified in the D^3s.

In the following discussion we assume synchronized site clocks and ordering of events as in [348]. We also assume that a *timestamp TS(T)* which indicates a real time value, is associated with a transaction T, similar to the *value date* by Litwin and Tirri [374].

Definition 17.6.6 *Transaction T_i precedes in time transaction T_j ($T_i \mapsto T_j$) if T_i has a timestamp smaller than the timestamp of T_j, i.e., $TS(t_i) < TS(t_j)$. This precedence order defines a temporal order between the transactions T_i and T_j.*

Definition 17.6.7 *Two transactions $T_i \in P_i$ and $T_j \in P_j$ are Temporally Serialized (TSR), if and only if, their serialization order coincides with their temporal order, i.e., if $T_i \rightarrow T_j$, then $T_i \mapsto T_j$, or if $T_j \rightarrow T_i$, then$T_j \mapsto T_i$.*

When we deal with temporal constraints, we have to expand our criterion of correctness in concurrent execution of polytransactions with temporal constraints as follows:

Definition 17.6.8 *A schedule of concurrent execution of polytransactions with temporal constraints is considered correct, if and only if every pair of conflicting transactions $T_i \in P_i$ and $T_j \in P_j$ is temporally serialized.*

It is obvious that the introduction of temporal constraints in the polytransactions modifies the definition of correctness. In particular, a schedule may be serializable, but still incorrect if temporal predicates are not satisfied. On the other hand, if a schedule obeys temporal predicates but is not serializable then it is not correct either. The problem introduced by the temporal constraints has been identified by researchers in the area of real time database systems, as a "trade-off with completeness, accuracy, consistency and currency" [463]. It should be noted that the existence of temporal constraints in polytransactions gives them characteristics of long-lived transactions, executing for prolonged periods of time, and potentially increasing the number of conflicts.

A variation of the Timestamp Order mechanism can be used to correctly serialize conflicting polytransactions. However, the timestamps we use in this mechanism reflect real time as mentioned earlier. The time of an update on a top object, identifies the timestamp TS, given to a polytransaction.

The basic idea is as follows: We order conflicting operations from transactions in timestamp order. If an operation comes out of order, the transaction is rejected, and resubmitted with a new timestamp. However, if the rejected transaction has a temporal constraint that triggered it, it will not be resubmitted. This can happen if transactions with temporal predicates carry information that is "older" than the conflicting transaction that has already accessed the target object. We do not want the temporal transaction to overwrite a value that has been written by a "younger" non-temporal transaction. This implies that the regular "recent" transactions have precedence over transactions triggered by temporal predicates. A temporal constraint is included in a D^3 specification to propagate the update to the target object, after some time period. In the meantime, if a more recent transaction updated the target object, then this update carries more recent information, and the rejected update should not be resubmitted. The same reasoning applies to the case of two conflicting transactions that both contain temporal predicates. This policy is similar to the Thomas-Write-Rule [49] which ignores write operations that attempt to place an obsolete value in the database. The above technique guarantees TSR between concurrent execution of polytransactions at the expense of rejection of transactions with temporal constraints.

The advantage of using IDS to manage interdependent data is that the periods of data unavailability can be controlled (or eliminated) by an appropriate specification of the temporal terms within the consistency predicates. However, the problems of temporal consistency of data needs further investigation.

17.7 Conclusion

This chapter addresses issues in managing interdependent data. We provided a brief overview of the specification of the dependency descriptors, and the interdatabase dependency schema. We also proposed a conceptual architecture of a system that can be used to manage interdependent data.

We explored the issue of correctness of specifications. It involved investigation of semantic information stored in the interdatabase dependency schema and potential conflicts that may arise due to the specification of the consistency and dependency predicates. Two correctness checks we proposed were a) avoiding multiple dependency descriptors directed to the same target data object and b) avoiding cycles that do not satisfy certain properties.

Then, we described the polytransaction mechanism that could be used to automatically enforce consistency of interdependent data according to the requirements specified in the interdatabase dependency schema. We also discussed issues concerning the consistency of interdependent data. We presented a classification of various states of consistency of a data object, and identified the events that lead to changes to its state of consistency. We showed that the consistency of interdependent data can be violated if uncontrolled updates are allowed outside of the polytransaction mechanism.

Finally, we investigated the concurrent execution of polytransactions. We discussed the information needed to reason about the correctness of schedules of concurrent polytransaction execution and identified cases where they can be enforced. We also presented a preliminary solution concerning the concurrent execution of polytransactions

with temporal specifications.

This chapter presents the results of our on-going research project on managing interdependent data at Bellcore. It is inspired by the data consistency requirements in industrial environments. Real examples of interdatabase dependencies specified as D^3s can be found in [514]. .

Chapter 18

FEMUS : A Federated Multilingual Database System

Martin Andersson*, Yann Dupont†, Stefano Spaccapietra‡, Kokou Yétongnon§ Markus Tresch¶, Haiyan Ye‖

18.1 Introduction

The increased availability of various databases in large corporations has created the need to federate the databases into loosely coupled collections of autonomous systems to allow controlled sharing of information and at the same time preserve the autonomy of each participant. Traditional distributed database (DDB) research has provided the earliest solutions to information sharing in distributed computing environments. The DDB approach, however, assumes that a single and integrated conceptual view of the databases must be provided to the users. Federated database (FDB) schema/system architecture design has partially benefited from this effort. For example, the design of both DDB and FDB systems includes functionalities such as: schema integration, query processing, transaction management. The DDB techniques used to provide these functionalities can be extended and applied to FDBS. The main difference between the two approaches are in the classes of users and the levels of component autonomy they support. Users of DDB systems access shared data only through the single conceptual schema of the DDB. This facilitates the enforcement of integrity constraints attached to the DDB, in much the same way (from user's point of view) as centralized DB systems. FDB systems, on the other hand, generally support two classes of users: federation users manipulate shared distributed information through one or more federated schemas; and local users, to whom the federation is transparent, access local data only. Preservation of local access to, and control over, the local database is essential for the support of component autonomy, the most salient feature of FDB systems.

Typically, federations are built upon heterogeneous systems and may include databases which support different data models: relational, CODASYL, and object-oriented

*EPF Lausanne, Department of Computer Science, Databases Laboratory, CH - 1015 Lausanne, Switzerland.

†EPF Lausanne, Department of Computer Science, Databases Laboratory, CH - 1015 Lausanne, Switzerland.

‡EPF Lausanne, Department of Computer Science, Databases Laboratory, CH - 1015 Lausanne, Switzerland.

§EPF Lausanne, Department of Computer Science, Databases Laboratory, CH - 1015 Lausanne, Switzerland.

¶Department of Computer Science, University of Ulm, Germany.

‖ETH Zürich, Department of Computer Science, Information Systems – Databases, CH - 8092 Zürich, Switzerland.

data models. One of the important issues that need to be addressed is therefore the data model translation problem. Usually, to avoid a proliferation of translators, the use of a common data model within the distributed/federated system is encouraged. Semantic or object- oriented data models represent the best candidates for this role. Translations between semantic, object-oriented and relational models have largely been dealt with in the past. They have been commonly developed for two major cases: 1) to support a database design process where the database is first described in a conceptual schema which is subsequently converted into a logical schema [77]; and 2) to support the design and operation of traditional DDB system. In the former case the translations are one-way processes which produce a single logical target schema from the source conceptual schema. As user queries are expressed and performed directly on the logical schema, there is no need for further translations. In the latter case, interoperability among data models is needed to move data in and out from the DDB. However, interoperability is usually not required among data manipulation languages, as most of the proposed DDB systems are monolingual: they support only one DML, the one associated to the common data model in use ([144] is an exception). The distinguishing feature of federated environments is that they have to be multilingual, because of the autonomy goal [318]. Therefore, the support of interoperability among DMLs becomes in fact the primary criterion for the specification of data model translations. Besides the translation problem, interoperability in the federated approach necessitates a mechanism for making data available to the federation and for ensuring access to the data. This mechanism is usually called an import/export facility. Data exchanges require first that description of accessible (exported) data be available to all users of the federation or designated set of partners, and second that the information be presented to them in their local model (in a model they understand). Partners may then select (import) data of interest to them and include/integrate them into their view of the FDB. Integration facilities are the essential feature in building this unified view from the different pieces of imported data.

This paper presents the approach of an ongoing research project, FEMUS (FEderated MUltilingual System), jointly developed by the database research groups at Swiss Institutes of Technology (EPFL and ETHZ). The primary goal of the project is to provide a framework for investigating semantic related issues of interoperable database architectures. The focus of this paper is particularly on federated database construction issues. The FEMUS project itself is not discussed in detail. Instead, we examine two of its most important aspects: data model translation and integration issues. The former is dealt with in section 3, while the latter is the topic of section 4. Section 2 presents the architecture of the FEMUS prototype and the modeling paradigms on which interoperability is being experimented. Section 5 discusses semantics issues related to schema negotiation and data/metadata exchange. Section 6 reviews implementation aspects related to the export/import mechanism. Section 7 concludes the paper.

18.2 FEMUS

The aim of the FEMUS project is to set up a framework for building a federated multilingual database system. By *federated* it is meant that the global system provides the functionalities to include, as components, different heterogeneous database systems cooperating together, and that the major goals are preserving site autonomy and supporting maximum flexibility in the interoperability mechanisms. By *multilingual* it is meant that an equally important goal is to build a system which can be accessed

by different users through the local interface (data model and manipulation language) they are used to.

The initial version of the project includes two different database approaches:

- **ERC+** [447], an object-based extension of the entity- relationship model, including the specification of an ER algebra [446] and calculus. The fundamental concepts of the ERC+ model are entities, relationships, and complex objects.
- **COCOON**, an object-oriented data model with an object algebra that was developed based on the nested relational algebra [501, 500, 494, 493]. The basic concepts are objects and functions. COCOON, like DUAL [451], promotes separation between the type and class hierarchies. This separation introduces original problems that need to be addressed by the model translation processes. The two approaches are briefly presented in the subsections hereinafter. Studying interoperability between these two approaches is of particular interest, as they represent, as stated above, two very important and widely used families of models possible in a federation.

In [515], five schema levels were proposed as a reference architecture for federated database systems, separating local pre- existing schemas, component schemas (common model counterpart of local schemas), export schemas, federated schemas (FS), and external schemas defined over federated schemas. The federated schemas hold a global dictionary with additional information about fragmentation and allocation of distributed objects. Thus, to the users of FS, both fragmentation and allocation are fully transparent. The FDBS is responsible for transforming the global queries and updates into statements for the component databases. FEMUS basically adheres to this proposal, while adding one more level to allow users to integrate various import schemas (derived from either export schemas or from federated schemas) to form their own federation. In this way ad-hoc federations may be built, enhancing the flexibility of the system from user's perspective. Figure 1 shows FEMUS six levels generic schema architecture. It includes the concept of median schema, as suggested in [496], to emphasize domain-specific federations. It also shows that export schemas may be derived as views over component schemas or directly from local views through data model translation. Finally, import schemas are derivable from any of the schemas available at the federated level: export, federated or median schemas. A simplified architecture (Figure 2) is currently being implemented for the first exploratory prototype. It mainly includes two process types, *Translators (mappers)* and *Integrators*. The mapping processes translate schemas, integrity constraints, and language elements from one data model/language to the other (ERC+, COCOON); the integration processes combine schema and instances from the two components.

18.2.1 The ERC+ Approach

A complete definition of ERC+, and a discussion of its features, may be found elsewhere [447]. Here we briefly recall the main features of ERC+:

- *entity types* bear one or more attributes. As attributes may in turn, iteratively, be composed of other attributes, the structure of an object type may be regarded as an unlimited attribute tree;
- *relationship types* may connect any number of participating entity types. As entity types, they may have attributes. They are said to be cyclic if the same entity type participates more than once in the relationship type;

- a *role* name is associated to each participation of an entity type in a relationship type. The participation is characterized by its minimum and maximum cardinalities;
- *attributes* may be either atomic (non decomposable) or complex, i.e. decomposable into a set of component attributes, which may themselves be either atomic or complex. An attribute is also characterized by its minimum and maximum cardinalities (mandatory/optional, monovalued/ multivalued). Attributes may be valued in a multiset (i.e. not excluding duplicates);
- two *generalization* relationships are supported, the classical "is-a" and an additional "may-be-a" relationships. The former corresponds to the well-known generalization/specialization construct; the latter has the same semantics, but does not require an inclusion dependency between the subtype and the type and is therefore used to describe multi-instantiation at the schema level;
- an *object identity* is associated to entities.

Figure 3 shows a sample ERC+ diagram. A single continuous line represents a 1:1 link (mandatory monovalued), a single dotted line represents a 0:1 link (optional monovalued), a double dotted line represents a 0:n link (optional multivalued), a double line (once dotted, once continuous) represents a 1:n link (mandatory multivalued). No generalizations appear in this particular example.

Two formal query languages, an algebra and an equivalent calculus, are associated with the ERC+ data model. The functionalities provided by the algebraic operators include: selection of entity type occurrences based on a given predicate, projection of entity type occurrences on a subset of its attributes, union of the populations of two entity types. Specific to ERC+ is the *reduction* operator, which allows the elimination of the values of an attribute which do not conform to a given predicate. Most important is the relationship-join (*r-join*, in short) operator. Let E1, E2, ..., En be the set of entity types linked by a relationship type R, the r-join of E1 with E2, ..., En via R builds a new entity type (and the corresponding population) whose schema includes the schema of E1 plus an additional complex attribute, named R, whose components are the schemas of R, E2, ..., En. In some sense, this operator groups into a single entity the information scattered over entities linked by a relationship. A *spe-join* operator allows to join entity types participating into a given generalization [538], thus providing for an explicit inheritance mechanism. Every ERC+ operation is objects preserving and creates a new derived entity type. The attributes, relationships, generalization links and population of the created type are derived from the operand types. Operations may thus be combined into expressions of arbitrary complexity.

18.2.2 The COCOON Approach

COCOON is an object-function model. Its basic constituents are objects, functions, types, classes and views. The following is an excerpt from [502]. Objects are instances of abstract types, specified by their interface operations. Functions are either retrieval functions or update methods. They are described by their name and signature. They may be multivalued. Types are described by their name and the set of functions that are applicable to their instances. A subtype hierarchy defines type inheritance relationships. Objects are instances of one or more types (multi- instantiation). Full static type-checking is supported. Classes are collection of objects (type extents). A subclass hierarchy defines class inclusion relationships. Objects are members of one

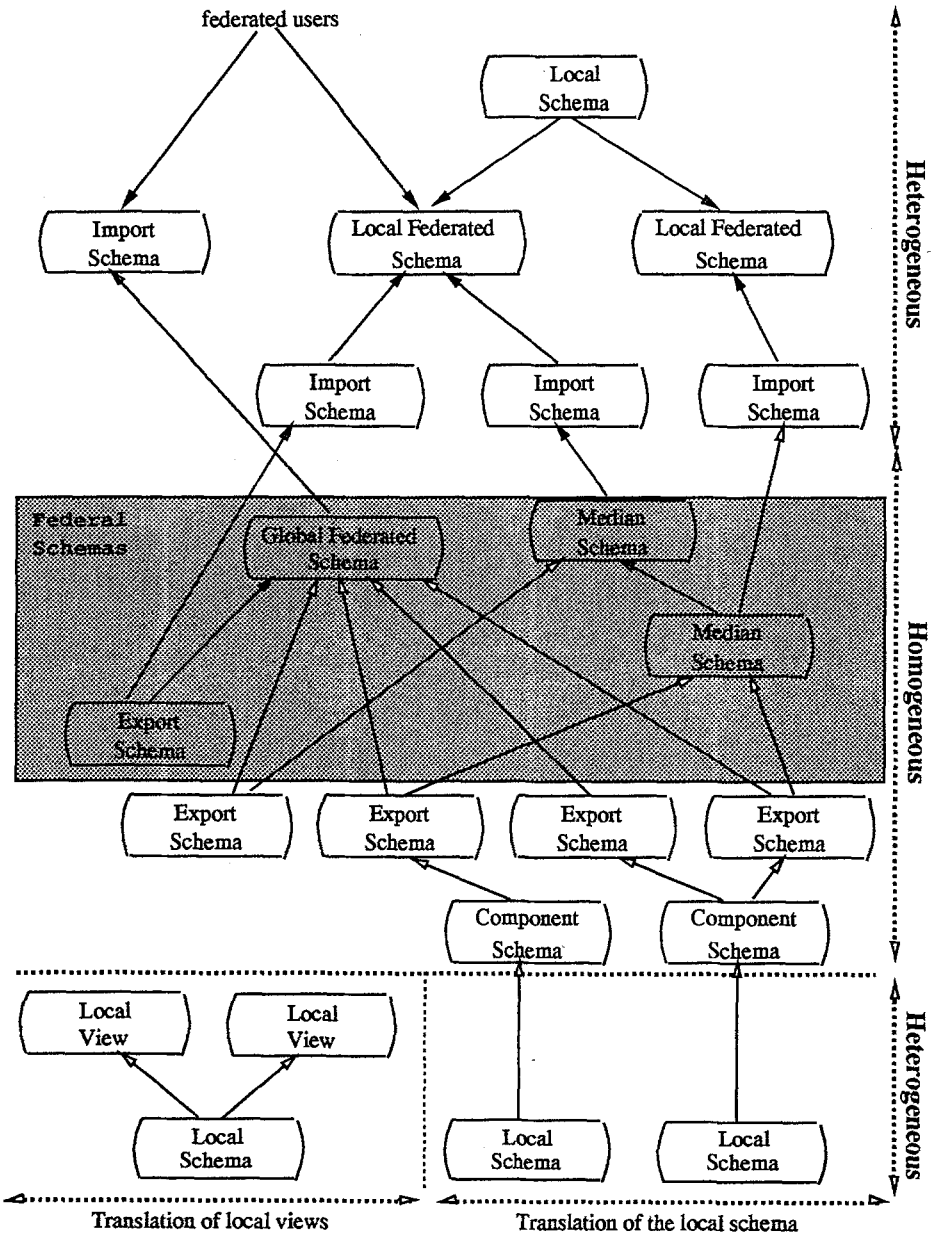


Figure 18.1: The generic schema architecture in FEMUS

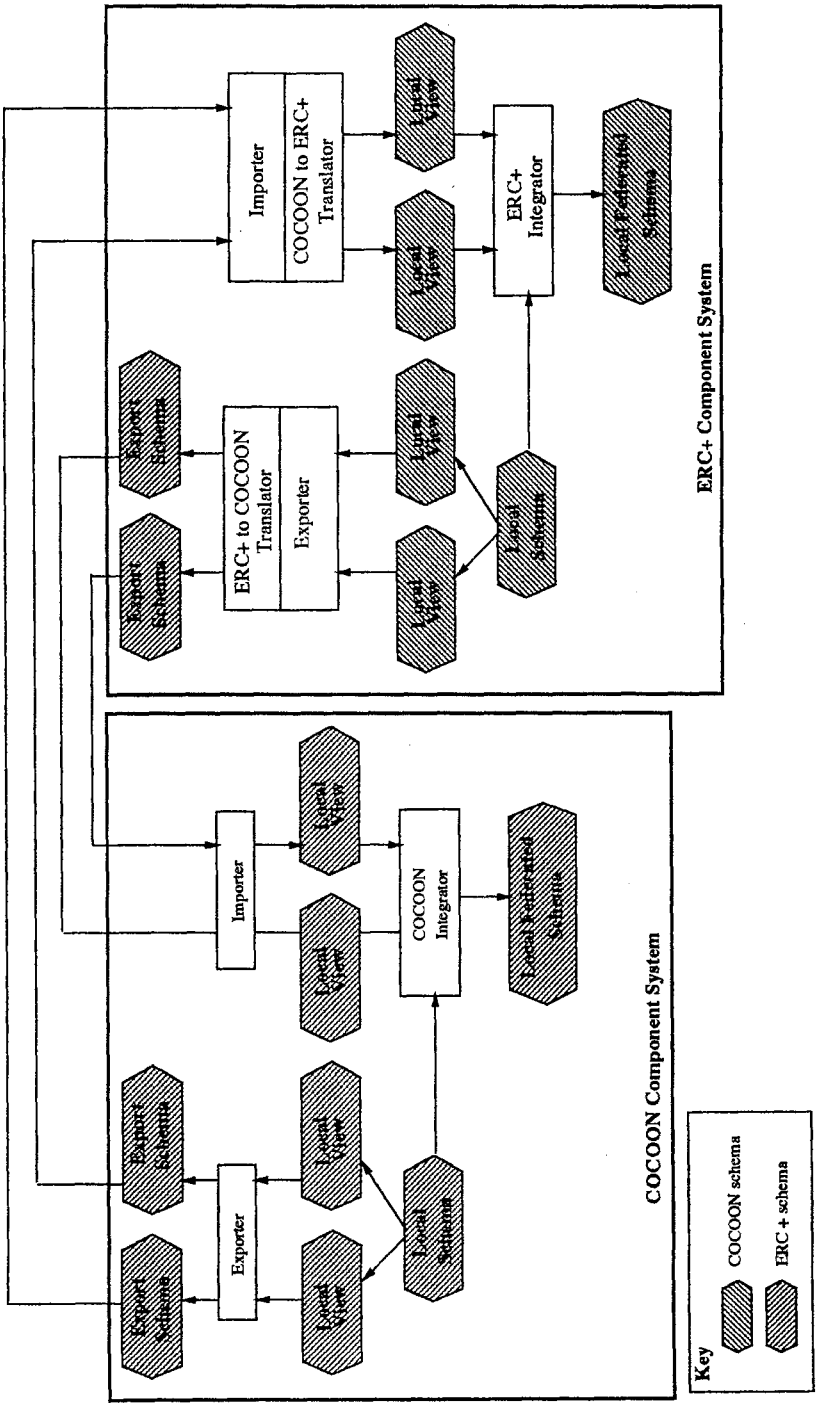


Figure 18.2: FEMUS prototype specific architecture

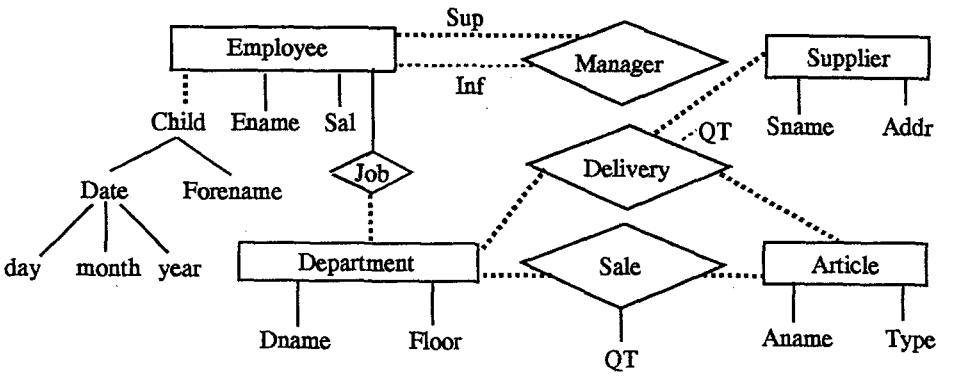


Figure 18.3: An example ERC+ schema

or more classes. For each object class, its member type is an associated type, and its extent denotes a set of objects of that type. Classes may be constrained by a predicate that must be satisfied by all members of the class.

A set-oriented query language, called COOL, similar to relational algebra, provides operators to build an output set of objects from input sets of objects. Query operators can be applied to extents of classes, set-valued function results, and query results. The algebra has object preserving semantics. Queries are also used as the view definition mechanism: they introduce new virtual classes, and define their extent. Views may be defined by basic COOL operators, over other views or by composite queries. COOL operators are: selection, projection, extend (allows the definition of new derived functions), set operators (union, difference, intersection), and generic update operators (update, insert, delete, add, remove, and set to assign return values to functions). Figure 4 shows a COCOON type diagram describing the same universe of discourse as the ERC+ diagram in Figure 3. Arrows with single arrow head (respectively, double arrow head) represent single- valued (respectively, set-valued) functions. A function and its inverse are linked by a straight line.

18.3 The Mapping Process

The translation process is well understood when converting from a conceptual data model to a target (logical) data model during the design of a database. In this context the main requirement is to minimize semantic loss that may occur when semantically rich modeling concepts of the source model are converted into less expressive constructs in the target model. These schema translations mainly focus on implementing data structures and access paths in the target model. By analogy to top down design methodologies, this conversion process is regarded as a "vertical translation". Since users queries during the operational phase of the database are expressed in the DML of the target model, there is generally no DML translation between the conceptual and logical models. On the contrary, cooperative DBS require bi-directional translation processes between any pair of component sites. These translations must take into consideration the reversibility and equivalence issues between two different data models of similar semantic expression power. Thus, the translations are called "horizontal and

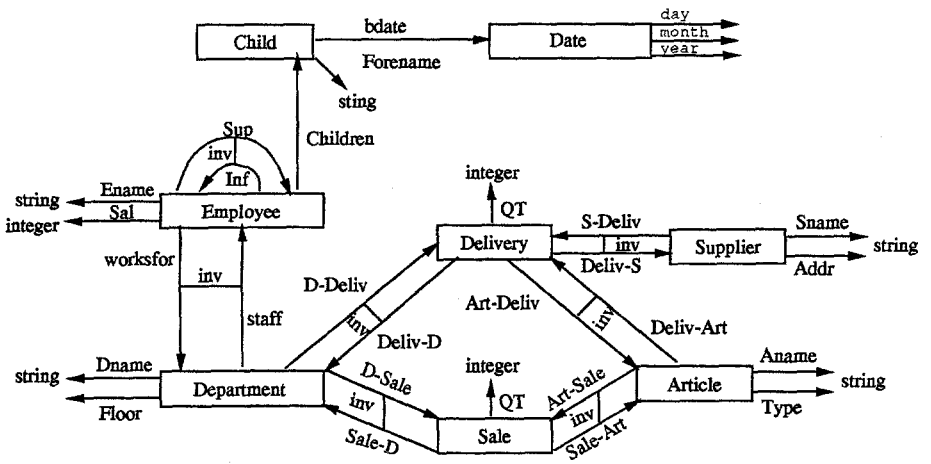


Figure 18.4: The COCOON equivalent to the example ERC+ schema

bi-directional translations". Moreover, interoperability implies that data manipulation requests will generally be issued by each component of the federation towards the other components, this also calling for a horizontal translation between the various DML in the federation. FEMUS focuses on the horizontal translations between object-oriented and semantic data models. The following subsections discuss mappings for the ERC+/COCOON case. The discussion, however, may be easily adapted to other similar models.

As stated, interoperable mappings need to be defined in terms of both static (schemas) and dynamic (operations) aspects. Of course, these two mappings are strongly related to each other, as illustrated in Figure 5. Let S1, S2 be schemas in two different data models, say model-1 and model-2. Let O1, O2 denote operations of the algebras of model-1, model-2 respectively. An operation is either a single operator or an expression. The result of an operation is both in terms of new schema elements and their instances. Let O1(S1) denote the result of applying O1 to S1 (similarly for O2(S2)). Let T be a translation from model-1 into model-2 such that T transforms S1 into S2 and the underlying database-1 into database-2. Let TOP be a translation from model-1 into model-2 such that it transforms an operation O1 into an operation O2. Then the following should hold:

$$T(S1) = S2 \wedge TOP(O1) = O2 \Rightarrow T(O1(S1)) = O2(S2)$$

18.3.1 Mapping an ERC+ Schema to COCOON

The major differences between the two modeling approaches are:

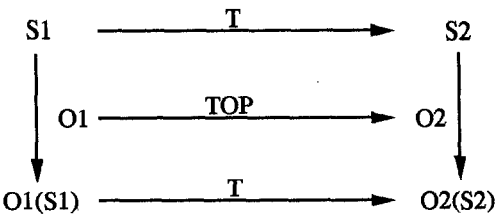


Figure 18.5: Mappings consistency

- ERC+ distinguishes between attributes and relationships, while COCOON treats them uniformly as functions,
- ERC+ supports the representation of a complex object as a single entity while COCOON decomposes it into a set of related objects.

Accordingly, the following schema translation rules may be stated.

a) Attributes

A simple attribute is mapped into a COCOON function whose domain is the COCOON object type corresponding to the attribute owner, and whose range is the COCOON primitive type corresponding to the ERC+ value domain of the attribute. The function is single-valued or set-valued depending on whether the attribute is monovalued or multivalued.

Examples:

- Ename (Figure 3) results in the Ename function: Employee → string (Figure 4).
- day (Figure 3) results in the day function: Date → integer (Figure 4).

A complex attribute is mapped into a new object type and its associated class. A function is added to link the owner object type to the new type.

Examples:

- Child (Figure 3) results in the Child object type. A set-valued function, named children in Figure 4, is added to link Employee to Child.
- Date (Figure 3) results in the Date object type. A single-valued function, named bdate, is added to link Child to Date.

b) Entity types

An entity type is mapped into an object type and its associated class. Functions are attached to the type according to attributes and relationships translation rules.

c) Relationship types

1. Binary relationship types without attributes A binary relationship type with no attribute is mapped into two inverse functions between the object types corresponding to the entity types participating into the relationship

type. These functions are single- or set-valued depending on the cardinalities of the corresponding roles.

Example:

- Job (Figure 3) results in the two inverse functions worksfor and staff. Worksfor is single-valued as the role of Employee in Job is monovalued. Staff is set-valued as the role of Department in Job is multivalued.

2. Binary relationship types with attributes and n-ary relationship types. These relationship types are mapped into a new object type and its associated class. The attributes, if any, of the relationship type are mapped into functions according to rules in a). Each role of the relationship type is mapped into a pair of inverse functions between the new type and the type corresponding to the entity type attached to the role.

   Examples:

   - Sale (Figure 3) results in the object type Sale (and its class), with the attached QT function: Sale integer, and two pairs of inverse functions, D-Sale and Sale-D, linking Sale with Department, and Art-Sale and Sale-Art linking Sale with Article.
   - Delivery (Figure 3) similarly results in one new object type, Delivery, and three pairs of inverse functions.

d) Generalization links

An is-a link between two entity types is mapped into an is-a link between the object types and classes corresponding to these entity types. May-be-a links are not mapped, as there is no such concept in COCOON at the moment.

18.3.2 Mapping a COCOON Schema to ERC+

The COCOON to ERC+ mapping rises problems typical of reverse engineering situations, in which a more decomposed representation has to be translated into a more compact one. A classical example is the relational to ER translation. A straight algorithm may be used to transform a COCOON schema into an ERC+ schema. The algorithm is driven by class and function definitions. A function from an object class to a primitive type is mapped into a simple attribute of the ERC+ element corresponding to the class. A pair of inverse functions between object classes is translated into a binary relationship between the two entity types corresponding to the related classes. A function between object classes which has no inverse is mapped as an attribute of the ERC+ element corresponding to the class where the function originates. COCOON classes which have not been translated as attributes using the above rules are mapped into ERC+ entity types, whose structure (the set of attributes) is derived from the membership type of the class. An is-a link between two object classes is mapped as an is-a link between the corresponding entity types.

The algorithm generates a valid ERC+ schema, which includes only binary relationship types with no attributes. Applied to the schema in Figure 4, it does not produce the schema in Figure 3. The consequence is that applications using the ERC+ schema will be overloaded in terms of data manipulations needed to get the desired result. For instance, assume we want a list of departments showing their deliveries in terms of quantity, supplier and article. To produce the list using the schema in Figure

3, a single r-join operation is needed. To do the same using the schema resulting from the translation of the schema in Figure 4, it will need three r-join operations (because Delivery is now an entity type linked by three binary relationship types to Department, Supplier and Article). There is no automatic solution to avoid such overloading, unless the COCOON schema is enriched with information on dependencies among its components. It is known that, if certain dependencies hold, a n-ary relationship type may be decomposed into relationship types of degree less than n. Reversing this reasoning, the translation algorithm can infer from the known dependencies if a set of pairs of inverse functions defined on the same object class can be mapped as a n-ary relationship type also representing that object class.

18.3.3 Operators Mapping

Both ERC+ and COCOON models incorporate an algebra as DML. In this section we compare the characteristics features of the two algebraic query languages in order to define a mapping (translation) of operators from one model to the other. The two algebra have in common several operations (such as selection and projection, for instance) with well known semantics. In addition, they contain operations which have no direct corresponding counterpart in the other model. Thus, an operator in one model may correspond to an algebraic expression, rather than a single operator, in the other model. The requirement for correctness of the translation is the consistency of the mappings as discussed at the beginning of section 3. The mappings relationships illustrated in Figure 5 can also be used to determine TOP. Knowing T, we can use it to translate S1 and O1(S1) into the second model, getting S2 and the result of O2(S2). We can then check how, in the second model, S2 can be mapped onto O2(S2). If there is no direct operator which does it, then we have to find the adequate expression to build the expected known result. Hereinafter we discuss the operators mapping based on ERC+ operators.

Select Operator

Select operators in ERC+ and COCOON have the same semantics. They preserve the schema of their input operands. Their main difference is in the type of predicate they allow. ERC+ predicates may contain quantifiers (over set-valued attributes), whereas COCOON selections may contain nested expressions and set comparison operators.

Consider the following query:

"select all employees who earn, at least, one salary greater than 6000".

Below we give ERC+ and COCOON algebraic expressions for the queries and derive the rules for translating the operators.

ERC+ query

The selection operator, noted σ in ERC+, creates a new entity type which contains the entities (objects) that satisfy the selection predicate. The ERC+ algebraic expression corresponding to the above query is given by:

$$E1 = \sigma[\exists_{s \in Sal}(s > 6000)]Employee$$

where the existential quantifier ($\exists$) is extended to apply to multisets.

COCOON query

In the following EmployeeC denotes the class of objects belonging to the type Employee. The above query is written in COCOON as:

$$C1 = select[select[s > 6000](s : Sal) \neq \emptyset](EmployeeC)$$

The nested select operation implements the selection predicate "at least one salary greater than 6000" by first retrieving the set of salaries over 6000 of an employee, and then checking whether the retrieved set is empty or not.

If the above query is modified to "select all employees who have all their salaries greater than 6000", then the corresponding ERC+ algebraic expression is:

$$E2 = \sigma[\forall_{s \in Sal}(s > 6000)]Employee.$$

The universal quantifier included in the predicate can be expressed in different ways in COCOON by using nested select operations and set comparison operations. The resulting queries are:

$$C2 = select[select[s > 6000](s : Sal) = Sal](EmployeeC).$$
$$C2' = select[select[NOT(s > 6000)](s : Sal) = \emptyset](EmployeeC).$$

The second query is obtained from the first one by the application of the standard transformation of the universal quantifier into the existential one: $\forall x(P) \Leftrightarrow \neg\exists x(\neg P)$.

In summary the following rules can be derived for converting selection operations between ERC+ and COCOON. P(a) is a predicate on variable a, and a is a variable on a multivalued attribute A.

Equivalence rules between ERC+ and COCOON selection predicates:

$$\exists_{a \in A}(P(a)) \Leftrightarrow select[P(a)](a.A) \neq \emptyset.$$
$$\forall_{a \in A}(P(a)) \Leftrightarrow select[NOTP(a)](a.A) = \emptyset.$$

Project Operator

1. Projection over a simple attribute
   Project operators in ERC+ and in COCOON have equivalent behavior when they are applied to simple attributes. In essence, they are used to drop one or more simple attributes from the schema of an entity or object type.

   The translation rule in this case is trivial. For instance:

ERC+		COCOON
π [Ename] Employee	$\Leftrightarrow$	project [Ename] EmployeeC

2. Projection over complex attributes
   As discussed in section 3.1 above, the translation of ERC+ complex attributes leads to the decomposition of an object over several objects. Therefore the translation of an ERC+ projection over a component of a complex attribute requires that the corresponding COCOON object type be augmented with a derived function which links it directly to the desired attribute. Derivation is through composition of COCOON functions. For instance, a projection over the attribute Forename of the complex attribute Child (Figure 3) is mapped as follows:

ERC+		COCOON
π [Child•Forname] Employee	$\Leftrightarrow$	project [forename] (extend [forename = Forename(Child)])EmployeeC

Product Operator

A product operator in ERC+ is useful for linking two entity types which are not linked by any relationship type. In the case where a relationship exists between the two entity types, it is simply ignored by the product operator. The operation extends every occurrence of its first operand with the set of all the occurrences of the second operand

(represented in the schema of the resulting entity type as a multivalued complex attribute whose components are the attributes of the second operand). For example, a product will be applied to the entity types Employee and Supplier of Figure 3 as a first step to check whether there are employees having the same name as a supplier.

The result of this product operation is an entity type, say ES, with all the original attributes from Employee plus an additional complex attribute supplier, which is composed of Sname and Addr. The check is then done through a selection on the result of the product. The ERC+ selection predicate is:

$$\exists_{n \in supplier \bullet Sname}(n = Ename).$$

The result of the product operation can be achieved in COCOON by extending the object type Employee with a new function to create the link to the object type Supplier.

Relationship-Join (R-Join)

The R-join transforms a relationship type into a complex attribute structure with respect to one of its participating entity types. Consider the two entity types Department and Article, and the relationship type Sale in Figure 3. The result of applying R-join to Department through the relationship Sale is an entity type with all the attributes of Department (Dname, Floor) and an additional complex and multivalued attribute named Sale, whose attributes are Qt and Article, the latter being a complex attribute with Aname and Type as components.

As the product operator, the relationship-join is used to merge information from various entity types into a single entity type. While product merges systematically each entity of one type with all entities of the other type, R-join uses as merging criteria the fact that the entities forming one occurrence in the result are linked by a relationship of the given type.

To some extent, we could state that R-join builds more complex entity types from the existing ones. COCOON objects are not complex objects. Because of decomposition rules of the schema mapping, the result of an R-join, translated in COCOON, would generate the same objects and functions as those already there. Therefore, the translation of an R-join is the identity operation.

18.4 The Integration Process

Database integration is the second key feature in building integrated database services in an interoperable environment. The autonomy goal inherent to FDBS requires that new integration techniques are developed to cope with all possible discrepancies among component databases, without altering them, while providing for maximum integratability. Although schema integration has already been investigated for a long time [38], existing methodologies lack the power to integrate schemas showing structural conflicts, i.e. to solve situations where, for instance, the same real world object is represented as an attribute in one schema and as an object in another schema [536]. These methodologies need a conforming step, prior to integration, such that all structural conflicts are removed from the input schemas. The conforming process relies on schema modification, a consequence which contradicts the primary FDBS requirement: to ensure continuation of local usage of data without any user visible impact due to the new federation services. Moreover, current methodologies do not really handle data model heterogeneity. They only propose to translate every input schema in some common

model within a pre-integration step. This is consistent with the usual approach to federated or distributed heterogeneous databases, but might become a bottleneck in more flexible architectures in which multiple federations, although defined on the same component databases, are not necessarily built upon the same data model.

Integration in FDBS is a bottom-up task, that must combine databases that may have existed for a long time. These databases may already have a considerable amount of data. Thus, integration in federated databases must cover both tasks, the integration of schemas and of existing objects. This is usually called database integration, in contrast with view integration where only schemas with no associated extensions are integrated. A relatively small amount of work concerns database integration. One recent exception is the Pegasus project [9, 335]. Kent emphasized the separation between real world entity objects and their. database counterpart, called proxy objects [315]. The latter ones represent entities in different component databases. Object integration must deal with the fact that due to historical evolution of databases, the same real world entity may be stored as different database approximations in different databases (the proxy objects). Thus, the FDB administrator must also specify what proxy objects of what component databases represent the same real world object and under what circumstances.

FDBS integration requirements also differ from DDB requirements. In DDB systems, integration is performed once, taking as input the schemas of existing databases and producing as output the global schema of the DDB. As there is no mandate to preserve site autonomy, local pre-existing databases may be modified to make integration easier or to redefine data allocation. For instance, if the same data appear in more than one local database, the DDB administrator may just keep one copy and have the other ones deleted to avoid the data replication problem. In FDBS, integration may be performed at *different levels*, depending on the organization's approach to FDBS architecture (cf. Figure 1b). As in DDBS, it might be the task of the FDB administrator (for each FDB being built). It might as well be a process performed by end users, if they are given the ability to import data from various sources (whether directly from local databases or through a FDB) to build their own, single user FDB.

Integration is a two-fold process. First, syntactic as well as semantic relationships among elements in the input schemas have to be precisely identified. This is the investigation phase. A first stream of research developed methods and tools to assist the database administrator in identifying interschema relationships [516]. In the second phase, the integration phase, related elements have to be "integrated". Various techniques have been proposed for this purpose. The first approach is manual integration, where the DBA is provided with a schema restructuring language. The latter allows the DBA to direct the integration process towards the step by step construction of the integrated schema the DBA is aiming at. The integrated schema is seen as a superview defined over input schemas [419]. This approach is relatively simple to implement, as the functionality supported by the system is limited to executing the restructuring operations. It is, however, of poor user friendliness and badly suited for the non expert users which might be allowed to build federations in a flexible FDBS. To cope with these inconveniences, more powerful assertional approaches have been proposed. They are intended to automatically perform integration from input correspondence assertions, which instruct the integrator tool on which interschema relationships exist. Assertional techniques provide a higher level of service to their users (users just have to care about existing correspondences, not about how corresponding items may be merged to form the integrated schema). They build the integrated schema (and the associated mappings to/from source schemas), using established integration rules

which allow to solve all types of conflicts supported by the tool. Finally, rather than building a new integrated schema, a third approach performs integration by extending input schemas (and databases) with the additional interschema descriptions. These either record correspondence information, or add new constructs to relate one element in a schema to another element in another schema [502].

FEMUS currently focuses on the integration phase. We discarded the manual superview approach as contrary to FDBS flexibility goal. As for the two other approaches - the integration assertional technique and the augmentation technique described just above - no definite assessment has evaluated and compared their pros and cons. Intuitively, building an integrated schema seems preferable if there is a heavy overlap among component schemas, with many component elements resulting in a single element after the merging. On the other hand, if the component databases have complementary content, augmenting existing schemas with interschema references is a simpler process. To know more about the comparison we have decided to investigate and experiment both techniques in parallel. They are hereinafter separately presented.

To show differences and/or similarities between these two integration techniques, we illustrate their usage on the same very simple example (Figure 6) modeled according to object-oriented notations. The example assumes a FDB environment with two databases, DB1 and DB2. DB1 contains information about cars (the class Car, with three attributes: the car's registration number and color, and the person identification number of the car's owner), whereas DB2 holds persons (the class Person, with three attributes: the person's identification number and name, and the registration numbers of the cars (s)he owns). Since cars are owned by persons, and persons own cars, there is some sort of inter- database correspondence. We show below how this correspondence is specified using the assertion-driven integration, and how it is done through the augmentation approach.

DB1: Class Car tuple <reg#: integer, color: string, owener: integer>

DB2: Class Person tuple <pin: integer, name: string, cars: setof integer>

Figure 18.6: The example integration case

As stated, the definitions of the two classes do not bear any indication of their interrelationship. It is assumed that the FDBA has the external knowledge about the semantics of the attributes and the object classes being described. (S)he is therefore responsible for directing the integration tool through explicitation of that knowledge. In a repository environment, the definitions would be complemented with some natural language descriptive information. For instance, the owner attribute could be described as: this attribute holds the person identification number of the person who owns the car. An investigation tool could then guess that there might be some relationship between the two classes, due to the fact that the term "person" appears in both de-

scriptions. Nevertheless, the FDBA would have to be prompted anyway to confirm or correct the proposed correspondence. Note that the guess made by the tool is only in terms of structural correspondence, unless both databases are accessed to check the corresponding populations.

18.4.1 Assertion-driven Integration

The EPFL team has proposed a new assertional method to integrate heterogeneous source schemas [537]. The method solves structural, semantic and descriptive conflicts without changing the input schemas. It is also able to directly integrate heterogeneous schemas, without going through a preliminary translation step. To this purpose, interschema correspondence assertions, stated by the DBA, may relate an element whatsoever in one schema to an element of any type in another schema. An assertion defines the relationships between the element "types" (structural description) and the related "classes" (sets of associated instances). It also includes the necessary definition of an object mapping at the instance level, which provides for the integration of the related databases. Integration rules are generically defined on an abstract "generic data model" (which basically supports objects, value attributes and reference attributes), but their implementation in the actual integration of two input schemas is tailorable to the specific features of the input data model.

Considering the example in Figure 6, and assuming that the two schemas describe exactly the same sets of cars and persons (for every car seen by DB1, its owner is seen by DB2, and vice versa), the interrelationship between the two schema would be first described through two element correspondence assertions:

Car ≡ cars	with corresponding attributes	reg# = cars
owner ≡ Person	with corresponding attributes	owner = pin

The first assertion states that the set of cars described by the object type Car in DB1 (the real world state of Car) is the same (≡) as the one described by the cars attributes of the Person object type in DB2. The set equality is between sets of real world objects, and holds independently of their representation in the two schemas. The "with corresponding attributes" clause describes the structural relationship between the two representations. In this case there is only one information about cars which is represented in both databases: the car's registration number. This is hold by the reg# attribute in DB1 and by the cars attribute itself in DB2. Hence the stated equality of the two attributes. The same considerations apply to the interschema relationship between DB1 car owners and DB2 persons.

The fact that associated elements in one schema (Car and owner in DB1) are equivalent to associated elements in another schema (cars and Person in DB2) does not necessarily imply that the association has the same semantics in both schemas. It could be the case that DB1 talks about ownerships, and DB2 talks about cars being driven by persons, while still referring to the same real world state for cars and persons. Therefore, one assertion remains to be stated: a path correspondence assertion, making explicit that the link between cars and owners in DB1 (noted Car—owner) has the same semantics as the link between persons and car in DB2 (noted Person—cars). Links are bi-directional: Person—cars and cars—Person denote the same DB2 link. The path correspondence assertion for the example simply is:

Car—owner ≡ cars—Person

The integrator tool, with the two schemas and these three assertions as input, will generate the following integrated schema:

IDB: **Class** Car **tuple** <reg#: integer, color: string, owner: **reference** Person>
Class Person **tuple** <pin: integer, name: string, cars **setof: reference** Car>

The corresponding diagram may be drawn as follows, with labeled arrows representing reference attributes:

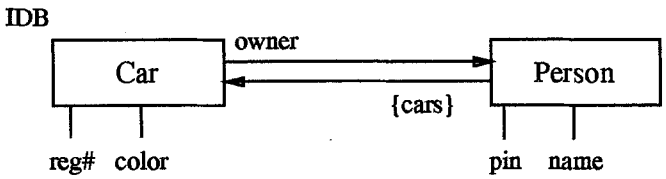


Figure 18.7: The integrated schema

IDB goes with the mapping information which states that the Car class is to be found in DB1, the Person class is to be found on DB2, and the references in between have to be evaluated through the matching criteria Car•reg# = Person•cars and Car•owner = Person•pin. Mapping information supports transformation of user queries against IDB into queries on the underlying DB1 and DB2 databases.

18.4.2 Integration Through Augmentation

The ETHZ team has developed an approach to provide a flexible way to specify the correspondence between existing objects of different databases. This is achieved by defining (global) object identity in terms of algebraic (extend) views [502]. We thereby make use of COCOON's view definition facility [500] that was extended such that it can span over multiple databases. It includes mechanisms for linking objects across systems and to deal with semantic conflicts. The necessary view definition method is to extend the local schema by elements of the schema of another system. Consider the example in Figure 6. The DBA will start the integration process by defining a view "Cars" as an extension of Car, with an additional function owned-by that returns for each car-object the person-object of the other database, that owns that car. This definition is stated as follows:

define view Cars **as extend**
[owned-by:= **select** [reg#(c) ∈ cars(p)](p:Person)](c:Car).

The selection predicate in the view definition materializes DBA's knowledge of the fact that the attribute cars of class Person specifies registration numbers of cars, which are recorded as values of the reg# attribute of Car in the other database. Knowledge that, similarly, the attribute owner of Car specifies a pin of a person in the other database, leads to similar extension of the Person type in DB2:

define view Persons **as extend** [owns:=
select [pin(p) = owner(c)](c:Car)](p:Person).

The additional function owns returns the cars owned by each person. It acts as the inverse of owned-by in DB1. Figure 8 below shows the schema of both databases, extended with views having functions that lead from one database to the other.

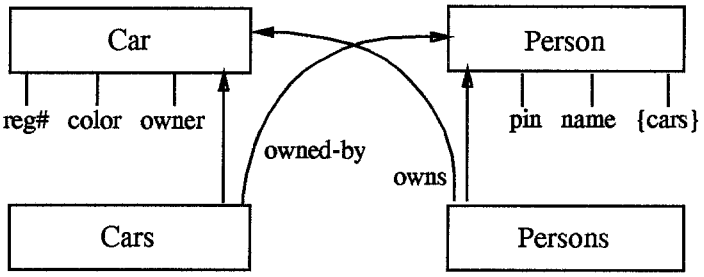


Figure 18.8: The augmented schemas

The advantage of this approach is that it uses almost exclusively the expressive power of a query language, together with a view definition facility. The only needed extension is a global object identity predicate. This is necessary, since, due to demand of local autonomy, we are not allowed to make any assumption on the local internal object-identities. Consequently, no change is required in the implementation of local database systems.

While from a conceptual point of view this approach looks simple and clean, more effort remains to be spent on the implementation issues. Particularly, an efficient, but overridable, implementation of the identity test seems to require further investigations. Till now, comparing object identities is typically hardcoded into the implementation of local object based management systems. Efficient support by indexes or some other form of replicated information is needed.

18.5 Negotiation

In a federated system, it is the responsibility of the local DBAs to decide and define which parts of their local data are available to external users (users from other sites). The decision process relies on negotiation with the other DBAs, for which specific tools may be developed within the federated system [245]. Negotiation needs an understanding of the semantics of data. This is usually supported by information attached to data descriptions and stored in a repository. The repository manager provides browsing capabilities and facilities to support dialogue and explanations among DBAs. Negotiation also settles an agreement on access restrictions and the desired level of consistency between imported materialized views, and the original version in the exporting system (as discussed below).

18.5.1 Exchanging Metadata

The specification of exported data is stored in an export schema (or export view) in the exporting system. The export schema provides access to both the data definitions (description of types, classes, ...) and the corresponding instances (the objects in the database). Export schemas may be defined using different techniques: subschemas, views, virtual objects,... Usage permissions granted (read, update, ...) are also specified. Export schemas may be made available to designated users only (a usage permission is attached to the view itself), or they may be broadcasted (available to everybody), or designed to become a component of some designated federation. It

is worthwhile noting that data is exported to become part of one or more federations (thus providing for location transparency), not for direct access as in multidatabase systems [372]. If a local system accesses a federation it contributed to, the system is importing data it exported. From the semantic perspective this is perfectly correct (similar to what happens everyday in international trade). From the performance point of view, it is expected that, as in DDBSs, the federated query processors will be able to access local data directly, rather than access some copy elsewhere.

18.5.2 Exchanging Data

A well-known problem in moving objects around in a federated system relates to object identity. The requirement clearly is that object identity has to be preserved (to avoid semantic loss). This implies that, when an object comes back to its exporter, updated by some importer, it is recognized by the exporter so that the update can be correctly applied to the local database.

A first problem obviously comes from the fact that the federation may include value based systems (relational DBMSs), which do not know about object identity. However, an object-oriented system can also offer values to the federation, instead of objects, either because it generally does not extract object identities, or because some type of queries it accepts generate values as result. If object identities can be provided by component systems, a global identity is easily built [245, 170]. It will include the local identity, which allows to later apply external updates on the original exported object. If no local identity is available to the federation, exported values can be turned into objects by the FDBS, but this does not make sure that an external update will be correctly propagated. To that extent, restrictions have to be imposed on exported values: i.e., they must contain a key which uniquely identifies the value in the local database (as in relational databases). Otherwise, export should be restricted to read only import. When an imported object is stored by the importer, it gets a local object identity. If the object might have to be returned to the exporter, the global object identity (which is locally meaningless) has to be stored as an additional attribute of the object (not visible to users), for future reuse in the federation layer. An object to be imported may contain object identities as references to component objects in the exporting database. If the component objects are not imported, references are just dropped before import. If a component object is imported as a component value in the importing system, the reference has to be replaced in the imported copy with the value of the corresponding object. If the component object is imported as an object, new references, local to the importing system, should complement the imported ones (or replace them, if the importer is not allowed to update the object).

18.6 Implementation Issues

Since object identifiers in general are not available outside a system, the specification of information to import has to be done by means of algebraic expressions [502]. Also exported information can be specified comfortably with a query expression. The query is given a name whose value is the result of the query, i.e., a data definition and a set of instances. At each use of the name the query can be re-executed in the exporting system and the result transmitted to the importer. To the user there is only one version of the information located in the exporting system, and up to date. A problem arises when the amount of data specified by the expression is large, the data transfer might

take considerable time. To avoid the delay, the user might want to create a local copy, thus he explicitly stores the result of the import expression at his own site. Now, on the other hand, the importer does not know whether his local copy is up to date or not, since updates on the original version are not by default propagated to his copy. As a compromise, it would be possible to refresh the copy of the importer at some access occasions, but not all.

Another solution to the delay problem would be to keep an automatically updated local copy that the user never sees. The problem with this approach is of course how to ensure the consistency between the copy and the original. The following is a summary of techniques used to solve materialized view management problems. Materialized view update strategies can be classified according to, first, the consistency that is required between the original and the materialized view, secondly how to detect that an update to the original is relevant to a materialized view, and lastly how to actually update the view.

18.7 Consistency Requirement

The strongest consistency requirement is *transactional consistency*, i.e., the materialized view is updated before the update transaction on the base has terminated. This implies total correspondence between the versions [66]. The consistency requirement may be relaxed in a well-defined way [15] where the notion of quasi-copy is introduced. A quasi-copy may deviate from the original in one of the ways:

- Time delay
- Version number
- Numeric deviation

A way to specify these deviations exactly and to calculate the export costs is proposed. An even weaker consistency requirement is proposed by [362], where a "snapshot" is a copy that is updated only periodically, to be used by update-insensitive applications.

18.7.1 Detection of Relevant Updates

An update operation on the database is made on a set of objects specified by a query. If these objects are also present in a materialized export view, the view has to be updated, otherwise not. Thus the problem is to say whether the sets of objects specified by two queries are disjoint or not, a problem that is in general impossible to solve. It is solvable for queries involving only the project or join operators, and the select operator with a condition predicate containing only conjunctions. This is used by [66] to obtain transaction consistency. [171] improves the method. A different approach is made in [362] to update snapshots. A "snapshot" is the stored result of a view definition expression. A snapshot is read-only and is updated "periodically", i.e. all updates on the database since the last snapshot are propagated. Here a timestamp is added to every tuple when it is updated in order to be able to say if the tuple object has been updated since the last snapshot refresh.

18.7.2 Differential Refresh

When an update has been found relevant to a view, the view materialization has to be updated. One way to do this is to recompute the query specifying the view and to update the whole query. This clearly may result in the rewrite of a lot of unchanged data. It is desirable to update the view only with the data actually changed. [362] proposes a differential refresh algorithm for snapshots. When the snapshot is to be updated, the tuples of the base table are traversed, and if a tuple is younger than the snapshot and satisfies the view condition it is transmitted to the snapshot. Improvements are made by discarding clusters of not interesting tuples. [66] proposes a different method based on the application of the same update operations on the view materialization, that were applied to the base table. This method relies on the distributivity properties of set union and set difference, which do not hold in the relational model, but hold for object oriented models. The method is used to provide transaction consistency, since the view update is included last in the update operation transaction of the base table. The authors claim that their results are also valid for snapshot update, i.e., when the view update is made after the transaction updating the original table has terminated. It is not evident that the application of an update operation at a time point when the execution environment may have changed will produce the same database state as in the original environment.

Another way to improve system efficiency at update, and to avoid the exporter bottleneck problem that occurs when a lot of importers are requesting updates from the same exporter, is proposed in [306]. The idea in this method is to let the importing systems re-export information to other importing systems, thus relieving the original exporter from work.

There are cases when keeping a local copy at the importing site is necessary for performance reasons. This copy can be invisible to the user, giving the impression that there is only one version of the information. The consistency requirement varies, depending on application requirements, from periodically updated snapshots, over quasi-copies to transaction consistency. The results in the referenced papers concerns the relational model, it remains to be studied whether they are also applicable to semantic and object- oriented models. Nothing is however pointing in any other direction, on the contrary, it seems like further problems encountered in the relational model do not occur in this environment. For example, the problem caused by the fact that projection is not distributive over difference in the relational model, does not occur in a model with object identities, see [66].

An interesting approach seems to be to combine the periodic update with the differential refresh method that applies the update operations to the view. To update the view at every modification of the original data even if the materialized view is not used at the moment seems inefficient, instead the update should be made on request from the view user, then all the updates on the base table can be applied at one time. An open question is if the update operations applied in the same order will produce the same result at the later time point. Another question is how the mechanism to update materialized views can be made symmetric, i.e., how updates on the materialized view can be propagated to the original.

18.8 Conclusion and Future Research

In this chapter we have presented the motivation and the key features of the ongoing FEMUS research project. The aim of the project is to explore architectural and semantic related issues of federated database systems. We have discussed, in particular, two of the most distinguishing features of the project: data model translation and database integration.

Cooperation between heterogeneous systems can involve bi- directional translations between data models of equivalent semantic power. Semantic data models and object oriented data models are more and more included in current federated database systems. In order to avoid a proliferation of translators in the federation, they are often used as common data model. We have examined in detail the characteristics of the translation process between object oriented and extended semantic data models. The translations rules between two representative models, ERC+ and COCOON, are given as an example. Their scope covered both schema and operation mapping.

Next, we have investigated database integration issues in interoperable systems. Integration techniques are used in these systems to build unified views of information imported from different databases. The main thrust of our research is to extend to federated database systems two alternative integration methodologies we have developed. The first one uses interschema correspondence assertions and integration rules to derive an integrated schema from a set of heterogeneous input schemas. The second one uses the existing view mechanism to augment the input schemas with interschema correspondences expressed as constructs of the model (interschema functions in the case of COCOON). We presented the key features of these integration methodologies and an example to support their comparison.

Our future research effort is on:

- the specification and identification of tools to aid the translation process. The translation rules can serve as a basis for defining a rule based generator for the translation process;
- the specification and implementation of a federation server to aid federation users in sharing information, meta information, and schema construction tools (translators and integrators). The server must provide sophisticated dictionary and/or directory look up services to aid navigation and negotiation throughout the federation;
- the extension of the assertional integration technique to solve more cases of schema discrepancies and the extension of the augmentation integration technique to cope with heterogeneous input schemas;
- the extension of both integration techniques to the integration of object-oriented methods.

ACKNOWLEDGMENTS

The authors are indebted to the following people, that contributed to the FEMUS project: Hans-Jörg Schek (ETH Zürich), Marc Scholl (University of Ulm), as well as the students Amadou Fall (EPF Lausanne) and Hans-Peter Waldegger (ETH Zürich) who did some of the prototype implementation.

Chapter 19

Communication and Synchronization Issues in Distributed Multimedia Database Systems

Shirley Browne*

19.1 Introduction

Current research into high-speed wide-area communications is raising the possibility of new distributed multimedia applications, including distributed multimedia database and information systems. The goals of research in this area will be to extend distributed database and information retrieval paradigms to the following:

- database access over large-scale wide-area packet-switched networks,
- storage, retrieval, and display of multimedia objects.

Although CD-ROM technology provides local mass storage for static information, distributed databases will be crucial for accessing information which is constantly changing, inherently distributed, or accessed infrequently by a given user. As the amount of information available threatens to overwhelm us, multimedia offers a way of increasing the machine to human bandwidth through the use of images, animation, sound, and video, as opposed to purely textual display. The economics of bandwidth sharing argue for use of packet-switching, as opposed to circuit-switching.

Motivating the development of wide-area multimedia information systems will be the desire for large-scale collaboration on the Grand Challenge problems, saving lives and reducing costs through the use of medical databases, and consumer demand for commercial applications such as video browsing and home shopping. For example, Project Sequoia 2000, funded by Digital Equipment Corporation at the University of California, involves work on distributed database management of large constantly changing global change datasets, along with network facilities for accessing, visualizing, and analyzing the data [300]. Distributed medical databases which allow a doctor handling an emergency to instantly review a patient's medical records remotely will greatly improve the quality of emergency care. Applications of video browsing, ranging from computer dating to long-distance real estate services [281], will provide convenience and cost savings to consumers.

Libraries will become important users and providers of multimedia information services. Although libraries have traditionally been repositories of printed information,

*Department of Computer Science, University of Tennessee, Knoxville, TN 37996-1301, U.S.A

print has limitations, such as fixed, static representation of information, lack of interactivity, and its restriction to single-user mode. Multimedia information systems hold promise for overcoming these limitations, as they can provide a multisensory, dynamic, interactive, multiuser environment. Future library multimedia computer systems are expected to utilize expert system technology to assist users in selecting and retrieving multimedia information. This technology will be deployed at specialized, powerful database servers in the networked environment [25]. The problems of interconnecting heterogeneous networks and databases remain to be solved, however, before integration of diverse information sources can take place.

Object-oriented software technology, with its properties of encapsulation of operations, distribution of logical autonomy, processing concurrency, and reactive control structures [603], may prove to be the most appropriate model for the design of multimedia databases. With the object-oriented paradigm, it will be possible to incorporate the display and processing information associated with objects into the objects themselves. Other relevant information includes the synchronization and communication performance requirements of multimedia objects.

Research into object-oriented and multimedia database systems is being carried out at a number of institutions. O-RAID is an object-oriented extension of the RAID distributed database system being developed at Purdue University [60]. O-RAID provides a class structure for objects, with support for large, complex objects and for object composition and replication. Although not currently a multimedia database, the modular, object-oriented architecture of O-RAID lends itself to a natural extension to multimedia objects.

The POSTGRES database system extends the relational model to incorporate object-oriented features such as inheritance and methods, along with a rules system for adding intelligence to the database [556]. An extended version of POSTGRES is being employed to help manage terabytes of global change data for the Sequoia 2000 project mentioned above [300]. Work at Boston University involves the capture of temporal relations for the storage and subsequent retrieval of multimedia objects, using POSTGRES as the database management system, an X11 user interface, and JPEG compression.

OMEGA is a object-oriented multimedia database system being developed at the University of Library and Information Science in Japan [391]. This research project is extending the object-oriented model to represent temporal and spatial information about multimedia data. The author of [391] argues in favor of the object-oriented approach, as opposed to the hypermedia approach.

Researchers at Washington University in St. Louis are designing and implementing an experimental ATM network which will link physician's workstations with medical databases. The system is being developed with the goals of eliminating the manual management, retrieval, and transportation of X-ray and other imaging data, and of allowing simultaneous access to this data by multiple users.

The Collaborative Visualization (CoVis) Project at Northwestern University involves the design of future distributed multimedia high school science learning environments [449]. Multimedia database servers will reside at CoVis network testbed sites. These servers will support collaborative visualization activities for geographically dispersed students and teachers working jointly on science projects.

Other research efforts include the Multimedia Information Laboratory at Syracuse University [50], the IRIS Hypermedia System at Brown University [239], research into efficient search and retrieval techniques at the MIT Multimedia Lab [364], and research at Fermilab and CERN on extensions to World-Wide Web for the high-energy physics

community [47].

A number of challenging problems remain to be solved before large-scale distributed multimedia databases become a reality. Efficient means of organizing and storing extremely large quantities of diverse types of data must be found. Appropriate indexing and search strategies, with flexible user interfaces, must be developed. Techniques for retrieving, composing, and synchronizing the components making up a complex multimedia object are needed. Specification of the communication requirements for the network transfer of multimedia objects will be a necessary function of a distributed multimedia database system. The distributed system must also interface with the underlying communication services to obtain the quality of service desired.

The remainder of this chapter surveys issues involved with synchronization and communication requirements of distributed multimedia databases. Solutions which have been proposed in the literature are described and related to each other. Finally, suggestions are made of fruitful areas for future research.

19.2 Characteristics and Requirements

Uses of distributed multimedia databases will span a variety of application areas, including the following:

- Education – electronic books, educational videos, simulations, multimedia courseware, and collaborative projects.
- Libraries and museums – Literary, musical, and artistic works, and multimedia presentations.
- Offices – documents, memos, training and sales videos.
- Medical databases – Case histories, X-rays, test results, and digital data from computer tomography, magnetic resonance, and ultrasound.
- Computer aided design – Design documents, symbols and components, simulation.
- Scientific databases – Sensory data, data visualization, animation, and simulation.
- Geographic databases – Maps, satellite images, demographic information.

Despite the wide arena of application areas, a common set of requirements can be defined. The functional requirements for a multimedia database server include the following [50]:

- management of complex multimedia information, in the form of text, images, audio, and video,
- spatial and temporal integration of information units of various types into the multimedia object of interest to the user.

Because of the extremely large storage requirements of some multimedia information types (e.g., images and video), accessing a remote multimedia database is expected to be a real-time playback application. A real-time playback application, as described in [113], involves packetization and transmission of some signal by the source followed by depacketization and replay at a designated playback point by the receiver. Any data that arrives before its designated playback point can be used to reconstruct the signal; data arriving after its playback point is useless. Hence, playback applications

are sensitive to network-introduced delay and jitter (variation in delay). A multimedia presentation has the additional requirement that the playback points of multiple signals must be coordinated.

The coordination of multiple streams during playback is called *temporal synchronization* in [369]. The temporal composition of a multimedia object may consist of either fine-grained *continuous*, or *isosynchronous*, relationships, or course-grained *synthetic* relationships [369]. An example of a continuous relationship is the coordination between audio and video streams representing the voice and image of a speaker. An example of a synthetic relationship is the synchronization of text and images for a multimedia display. A distributed multimedia database system must provide facilities for storing these relationships and for enforcing them when a multimedia object is retrieved and transmitted over the network for display to the user.

In general, the need for synchronization arises from several causes, including the following:

1. Transmission of a multimedia object may involve transmission of related data streams over different communication channels (e.g., motion video and accompanying voice). Because of their different communication requirements (e.g., low delay jitter for voice, low loss rate for compressed video), it is desirable to use separate communication channels for different types of media. Since timing relationships are typically not preserved across different channels, these relationships will need to be restored at the destination before correct playout of the related streams can occur.

2. A displayed object may be synthesized from individually stored components. In a multimedia database, a complex object may consist of components of different media types which are stored separately. Synchronization requirements for such an object would specify how the various components fit together temporally. A database server would need to retrieve the object's components and transmit them in such a way that the synchronization requirements are not violated or made impossible to enforce at the destination. The specification for the synthesis of an object from its components may be pre-orchestrated and stored in the database or may be constructed at query time.

3. In a distributed environment, retrieval of a complex object may involve the retrieval of individual components from different database servers. For example, a sequence of images (i.e., slide show) may be stored at one location, with a voice commentary stored at a different location. Display of such an object will involve the combination at the destination of data streams coming from different database servers. Somehow the timing relationships between the components must be enforced by synchronization actions taken at the various sources, at the destination, or at both the sources and the destination.

4. Pipelined decompression architectures can introduce additional delay and delay jitter into a data stream. Since the use of compression is highly desirable in a distributed multimedia environment for reducing the amount of data that must be transmitted, techniques are needed to compensate for the jitter introduced by the decompression algorithm.

5. Starvation of a data stream may occur because of a shortage of network resources. With appropriate techniques for statistical multiplexing of network bandwidth, such starvation can be made highly unlikely, but there is still a chance that it can occur. Network users may also take a calculated risk of starvation occurring in

exchange for a lower charge for use of the network. Regardless of why starvation occurs, data streams that are receiving adequate service must be synchronized with those that have fallen behind because of starvation.

The communication requirements of multimedia objects vary greatly, depending on the type of media and on the degree of compression being used. The bandwidth required for sound can vary from 2.4 kb/s for coded voice up to 1.4 Mb/s for CD quality. Full-motion video can range from 30 Mb/s for medium resolution black-and-white to 750 Mb/s for high resolution color [368]. Sizes of images can vary from a few to tens of megabits, depending on the resolution. Lossless data compression can reduce these quantities by factors of 3 to 4; greater reduction, up to a factor of 100, is possible using lossy compression, but with some image degradation. The quality of service required for different media can be expressed in terms of bandwidth, delay, delay jitter (i.e., the variation in delay), and error rate. Sound has stringent delay and delay jitter requirements, but can tolerate some loss due to error. Image and compressed video may have stringent error requirements. Video has less stringent delay and delay jitter requirements than audio. Use of delay compression can have an adverse effect on delay jitter. Throughput, delay, jitter, and reliability requirements may be expressed either deterministically or statistically [186]. A distributed multimedia database system must provide facilities either for storing quality of service requirements of objects or for calculating them on demand, and for negotiating with the communication service provider to obtain the desired quality of service.

Quality of service requirements can also be classified as either *guaranteed* or *predicted* [113]. If a communication system provides guaranteed service, it ensures that the quality-of-service commitment will be met, provided the client conforms to its end of the service contract (i.e., by staying within its declared average and peak transmission rates and its declared burst length). Predicted service, on the other hand, attempts to deliver consistent performance to the client, based on past network conditions, but there is no guarantee. The advantages of predicted service are that it allows greater sharing of bandwidth, thus reducing communication cost, and that it allows an adaptive application to adjust its playback point to take advantage of current network conditions. Predicted service may suffice for casual users of multimedia database systems (e.g., library browsers, office applications), whereas time-critical and real-time applications may require the more dependable but more costly guaranteed service.

19.3 Communication Approaches

The Internet is expected to evolve into a network that can support multimedia applications [448]. Rather than interfacing directly to the underlying network, the communication manager of a multimedia database system would most likely interact with a higher level protocol such as the *Multi-flow Conversation Protocol* (MCP) [636], being developed at the University of Kentucky, or Bellcore's Touring Machine System [343]. To satisfy bandwidth, delay, jitter, and bit error requirements, data streams carrying multimedia information will need to be sent over flows having performance guarantees [128]. In the approach in [636], flows are grouped into a logical unit called a *multi-flow conversation*. MCP is a transport level protocol that provides service primitives for establishing, managing, and terminating multi-flow conversations. The Touring Machine System provides a set of logical abstractions for establishing and managing multimedia communication sessions. These abstractions hide the complexities of resource allocation, network access control, and session management from the application. A session

consists of a set of medium-specific *connectors*, where a connector is a multiway transport mechanism among multiple *endpoints*.

In the context of public broadband networks [543], the communication manager for future multimedia database systems will need to interact with the admissions control and policing mechanisms in use by the service provider. The admissions control mechanism ensures that the average and peak rates, burstiness (defined as the ratio of peak to average rate), and burst length of a source are not exceeded. Thus, when making a call request (e.g., setting up a query session), the database communication manager will need to give information about call requirements and source traffic characteristics to the network service provider. One possibility is to store this information in the database schema itself (e.g, in the form of Estimated Bit Rate tables, as in [167]). Another approach is to calculate the required information from the parameters of a particular query session and network and from accumulated statistical information collected during previous query executions. For example, an algorithm for calculating buffering requirements by processing the database schemas is given in [370].

Various bandwidth allocation mechanisms have been proposed for admissions control [262, 168, 607]. These mechanisms attempt to calculate the effective requirements of a connection request and determine whether or not sufficient resources exist to satisfy the request. Most proposed policing mechanisms use the notion of a token bucket filter [465]. The token bucket filter works conceptually as follows. Tokens enter the token bucket at a constant rate, up to some maximum depth. A source that wishes to transmit a packet must remove an appropriate number of tokens from its token bucket (e.g., a packet of length 1000 bits would be required to remove 1000 tokens). If the tokens are not available, the packet may be either queued or dropped. The token entry rate governs the average rate of the source, while the burstiness is governed by the bucket depth.

Packet scheduling algorithms for controlling congestion and handling real-time traffic have been proposed in [143, 226, 113]. The weighted fair queuing algorithm presented in [143] is a rate-based flow control scheme that guarantees a given rate to each flow. In [445] it is shown that weighted fair queuing, when combined with token bucket filtering, allows the network to make worst-case performance guarantees on delay. Simulation results in [113] show, however, that the worst-case bounds are typically much worse than the actual delay, and that a modification of FIFO scheduling, while not guaranteeing a worst-case bound, appears to achieve much lower variance in delay. The stop-and-go queuing technique described in [226] makes less efficient use of bandwidth than the above schemes, resulting in larger average delay, but maintains tighter control of jitter in the network. As research on the performance of packet scheduling algorithms continues, it appears likely that network providers implementing these algorithms will be able to make deterministic and/or statistical guarantees which can be used by applications to guarantee adequate quality of service to users.

Future multimedia database communication services will likely run over Asynchronous Transfer Mode networks. Asynchronous Transfer Mode (ATM) is a fast transport scheme, based on fixed-length cells, that is projected to become widely used for carrying multimedia traffic. ATM achieves bandwidth efficiency by statistically multiplexing bursty traffic from virtual connections at the expense of cell delay and loss. Such multiplexing allows varying bit rate on demand, with the ATM specifications of maximum bit rates at 150 and 600 Mbps. To support multimedia database retrieval, ATM will need to provide guarantees for cell delay and cell lost performance requirements for all media types supported. For example, voice data are delay-sensitive while image data are loss-sensitive. These guarantees will be accomplished through the

design and implementation of appropriate traffic control mechanisms. Possible traffic policing mechanisms are surveyed and compared in [465]. Establishing an ATM call requires negotiation of a traffic contract, which includes the attributes of maximum and mean cell rate as well as the quality of service requirements with respect to call loss, delay, and delay jitter. The policing function then controls the cell stream during the active phase of the call and restricts the traffic source to the characteristics negotiated in the contract.

A layered view of an ATM network is shown in Figure 1. Traffic flows originate from sources at the services layer. An ATM adaptation function is required at the adaptation layer to segment the traffic flow at the source into fixed-length cells and to reassemble cell contents at the receiver. For message-oriented services, messages are reassembled. For continuous traffic, the variable transfer delays of the cells must be smoothed by buffering. The adaptation function may be provided either by the network or by the end system. In the case of distributed multimedia databases, specialized adaptation layers may need to be designed to support the coordinated retrieval, transfer, and display of related streams of data from different sources. The cells are transferred across the network by the transport layer which is supported by the underlying physical layer. Performance impairments which may be introduced include cell error, cell loss, and cell delay. Cell loss can result from bit errors at the physical layer, buffer overflow at transport-layer switches, and smoothing buffer overflow at the adaptation layer. Cell delay can result from physical propagation delay, transport-layer switch queuing delay, and adaptation layer cell assembly and smoothing delay. Although there have been numerous studies of performance requirements at the ATM transport layer [168, 630], there has been little work on performance evaluation of adaptation layer functions. Adequate performance at the adaptation layer, as well as effective allocation of performance requirements between the adaptation and transport layers, will be critical for good performance of multimedia databases.

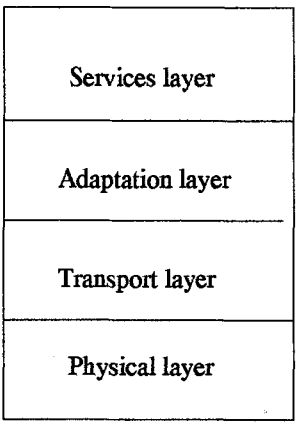


Figure 1. Layered model of ATM network

19.4 Synchronization Approaches

Synchronization requirements will need to be expressed at the application level. The Object Composition Petri Net (OCPN) has been proposed as an abstract model for representing the synchronization of elements composing a multimedia object [369]. The OCPN is shown to be capable of modeling any of the thirteen possible temporal relationships between two temporal intervals (The logic of these relationships is presented in [13]). One of these temporal relations, the overlaps relation, is shown in Figure 2(a) for a multimedia object with components α and β. Here, τ_α represents the duration of α, τ_β the duration of β, and τ_δ the interval from the beginning of the playout of α to the beginning of the playout of β. The OCPN is a timed Petri net, with the playout processing of object components represented by places, and with instantaneous firing of transitions. The firing rules for the OCPN are summarized as follows [369]:

1. A transition fires immediately when each of its input places contains an *unlocked* token,
2. Upon firing, a transition removes a token from each of its input places and adds a token to each of its output places,
3. Upon receiving a token, a place locks the token for the interval specified by the place's duration, after which time it unlocks the token.

The OCPN corresponding to the overlap temporal relation is shown in Figure 2(b), with places represented by circles and transitions by vertical lines. The OCPN is actually a hierarchical model, in that a subnet may be replaced by an equivalent abstract place. In [369], a hierarchical tree-structured database model is proposed for representing and storing OCPN's. Two types of tree nodes are used – the *terminal* node which represents an atomic object and includes a pointer to the location of the actual data, and the *nonterminal* node which indicates the temporal relation imposed on its children. The representation of the overlap relation in this schema is shown in Figure 2(c). The OCPN model is extended to n-ary temporal relations (for more efficient representation) and reverse temporal relations (to allow reverse playout) in [371].

Algorithms for processing the OCPN associated with a multimedia object that is to be retrieved, transmitted over a network, and displayed are described in [370]. Algorithms are given for carrying out the calculation of the following quantities:

1. the playout deadline schedule Π,
2. the retrieval time schedule Θ, and
3. the destination buffer requirements.

The timing for a single object is shown in Figure 3, where the network latency λ includes activities such as packetization, transmission, and decompression, and the control time T is chosen by the algorithm so as to guarantee synchronization within a given probability of failure. Buffer requirements are calculated for the purpose of smoothing variations in latency and of storing objects that must be held at the destination for some amount of time prior to their playout deadline. The schedules produced by the algorithms must somehow be enforced by the communication system. One possible interface to a proposed system-level network synchronization protocol, the Flow Synchronization Protocol being developed at BBN [181], is described below.

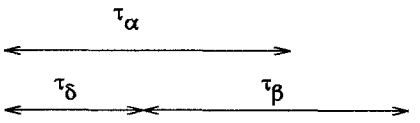


Figure 2(a). Overlaps relation

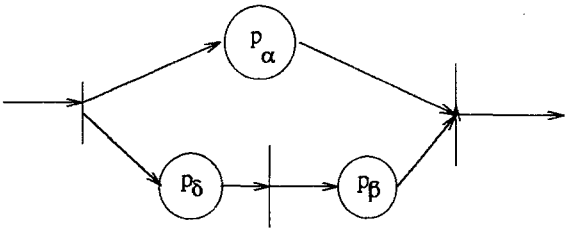


Figure 2(b). OCPN for overlaps

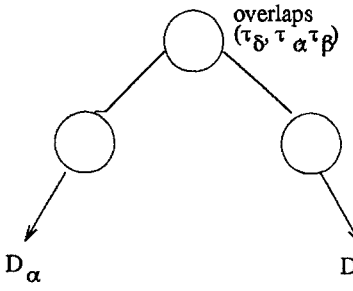


Figure 2(c). Tree-structured schema

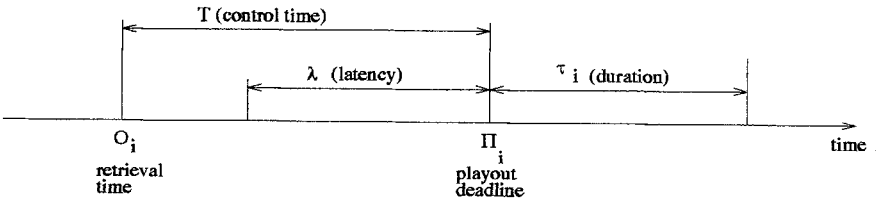


Figure 3. Timing for retrieval, transmission, and display of a single object.

An alternative approach to expressing synchronization requirements is proposed as an extension of the CCITT standard Office Document Architecture (ODA) to multimedia objects [253]. ODA is a hierarchical object-oriented model designed to allow transmission of documents over a network and either direct display or storage and subsequent editing by the recipient. The structure of the model is shown in Figure 4. Document types are defined as subclasses in the *generic description* class hierarchy. Each instance of a document has a specific description, derived from the appropriate generic description, which details the actual contents of the document. Logical objects include such things as sections, paragraphs, and headings. Layout objects include page formats, fonts, page breaks, etc. The generic description defines what logical and layout objects may exist and contains correspondence relation rules describing how logical and layout objects interrelate. In the specific description, the *logical structure* subdivides the document on the basis of meaning, and the *layout structure* subdivides it on the basis of paginated layout. A *basic* object is an object that is not subdivided into smaller objects. In the specific description, the basic logical and layout objects contain pointers to actual content portions. The specific description contains correspondence relations that interrelate the specific content portions. The document profile contains information such as the title, author, and revision history. An *attribute* is a property of a document or of a document component which expresses a characteristic of the component or a relationship with one or more other components. The set of attributes associated with a document as a whole is the document profile. An ODA document may by transmitted in either finalized or editable form. In the case of finalized form, only the layout structure and document profile need to be sent, since these are all that is needed to display the document. In the case of editable form, however, the logical structure must also be sent so that the recipient can modify it.

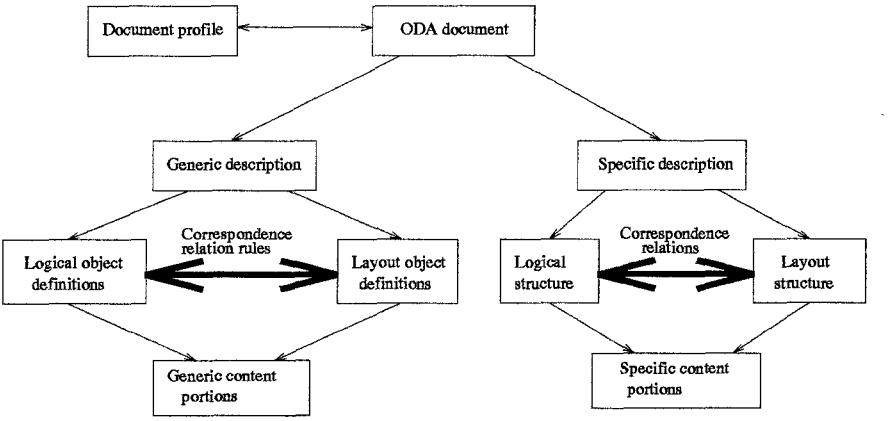


Figure 4. ODA document model

A notation based on path expressions for the description of synchronized actions composing a multimedia object is presented in [253]. ODA extensions are described which integrate the semantics of path expressions into the layout structure of a document. The extensions are compatible with the current version of ODA. Multimedia objects are assumed to be logically decomposed into atomic actions, which may be synchronized only at their startpoints and endpoints. *Path operators* are defined for the

possible temporal relations between pairs of actions and among multiple instantiations of an action. A path expression is composed of atomic actions and path operators, and it describes which paths are allowed for the actions – i.e., the possible orders in which the actions may be executed. Path expressions define the semantics of the synchronization without imposing a specific synchronization mechanism. A total of six path operators are described, but the following three are sufficient for describing Allen's thirteen possible temporal relations between pairs of actions:

$A \wedge B$ *Parallel-last:* Actions A and B are started together and execute concurrently. The composed action terminates when both A and B have terminated.

$A \vee B$ *Parallel-first:* Actions A and B are started together and execute concurrently. The composed action terminates when either A or B has terminated, whichever terminates first.

A ; B *Sequential:* B executes immediately after A – i.e., the endpoint of A equals the startpoint of B.

For example, the overlap of α and β, followed by the overlap of β and γ, as shown in Figure 5(a), would be represented by the path expression shown in Figure 5(b). In this example, δ (a dummy action which functions as a place holder) is executed only once but appears in the two path expressions to produce the desired overlap between α and β and between β and γ.

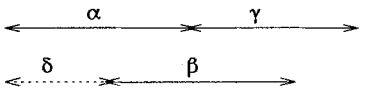


Figure 5(a). Overlaps relations

$\alpha / \backslash \delta$; γ

δ ; β

Figure 5(b). Path expressions

A path expression may be viewed as a tree structure, with the path operators as internal nodes and the actions as leaves. Such a tree structure corresponds to the hierarchical structure of ODA. The current version of ODA provides for a time-invariant presentation of a document, with the layout structures describing the spatial relationships between objects. The only current time-based attribute is *Imaging Order*, which gives the sequential ordering for layout objects in the layout object to which they are immediately subordinate. The author in [253] proposes extension of the layout structure of ODA to include attributes describing temporal relations. Extensions to the logical structure of ODA documents and to time-variant content portions are left for future work. The following new attributes are proposed for the basic layout objects:

B1 *Content Temporal Type* – static (time-invariant) or dynamic (time-variant),

B2 *Duration* – presentation time in Basic Time Units.

It is possible for a static object, such as an image, to have a positive duration. In the case of a static object with zero for duration, the object is presented until it is terminated by its successor in the Imaging Order, thus preserving compatibility with the current version of ODA. The following new attribute is proposed for composite layout objects:

C1 *Object Synchronization Type* – the type of path operator associated with the object, which may be parallel-last, parallel-first, sequential, or selective.

The attribute values parallel-first, parallel-last, and sequential have the meanings described above. With the *selective* attribute, only one of the subordinate layout objects will be displayed, depending on the user's choice. Additional attributes are defined for composite layout objects which concern additional timing constraints and user interactions.

Before presentation of a time-variant document, the layout structure must be evaluated, and the timing relationships must be extracted and enforced. As in the current ODA standard, the document imaging process, concerned with presentation of the document to the user, is not defined by the proposed extensions and depends on the presentation device being used. An important research question, however, will be how to interface the proposed ODA extensions with communication protocols and database user interfaces.

In still another approach to the expression of synchronization properties, the object-oriented multimedia database system OMEGA [391] incorporates temporal properties into the class TemporalObject. Members of this class, called temporal objects, have internal state variables *birthTime*, *deathTime*, *tempoPrec*, and *tempoSync*. The value of *tempoPrec* is calculated from the birth time and death time information, and indicates which object or objects immediately precede an object, and by how many seconds. The value of *tempoSync* represents which other objects must be synchronized with the object. Two objects are considered to synchronize if their lifetime periods overlap. Temporal properties are then addressed through the class definition language, the database programming language, and the query language.

The database synchronization manager may interact with a lower-level synchronization protocol such as those described in [181, 635, 17, 464]. A lower-level synchronization protocol provides primitives which may be called from an application program to access communication system synchronization facilities.

The Multi-flow Conversation Protocol (MCP) described in [635] provides a way of combining logically related data streams which have performance guarantees, called *flows*, into a logical unit called a conversation. MCP provides a token mechanism for enforcing conversation concurrency control, or floor control. The mechanism includes primitives for creating, replicating, distributing, and deleting tokens. Synchronization is approached through the enforcement of a Δ-*causality* relation which is defined as follows for messages m_1 and m_2 and a source S:

$m_1 \rightarrow m_2$, or m_1 *precedes* m_2, if

1. S sends m_2 after S sends m_1, or
2. S sends m_2 after S receives m_1.

$m_1 \stackrel{\Delta}{\rightarrow} m_2$, or m_1 Δ *precedes* m_2, if

1. S sends m_2 after S sends m_1, or
2. S sends m_2 after S receives m_1 and end-to-end delays for m_1 and m_2 are both less than or equal to Δ.

If $m_1 \stackrel{\Delta}{\rightarrow} m_2$, then m_1 and m_2 are delivered in causal order at all destinations. If $m_1 \rightarrow m_2$, but $not(m_1 \stackrel{\Delta}{\rightarrow} m_2)$, then m_1 may be delivered after m_2 or may be dropped.

MCP also provides for expedited delivery of control messages. MCP does not require synchronized clocks, but does require a bound on clock drift between real-time clocks at different sites. Synchronization requirements for a multimedia object would be enforced by sending appropriate control messages and taking advantage of the causal order that is enforced on the delivery of these messages and the actual data messages.

The Logical Time System, or LTS, is a synchronization abstraction implemented over TCP (the Internet Transmission Control Protocol) as part of the Acme continuous media I/O server developed at UC Berkeley [17]. Application programs may deal with *logical devices*, which are abstract versions of physical devices. Each logical device is bound to an LTS. I/O on logical devices is synchronized in that data units with the same timestamp are displayed at approximately the same real time. Each LTS has a current time value. While an LTS is running, its current value increases at approximately the same rate as real time. Acme provides primitives for creating an LTS, for binding a logical device to an LTS, for starting and stopping an LTS, for querying an LTS as to its current time, and for setting an alarm. The bind primitive allows specification of a max_skew value, which is the maximum allowed difference between a logical device and the current LTS time value. Acme enforces synchronization by adjusting the rates of logical devices, using the techniques of skipping and pausing. The LTS abstraction provides local abstraction only. That is, all logical devices for an LTS must reside at a common location. It is not required however that the associated physical devices all reside at this location, but delay and jitter induced by the network connection between a logical device and its associated physical device would not be handled by Acme. The Acme system requires synchronized clocks.

The Flow Synchronization Protocol described in [181] uses a modular architecture that permits the application to tailor the synchronization actions to its own synchronization requirements. The protocol timestamps the data at the sources and equalizes the delay at the destinations, so that the end-to-end delay among flows is synchronized. The protocol relies on the underlying use of clock synchronization protocol to achieve a global time reference. The timing relationships are illustrated in Figure 6. In the database context, the collection delay would include the time required to process the query and seek and access the data, and the delivery delay would include the time to decode and present the data to the user [368]. The application is allowed to specify the following three application-specific function to override the system-provided default functions:

1. a Filtering Function to estimate the unequalized flow delay,
2. a Synchronization Function to compute – given the estimated individual flow delays, the maximum acceptable end-to-end delay for each flow, and the flow priorities – the common synchronization delay Δ.
3. a Delivery Function to schedule data delivery (following insertion of the equalization delay) to the application.

The protocol is adaptive, in that it can adjust the synchronization delay Δ to reflect network conditions and performance. For example, if too many packets are being discarded because of late arrival, the synchronization delay may need to be increased. Thus, the database synchronization manager would be responsible for supplying the application-specific functions and for contacting the synchronization protocol controller if necessary to change protocol parameters, such as the interval at which Δ is updated.

The protocol assumes a flow model in which each flow has a source process sending data to a destination process. The source timestamps the data and the destination implements equalization by buffering the data. The destination processes execute the application-specific functions. Destination processes belong to *synchronization groups*. The processes in a synchronization group regularly exchange messages to compute a common synchronization delay Δ for the group. Synchronization groups can be used both for multiple destinations for the same source and for multiple related flows for

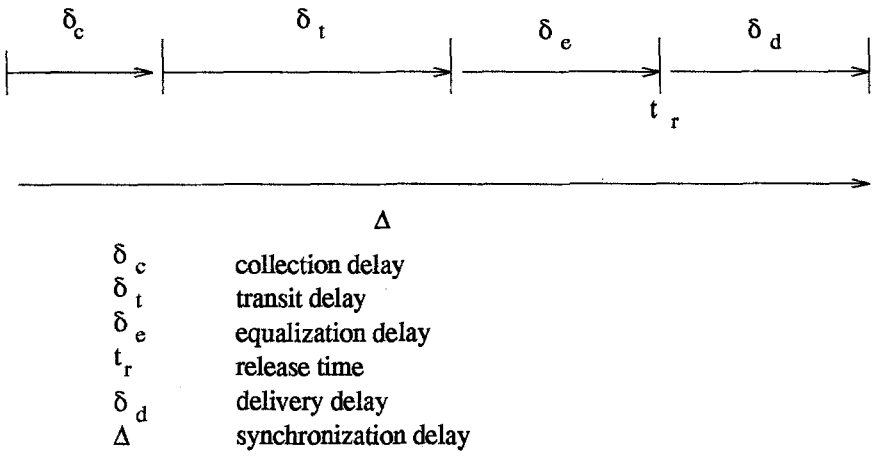


Figure 6. Timing relationships for Flow Synchronization Protocol

the same application. Because the synchronization and delivery functions are executed by the destination processes, it might appear that the OCPN model for application synchronization, described in [369], could not be used together with this synchronization protocol, since the processing of the OCPN which corresponds to computing these functions must take place at the source. However, the Flow Synchronization Protocol provides a mechanism of event messages by means of which the a source process may send information about the flow known only to it to its corresponding destination processes. Thus, the OCPNs could be processed at the source and the results could be sent via event messages to the destinations where they could be used to compute the functions.

A synchronization protocol designed to work in the absence of both synchronized clocks and control messages is described in [464]. The *media mixing* problem has the goal of minimizing the differences between generation times of packets being mixed from different sources, in the absence of synchronized clocks, but in the presence of jitter and transmission delays. The set of packets from different sources that are to be mixed to form a composite media packet is called the *fusion set.* Packets are transmitted from their sources to a mixer. Minimum and maximum bounds on the communication delay from a source to the mixer are assumed to be known. Packets are assumed to be generated at regular intervals, i.e., with a period p. Two packets may belong to the same fusion set if and only the difference between their generation times is less than or equal to p. This requirement is called the *Mixing Rule.* Conditions on jitter and generation intervals are given under which it is possible for the mixer to determine fusion sets. For the possible cases, algorithms for determining fusion sets are given. The possible communication architectures for media mixing range from *centralized,* with a single central mixer, to *fully distributed,* where mixing is performed independently at each destination. To make the mixing algorithms scalable with respect to both the number of participants and the geographical separation between participants, a hierarchical tree-structured communication architecture is proposed, with destinations at the leaves and mixers at the root and internal nodes. A bound on the time a mixer must wait for lost packets is derived that minimizes the real-time delay in a hierarchical architecture.

19.5 Conclusions

This chapter has discussed the issues involved with synchronization and communication in the context of distributed multimedia databases. Some proposed strategies for dealing with these issues have been described. As this field is in its infancy, a great deal of work remains to be done. Further research and development are needed in the following areas:

- Interfaces between multimedia database systems and communication subsystem synchronization primitives.

  Several system-level synchronization protocols are currently being implemented and tested. Methods of storing and/or dynamically determining synchronization requirements of multimedia database object are needed, as well as the means of relaying these requirements to the underlying communication system.

- Automated determination of quality-of-service requirements for multimedia database query sessions.

  Quality-of-service requirements such as bandwidth, delay, and error rate will need to be specified at the time the network connection for a query session is established and possibly modified during the session. It is not reasonable to expect the end user to bear the burden of specifying these requirements. Thus, the database user interface should be able to determine these requirements and relay them to the communication service provider.

- Integration of abstract synchronization and object-oriented database models.

  Some work has already started in this direction, but it needs to be extended. Currently proposed abstract synchronization models require temporal relations to be defined when an object is created and do not allow dynamic, query-time composition. Playout is also restricted to a predetermined rate and direction, with the user as a passive observer. A more interactive model, such as the VCR model described in [386], with operations such as fast forward, reverse, suspend, and resume, would be an improvement. The object-oriented paradigm offers the possibility of having the display and synchronization properties handled by the object themselves, rather than having these properties overlaid in a separate database schema. Methods could then be defined which would allow the objects to react to user input.

- Performance studies using simulation of present and future network technologies.

  Multimedia database systems should be designed so as to be able to adapt to changing network technologies. Projected technologies can be simulated before they become widely available, so that the development of database systems that use these technologies need not lag behind.

- Studies of the effect on synchronization and communication performance of concurrency control mechanisms.

  Certain multimedia databases, such as library-type information repositories, may be largely read-only, but others, such as medical databases that are accessed and updated by physicians, will be updatable. The presence of updates introduces the problem of database concurrency control. Conventional concurrency control

mechanisms can introduce indeterminate delay in the accessing of database objects. Such delay could violate the synchronization and display requirements of multimedia objects. Research is needed to determine under what conditions updating may be allowed, and to determine what concurrency control mechanisms will be most appropriate.

- Participation by the database community in international standards efforts, such as ATM, ISDN, B-ISDN, ODA, and X400.

  To achieve the goal of interoperability over large-scale wide-area networks, standards will be needed, both at the application level and at the communication system level. Database researchers should be actively involved at both levels to insure that the capabilities they need are deployed as part of these standards.

Chapter 20

Multimedia Database Systems

Arif Ghafoor*, P. Bruce Berra†

20.1 Introduction

The need for multimedia information systems is growing rapidly in a variety of fields including business [423, 22], manufacturing [317], education [633, 297], Computer-Aided Design (CAD)/Computer-Aided Engineering (CAE) [383], medicine [466, 587, 309], weather, entertainment [363, 169, 386], etc. Multimedia data include images, audio, full motion video, text and numeric data. Due to the diverse nature of the multimedia data, systems designed to store, transport, display and, in general, manage such data must have considerably more functionality and capability than conventional information management systems. The main issues which multimedia database management researchers/designers need to face include:

1. Development of sophisticated conceptual models which are rich in their semantic capabilities to represent complex multimedia objects and express their synchronization requirements. A transformation from models to a database scheme is then needed. Subsequently, one also needs to specify the object retrieval algorithms.

2. Designing multimedia query languages which are not only powerful enough to handle various manipulation functions for multimedia objects but also simple in handling user's interaction for these functions.

3. Designing powerful indexing and organization techniques for multimedia data.

4. Developing efficient storage layout models to manage real-time multimedia data.

In this chapter, we focus on the first issue of multimedia database systems. The major challenge in multimedia information management is how to synchronize various types of data both in space and time in order to compose complex multimedia objects [367]. The problem is especially acute if we are dealing with "live" data such as digital video or audio. The synchronization requirement plays a key role while designing a multimedia database. The traditional relational data model is not capable of handling synchronization aspects and the heterogeneous nature of the multimedia data. More powerful data modeling schemes are needed for this purpose. Recently, we have seen the emergence of various new data models which are capable of representing different characteristics of multimedia objects. The intent of this chapter is to discuss these models and their pros and cons in terms of suitability for developing multimedia databases.

*School of Electrical Engineering, Purdue University, West Lafayette, IN, 47907, U.S.A.

†CASE Center and Department of Electrical and Computer Engineering, syracuse University, Syracuse, NY, 13244, U.S.A.

We begin this chapter with a discussion on the characteristics of multimedia data. We then elaborate on the problem of multimedia synchronization in Section 2. The conceptual modeling of multimedia data is also discussed in this section. This is followed by an overview of the current state-of-the-art in multimedia databases. We conclude this chapter by describing the future trends and challenges in this area.

20.2 Characteristics of Multimedia Data

20.2.1 Text and Formatted Data

Formatted textual and numeric data will remain an essential part of any multimedia information system. Database systems based on relational, network or hierarchical models are well suited for managing such type of data and generally provide efficient access paths to data in secondary storage by maintaining some type of indexing scheme. Queries based on a number of key values such as partial match queries are supported by using intersection operations. Tree structure and hashing can be extended to multi-dimensional data in order to support such queries. Examples of such methodologies include k-d trees and multi-dimensional extensible hashing.

Each indexing scheme has its own advantages and disadvantages depending on the nature of query processing and data lifetime. When the database is static and most queries are based on few keys, an inverted index can be the best indexing scheme. On the other hand, when the database is frequently updated, and partial match queries based on a number of key values are common, compressed files [51] and most dynamic multidimensional file structures can provide better performance. Maintaining indices for large unformatted textual data requires an even more complicated strategy as the index file itself can easily grow beyond any manageable size when full text inversion is used for indexing. Although the access rate of this type data is not as crucial as the real-time deliverable data, such as video, audio etc., a large index file can prevent the database system from carrying out real-time updating.

20.2.2 Audio and Music Data

Due to their nature the audio waveform signals are usually sampled, encoded and then stored. In general, a high quality encoding scheme requires a great amount of storage space. Another way to provide users with the desired audio signal is to use some speech synthesis approach [619]. However, such an approach requires sophisticated processing and is usually slow. Compared to the speech signal, musical signals are much more regular and structural. This implies that forming an abstraction of music signals at a high level is possible. Graph model has been proposed for this abstraction [477], where common musical notations are used to represent the structures of the graph.

20.2.3 Images and Pictures Data

Many data structures have been proposed for storing images in a database system. These include pixel-oriented [110], quadtrees [484], R-trees [476] or vector based [304]. However, irrespective of the representation, the storage of this type of data is essentially in a digital form with the indexing scheme to provide 2-dimensional browsing. The digital information is generally stored in a compressed form. For the data that is

highly structured and formatted, high speed retrieval and storage techniques can also be employed in this case.

20.2.4 Full-Motion Video Data

These data are unique in their nature in the sense that they can be totally in analog form containing both video frames and associated audio signals or in digital form. The information can be stored in the same way as in video cassetts with the functions of replay, freezing frame, advancing etc. However, in order to integrate this information with other data they must be digitized in order to prepare composite digital data packets carrying requested multimedia object information for the user. Therefore, analog to digital conversion and compression of video data needs to be carried out by the server. The equivalent inverse function needs to be performed at the user end. This signal processing can be avoided if the video information is prestored in a digitized compressed form. In digital form data manipulation is much more flexible as compared to analog form. However, irrespective of the nature of information, this service requires an enormous capacity for storage and very high transfer rate.

From the data modeling point of view virtually nothing exists. Selection of attributes, indexing techniques, access and retrieval of video data are some of the open research issues in characterizing and managing video data.

20.3 Notion of Time for Multimedia Data

A multimedia object may contain real-time data like audio and video in addition to the usual text and image data that constitute present-day information systems. Real-time data can require time ordered presentation to the user. A composite multimedia object may have specific timing relationships among the different types of component media. Coordinating the real-time presentation of information and maintaining the time-ordered relations among component media is known as temporal synchronization. Assemblying information on the workstation is the process of spatial composition, which deals basically with the window management and display layout interface.

For continuous media, the integration of temporal synchronization functions within the database management system is quite desirable since it can make the storage and handling of continuous data more efficient for the database system. Also, implementation of some standard format for data exchange among heterogeneous systems can be carried out more effectively. In this section we first elaborate on the problem of temporal synchronization of multimedia data for composing objects, followed by a discussion of modeling time. These models are then used to develop conceptual models for the multimedia data, as described in a later section.

20.3.1 The Temporal Synchronization Problem

The concept of temporal synchronization is illustrated in Figure 20.1, where a sequence of images and text is presented in time to compose a multimedia object. One can notice, from this figure that the system must observe some time relationships (constraints) among various data objects in order to present the information to the user in a meaningful way. These relationships can be *natural* or *synthetically created* [369]. Simultaneous recording of voice and video through a VCR, is an example of *natural* relationship between audio and video information. A voice annotated slide show, on the

other hand, is an example of *synthetically created* relationship between audio and image information. In this case, change of an image and the end of its verbal annotation, represent a synchronization point in time.

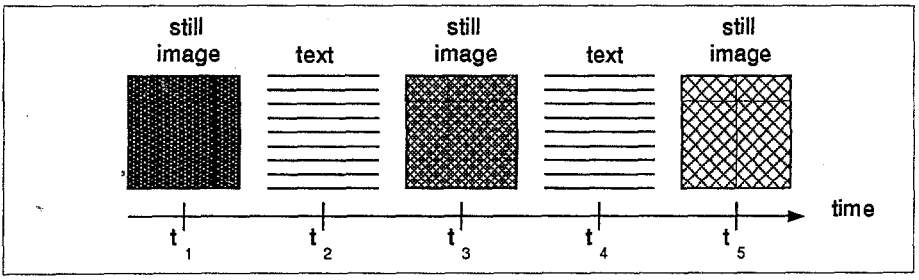


Figure 20.1: Time-Ordered Multimedia Data

A user can randomly, access various objects, while browsing through a multimedia information system. In addition to simple forward playout of time-dependent data sequences, other modes of data presentation are also viable, and should be supported by a multimedia database management system. These include reverse playout, fast-forward/fast-backward playout, and random access of arbitrarily chosen segments of a composed object. Although these operations are quite common in TV technology, (e.g., VCRs), these capabilities are very hard to implement in a multimedia system. This is due to the non-sequential storage of multimedia objects, the diversity in the features of hardware used for data compression, the distribution of data, and random communication delays introduced by the network. Such factors make the provision of these capabilities infeasible with the current technologies.

Conceptually, synchronization of multimedia information can be classified into three categories, depending upon the "level of granularity of information", requiring synchronization [568]. These are the physical level, the service level, and the human interface level [568], as shown in Figure 20.2.

At the physical level, data from different media are multiplexed over single physical connections or are arranged in physical storage. This form of synchronization can be viewed as "fine-grain". The service level synchronization is "more coarse grain", since it is concerned with the interactions between the multimedia application and the various media, and among the elements of the application. This level deals primarily with intermedia synchronization necessary for presentation or playout. The human interface level synchronization is rather "coarse grain" since it is used to specify the random user interaction to a multimedia information system such as viewing a succession of database items, also known as browsing.

In addition to time dependent relational classification (i.e., synthetic/natural), data objects can also be classified in terms of their presentation and application lifetimes. A persistent object is one that can exist for the duration of the application. A non-persistent object is created dynamically and discarded when obsolete. For presentation, a transient object is defined as an object that is presented for a short duration without manipulation. The display of a series of audio or video frames represents transient presentation of objects, whether captured live or retrieved from a database. Henceforth,

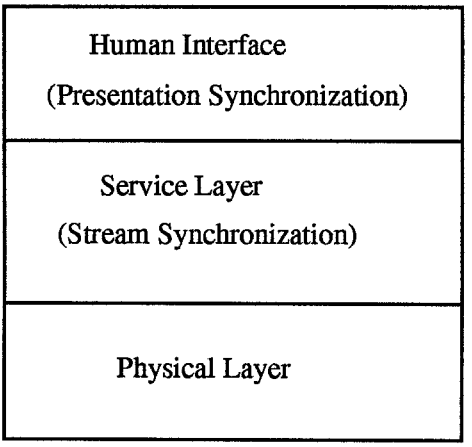


Figure 20.2: Levels of Synchronization of Multimedia Data

we use the terms static and transient to describe presentation lifetimes of objects while persistence expresses their storage life in a database.

In another classification, multimedia data have been characterized as either continuous or discrete [250]. This distinction, however, is somewhat vague since time ordering can be assigned to discrete media, and continuous media are time-ordered sequences of discrete ones after digitization. We use a definition attributable to Herrtwich [250], where continuous media are represented as sequences of discrete data elements that are played out contiguously in time. However, the term continuous is most often used to describe the fine-grain synchronization required for audio or video.

20.3.2 Modeling Time

The problem of multimedia synchronizing at presentation, user interaction, and physical layers, reduces to satisfying temporal precedence relationships among various data objects under real timing constraints. For such purpose, models to represent time must be available. Temporal intervals and instants provide a means for indicating exact temporal specification. In this section, we discuss these models and then describe various conceptual data models to specify temporal information necessary to represent multimedia synchronization.

To be applicable to multimedia synchronization, time models must allow synchronization of components having precedence and real-time constraints, and provide the capability for indicating laxity in meeting deadlines. The primary requirements for such a specification methodology include the representation of real-time semantics and concurrency, and a hierarchical modeling ability. The nature of presentation of multimedia data also implies that a multimedia system has various additional capabilities such as: to handle reverse presentation, to allow random access (at an arbitrary start point), to permit an incomplete specification of intermedia timing, to handle sharing of synchronized components among applications, and to provide data storage for control information. In light of these additional requirements, it is, therefore, imperative that

a specification methodology must also be well suited for unusual temporal semantics as well as be amenable to the development of a database for storing timing information.

The first time model is an instant-based temporal reference scheme that has been extensively applied in the motion picture industry, as standardized by the Society of Motion Picture and Television Engineers (SMPTE). This scheme associates a virtually unique sequential code to each frame in a motion picture. By assigning these codes to both an audio track and a motion picture track, intermedia synchronization between streams is achieved. This absolute, instant-based scheme presents two difficulties when applied to a multimedia application. First, since unique, absolute time references are assumed, when segments are edited or produced in duplicate, the relative timing between the edited segments becomes lost in terms of playout. Furthermore, if one medium, while synchronized to another, becomes decoupled from the other, then the timing information of the dependent medium becomes lost. This instant-based scheme has also been applied using Musical Instrument Digital Interface (MIDI) time instant specification [416]. The same scheme is used to couple each time code to a common time reference [252].

In another approach, temporal intervals are used to specify relative timing constraints between two processes. This model is mostly applicable to represent simple parallel and sequential relationships. In this approach, synchronization can be accomplished by explicitly capturing each of the thirteen possible temporal relations [369] that can occur between the processes. Additional operations can be incorporated in this approach to facilitate incomplete timing specification [250].

We now discuss how these approaches can be used to develop conceptual models for multimedia objects. We discuss pros and cons of these models and compare them in terms of their effectiveness to represent objects and user's data manipulation functions.

20.4 Conceptual Models for Multimedia Objects

A number of attempts have been made to develop conceptual models for representing multimedia objects. These models can be classified into five categories, namely; graphical models, Petri-Net based models, object-oriented models, language based models, and temporal abstraction models. Some models are primaly aimed at synchronization aspects of the multimedia data while others are more concerned with the browsing aspects of the objects. The former models can easily render themselves to an ultimate specification of the database schema, as briefly discussed later in this section. Some models, such as based on graphs and Petri-Nets have the additional advantage of pictorially illustrating synchronization semantics, and are suitable for visual orchestration of multimedia presentations. These models are discussed below.

20.4.1 Graphical Models

Labeled directed graphs have been extensively used to represent information [586]. Hypertext systems provide an example of such a mechanism. This approach allows one to interlink small information units (data) and provides a powerful capability for users to navigate through a database. Information in such a systems represents a "page" consisting of a segment of text, graphics codes, executable programs, or even audio/video data. All the pages are linked via a labeled graph, called hypergraph. The major

application of this model is to specify higher level browsing features of multimedia system. The essence of hypertext is a nonlinear interconnection of information, unlike the sequential access of conventional text. Information is linked via cross-referencing between keywords or subjects to other fragments of information. An application has been implemented [492] for interactive movies by using the hypertext paradigm.

Various operations, such as updating and querying, can be performed on a hypergraph. Updating means changing the configuration of the graph and the content of the multimedia data. Querying operations include navigating the structure, accessing pages (read or execute), showing position in the graph, and controlling side effects. Basically, it is a model for editing and browsing hypertext.

The hypergraph model suffers from many shortcomings. The major drawback is that there is no specific mechanism to handle temporal synchronization among data items.

20.4.2 Petri-Net Models

Recently, the use of Petri-Nets for developing conceptual models and browsing semantics of multimedia objects [145, 369, 558] has been proposed. The basic idea in these models is to represent various components of multimedia objects as places and describe their inter-relationships in the form of transitions. These models have been shown to be quite effective for specifying multimedia synchronization requirements.

For example, one such model is used to specify high level (object level) synchronization requirements which is both a graphical and mathematical modeling tool capable of representing concurrency. In this approach Timed Petri Net has been extended to develop a model that is known as Object Composition Petri Nets (OCPNs) [369]. The particularly interesting features of this model are the ability to explicitly capture all the necessary temporal relations, and to provide simulation of presentation in both the forward and reverse directions. Each place in this Petri-Net derivative represents the playout of a multimedia object while transitions represent synchronization points.

In [369], thirteen temporal relationships between two objects are presented, which are sufficient to specify temporal composition of any complex multimedia object. An OCPN model can represent all these relations. It has been shown in the paper that an arbitrarily complex process model of temporal relations can be constructed with an OCPN. Figure 20.3 shows an example of an OCPN that describes a slide show. As can be noticed, in this model the duration of each object is also specified.

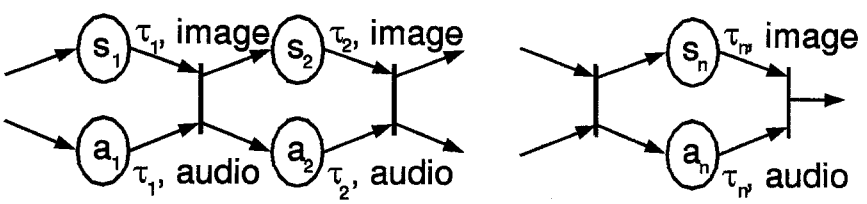


Figure 20.3: AN OCPN Model for A Slide Show

In another model, called Petri-Net-Based-Hypertext (PNBH), the higher level browsing semantics can be specified. In this model information units are treated as

net places and links as net arcs. Transitions in a PNBH indicate the traversal of links, or the browsing of information fragments. Figure 20.4 illustrates a PNBH model consisting of segments of an interactive movie. These segments can be played-out in a random order, as selected by the user and restricted by the semantics of the net.

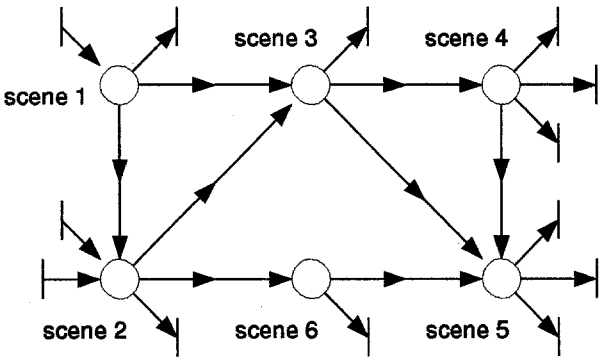


Figure 20.4: PNBH Petri Net

Unlike the OCPN, net places in PNBH can have multiple outgoing arcs, and therefore can represent nondeterministic and cyclic browsing. On the other hand, the OCPN specifies exact presentation-time playout semantics, useful in real-time presentation scheduling. Clearly these two models complement each other for specifying both user interaction and presentation orchestration.

The above mentioned Petri-Net based models lack one capability. That is, there is no mechanism to specify communication requirements and control functions for distributed composition of objects. An attempt has been made in [145] where another Petri Net based hierarchical model, called G-Net, has been proposed. This model does allow specifications of communication primitives and types of connections that can be established among communicating sites. However, the model is rather over simplified since, unlike OCPN, it does not facilitate generation of database schema in a straight forward manner for the multimedia objects. On the other hand, the Petri-Net models, especially the OCPN, can be easily extended to model databases. For this purpose, a place represents an object. Attributes, operations on objects, pointers to physical data, etc. can be associated with each object. The semantics of the schema, therefore, can be represented by the net structure. This conversion is explained at the end of this section.

20.4.3 Object-Oriented Models

The basic idea in this model is to represent a real world thing or concept as an object. An object usually has an identifier, attributes, methods, a pointer to data, etc. One such approach has been proposed in the OMEGA system [390]. To facilitate the presentation of multimedia objects, OMEGA uses temporal information associated with each object to calculate precedence and synchronization between objects. In this model, a multimedia object has attributes, relationships which are its value reference to other objects, components (its value reference to other object(s) that are dependent on the

referring superordinate object), and methods. Some integrity rules also apply. These include class, instance, subclass, superclass, inheritance, generalization, and aggregation. For instance, in OMEGA systems, IS_PART_OF and IS_ REFERENCE_OF can be specified between objects. In order to handle different types of multimedia data, a metaclass called $multimedia_{class}$ is defined in the OMEGA system that consists of sound, image, text, and video class.

In another approach [45], a three dimensional class hierarchy is used to represent spatial objects. Definition and manipulation (usually specified in methods) are operations applied to objects.

This approach can facilitate the generation of multimedia database schema, by using either programming languages or ER diagrams.

20.4.4 Language Based Models

Concurrent languages have been extensively used to specify parallel and distributed process structures. These languages also have the potential to specify multimedia synchronization requirements. For one such scheme, an extension to the language called Communicating Sequential Processing (CSP) has been proposed. This extension supports multimedia process synchronization, including semantics for real-time synchronization of multimedia data. The extension is based on a proposed concept, called "restricted blocking" which provides a resolution mechanism for the synchronization problem encountered while handling continuous media [548]. In the "restricted blocking" mode, an object may be forced to wait for an other object, to perform synchronization if the later does not arrive in time. For this purpose, the extention to CSP includes various constructs such as SYNCHRONIZE, WITH object-name, AT end, MODE type-of-blocking, WHILE_WAITING do-something. In this command, an object is forced to wait for the other object to arrive for synchronization. For this purpose, the parmater for the MODE (type-of-blocking) primitive can be set as *restricted_blocking*. The waiting object can be replayed or slowed down during the wait state. These constructs are initiated by the system. Also, various time operands can be specified in this command to adjust the relative display time of two objects.

Various other language-based approaches have also been proposed. Two such examples include the specification using LOTOS (Language Of Temporal Ordering Specification) [605], and process-oriented synchronization in CCWS [453].

The major advantage of language based models is that they can directly lead to an implementation. However, their drawback is that, unlike graphical models, they are hard to conceptually visualize and are difficult to verify.

20.4.5 Temporal Abstraction Models

Some of the requirements for multimedia presentation are not well described by either of the above mentioned models. For example, to reduce (slow motion) or increase (fast-forward) the speed of a multimedia presentation, the temporal models are deficient. These requirements can be addressed by temporal abstractions, which are means to manipulate or control the presentation of a temporal specification via time reference modification. Various virtual time abstractions have been described in the literature [250, 18]. These describe the maintenance of a time reference that can be scaled to real-time and adjusted to appropriate playout speeds. If real-time is defined as nominal clock time as we perceive it, then virtual time is any other time reference system suitable for translation to real-time. For example, a unitless reference can be converted, or

projected to real-time system by any scaling or offsetting operations. In this manner, the output rate and direction for a sequence of data elements can be changed by simply modifying this translation, i.e., an entire temporal specification, either language or graph-based, can track a specific time reference or translation process.

20.4.6 Database Models for Multimedia Synchronization

Once temporal specifications of time-dependent multimedia objects are effectively modeled, a multimedia information system must have the capability for storing and accessing these objects. This problem is distinct from historical databases, temporal query languages [529, 573], or time-critical query evaluation [256]. Unlike historical data, time-dependent multimedia objects require special considerations for presentation due to their real-time playout characteristics. Data need to be delivered from storage based on a prespecified schedule, and presentation of a single object can occur over an extended duration (e.g., a movie).

A conceptual data model for time-dependent multimedia objects must support forward and reverse playout as well as random access (in time) to the object in addition to conventional DBMS queries. Temporal intervals can be described either by a timeline representation in an unstructured format, or in a structured format such as the OCPN. Using the OCPN, temporal hierarchy can be imparted to the conceptual schema as sets of intervals bound to a single temporal relation.

With this approach, the conceptual schema forms a temporal hierarchy representing the semantics of the OCPN, as shown in Figure 20.5. Subsets or subtrees of this hierarchy represent subnets of the OCPN, illustrating the capability of composing complex multimedia presentations. Terminal elements in this model indicate base multimedia objects (audio, image, text, etc.), and additional attributes can be assigned to nodes in the hierarchy for conventional DBMS access. Timing information is also captured with node attributes, allowing the assembly of component elements during playout.

As mentioned above, temporal information can also be encapsulated in the description of the multimedia data using the object-oriented paradigm [390]. Temporal information including a time reference, playout time units, temporal relationships, and required time offsets are maintained for specific multimedia objects. For stream type data, this approach can define the time dependencies for an entire sequence by defining the period or frequency of playout (e.g., 30 frames/s for video) analogous to a set of intervals bound to a single temporal relation.

Given an application, the synchronization requirements then can be described by an OCPN model. The OCPN is then transformed into a database schema. A tree (network) structure is used to represent the schema. This is depicted in Figure 20.6. As can be noticed from this figure, three types of nodes are used to capture the information in OCPN. The first is a terminal node. It has attributes that indicate node type, media type, an unspecified field, and a pointer indicating the location of data. Nonterminal nodes have the following attributes : node type, an unspecified field, left and right child pointers and temporal data. The third type of nodes is the meta node, which is similar to a nonterminal node except that it has n (≥ 2) child pointers.

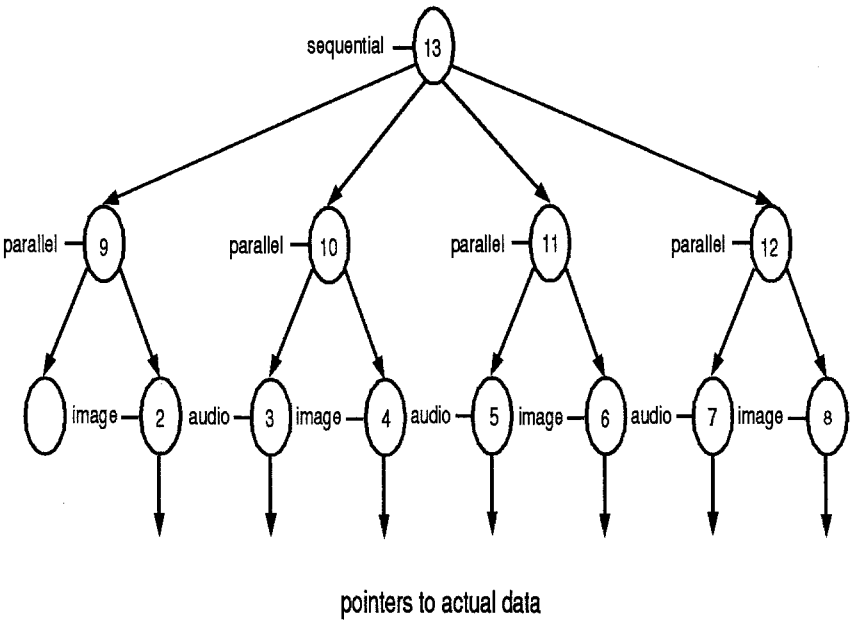


Figure 20.5: A Temporal Hierarchy for the Semantics of the OCPN

20.5 Some Multimedia Database Systems

In this section we discuss various multimedia databases. We start with an introduction to image database, followed by a description of audio databases.

20.5.1 Image Database

Conventional database management systems have been designed for managing numeric and text data, which are single dimensional in nature. But spatial or pictorial data are necessarily two- or three-dimensional and contain a considerable amount of information that cannot be stored directly as a set of linear relationships. The spatial or two-dimensional image data have a great deal of implicit and explicit knowledge. Explicitly, there is sensory information pertaining to the lighting-brightness, shadows, colors, etc. Implicit in all the spatial data is the information related to the concept of position as well as the notion of distance.

What makes the management of spatial data very complicated is that the same image can be viewed in different perspectives by various users. Consequently, the primary features that need to be recognized and extracted can be different for each problem domain. Moreover, there is a need to be able to represent image data of various kinds. For example, a geographical map would directly pertain to two-dimensional information, whereas a machine design drawing contains information about three-dimensional objects. In addition, in order to support a large variety of continuous and discrete representations, an image database provides access to special-purpose functions for image

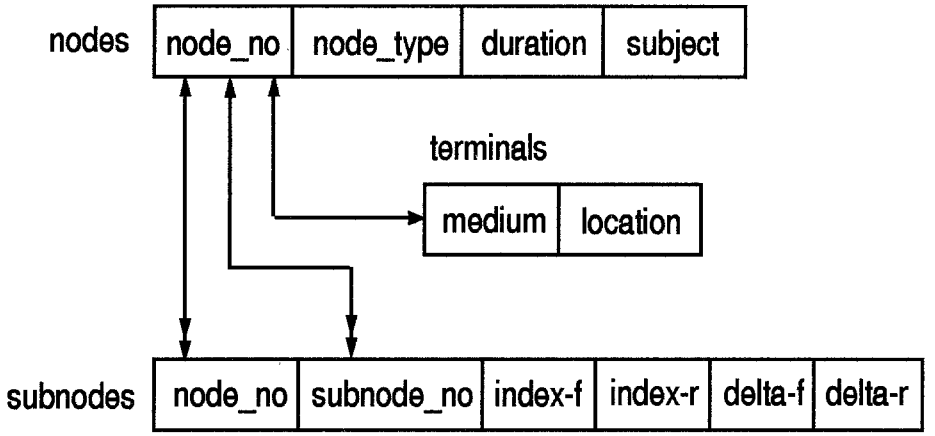


Figure 20.6: A Network Database Schema for the OCPN

processing such as primitives for feature extraction. In order to make progress in image databases from the software engineering perspective, it is important to examine innovations in both processing and representations of images.

One of the most important problems to be considered in the design of image database systems is how images are stored in an image database. Many data structures have been proposed. Some are pixel-oriented; some utilize quadtrees, or R-trees, and some are vector based. To make an image database system more intelligent, more flexible, and an efficient data structure should be used. Also, the knowledge embedded in images should be captured by the data structure as much as possible, especially spatial knowledge. Extracting information from images is a time consuming process. On the other hand, if information or knowledge is extracted from images item by item in advance and stored for later retrieval, we need much more storage capacity and therefore, retrieval would take a long time.

In pictorial information retrieval, many approaches have been proposed and include relational database queries, query-by-view [412], quadtrees etc. We now describe some image database systems for which such approaches have been proposed and prototyped.

1. Intelligent Image Database System (IIDS) [103]: This is a prototype intelligent image database system that is based on a new pictorial data structure. Specifically, a new way of representing a picture by a 2-D string has been introduced in IIDS. A picture query can also be specified as a 2-D string. The problem of pictorial information retrieval then becomes one of 2-D string subsequence matching. This approach allows an efficient and natural way to construct iconic indexes for pictures. The 2-D string representation is ideally suited to formulating picture queries. The iconic index can not only be used in pictorial information retrieval, but also provides an efficient means for picture browsing. The corresponding 2-D string for this kind of query contains two special icons. One is called single-variable icon, which can match any single object; the other is called multivariable icon, which can match any set of objects.
   In order to increase the power of the IIDS, it would be advantageous to add

attributes to symbols in the 2-D strings. In this way, we could convey not only the relative positions of objects, but also information such as orientation, size, and other characteristics.

2. Image Database System (IDB) [591]: Most information management systems are designed to handle traditional alphanumeric data. Today technology makes available resources that allow the management of new classes of information, such as image and voice. When dealing with images, we must generalize the input, processing, and output phases that characterize the management of traditional data types. Furthermore, all these activities require suitable hardware and software instruments. From the end-user point of view, the interface between users and systems would be much more attractive if it were possible to use images to manage images as well as we use words to manage traditional information. Two kinds of data must be managed by an image system: image files and their descriptions. The former are characterized by large sizes and unstructured forms, while the latter have small sizes and structured forms. Images and descriptions are stored on different kinds of devices. The main requirement for image data is the availability of a large memory at low cost. Optical disks can meet this requirement, and image data are now increasingly stored on those special devices, while descriptive information continues to be stored on magnetic disks.
   The IDB system exploits images as a vehicle of interaction with the user; index images play a fundamental role in completing the selection of images from the archive. The architecture is characterized by modularity and flexibility; each single module is related to a specific task to be performed during the image management process. Functions have been integrated by distributing resources among the nodes of a LAN; each node corresponds to a workstation, and many users can work with the system.
   Future extensions of IDB will involve the integration of new kinds of information such as audio data and image animation. A hypermedia approach is also being evaluated.

3. Map Database: A map database management system contains facilities to create, modify, store, and retrieve spatial information. A Map Database System (MDS) goes beyond simply replacing paper maps. MDS allows users to view, compare, and analyze spatial relationships. Map databases allow the generation of maps that contain only the information required by the map user. The map information is divided into different layers which overlay on the same area. Typical layers include streams, cities, sewers, roads, highways, secondary streets, water pipes, gas lines, telephone cables and so on. The information in layers may also contain per capital income, product consumption, or other thematic information.
   Map databases contain large amounts of data. Efficient encoding of the graphical information into a format suitable for digital storage is required. The nongraphical attributes are usually stored using the normal methods. Several different encoding methods are polygon encoding, dual independent map encoding, and 2-D encoding [236].

20.5.2 Audio Database

Until now, there seems to be no practical audio database management system even though the technology for audio acquisition is available. In fact, an audio database

can be reduced to an audio storage system. However, in order to support real time applications in multimedia representation, an efficient placement of audio data on optical disks is important. The placement of audio data on the optical disk is of primary importance because the audio data is being extracted from the optical disk in real-time. Real time means that system interrupts greater than 30 msec cannot be tolerated. Yu et al. [637] proposed an optimal algorithm for merging two audio records, resulting in a record having minimum length. This result is useful in minimizing space for storing audio files, while maintaining realistic sound.

20.6 Challenges in Multimedia Database

Conventional databases are mainly designed to process and access the text and numeric data. These databases can be an important part of the overall multimedia system. Considerable research has been done during the last two decades in these databases. Their architecture ranges from centralized database to distributed database systems. However, all these conventional database systems lack the capabilities needed to support advanced multimedia applications in offices and factories, that require integration of various types of data such as text, image, video and audio into a single object so that users can interact with it without being aware of the heterogeneity of data in types and operations.

The organization and management issues for multimedia databases have been the subject of extensive research and development since the middle of the 1980's. However, a number of challenges are faced by the database community to provide a comprehensive solution for designing and managing multimedia database systems. These include designing new data models to capture semantics for multimedia objects, storing and accessing multimedia data, indexing techniques for digital images, video and audio data, version management for distributed objects, query language development for multimedia data etc. Some approaches proposed for this purpose are based on the extension of existing relational, networking, and object-oriented models, as briefly described earlier.

Coming back to four issues discussed in Section 1, we have only addressed the first issue in this chapter. Extensive research is needed to handle other issues as well. For example, the base technology that deals with the management of various data types poses a number of challenges. The major issue is related to storing live data such as digital video and audio. Specifically, if we need to extend the data models used in conventional databases, selection of attributes, designing suitable indexing schemes, searching for video and audio data items, linking objects, etc., pose unique challenges that are not faced in conventional databases. It is possible that a mere extension of existing data models may not prove fruitful for this purpose. More advanced data models, as discussed above, may provide better and more powerful methodologies and warrant more extensive scrunity and evaluation.

Another major challenge is integration of enabling technologies, where diverse and heterogeneous (such as text, image, video etc.) databases need to interact with each other in order to provide unified composite objects to the end users. Schema integration for the individual multimedia databases, provision of a high level query language to manipulate objects across these databases etc., are some of the important issues which need to be addressed. A number of papers have recently started appearing that provide more powerful query languages which can be effectly used for multimedia databases [412]

20.7 Conclusion

One of the requirements of multimedia database systems is that they will need a data model more powerful than the relational model, without compromising its advantages. The relational data model exhibits limitations in terms of complex object support, type system, and object management. To address these issues, we have emphasized one key requirement for multimedia databases, which is the process of temporal synchronization. We have discussed various conceptual models to specify this requirement and have highlighted their capabilities and limitations. Still, a number of other issues need to be faced before multimedia database systems become a reality.

Bibliography

[1] I. Aalbersberg and F. Sijstermans. High-quality and High Performance Full-Text Document Retrieval: the Paralell infoguide System. In *Proceedings of the IEEE Conference on Parallel and Distributed Information Systems*, pages 142–150, December 1991.

[2] S. Abiteboul, P. Buneman, C. Delobel, R. Hull, P. Kanellakis, and V. Vianu. New Hope on Data Models and Types: Report of an NFS-INRIA Workshop. *ACM SIGMOD*, 19(4), December 1990.

[3] S. Abiteboul and C. Kanellakis. Object Identity as a Query Language Primitive. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 159–173, Portland, Oregon, June 1989.

[4] ACM91. Special Issue on Next-Generation Database Systems. *Communications of ACM*, 34(10), October 1991.

[5] N. Adam and R. Subramanian. An Object-Oriented Model for Intelligent Query Processing in Geographical Databases. In *Proc. 3rd Australian Database Conference*, 1992.

[6] R. Agarwal, N. H. Gehani, and J. Srinivasan. OdeView: The Georaphical Interface to Ode. In *Proc. ACM-SIGMOD Conf. on Mangement of Data*, Atlantic City, New Jersey, May 1990.

[7] R. Agrawal and N.H. Gehani. Ode(Object Database and Environment): the Language and the Data Model. *Proc. ACM-SIGMOD 1989 Int'l Conf. Management of Data*, pages 36–45, May 1989.

[8] R. Ahad et al. Supporting Access Control in an Object-Oriented Database Language. In *Proc. Third International Conference on Extending Database Technology (EDBT), Vienna (Austria), Springer-Verlag Lecture Notes in Computer Science*, volume 580, pages 184–200, 1992.

[9] R. Ahmed and et al. The Pegasus Heterogeneous Multidatabase System. *IEEE Computer*, 24(12), December 1991.

[10] T.H. Ahn, H.J. Jo, J.H. Kim, Y.J. Lee, and B.C. Kim. Statistical and Scientific Database Management. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science 420*. Springer-Verlag, 1990.

[11] A. Alashqur, S. Su, and H. Lam. OQL-A Query Language for Manipulating Object-Oriented Databases. In *International Conference on Very Large Databases*, pages 433–442, 1989.

[12] A. Alashqur, S. Su, and H. Lam. A Rule-Based Language for Deductive Object-Oriented Databases. In *Proceedings of the Sixth Int'l Conference on Data Engineering*, 1990.

[13] J. F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of ACM*, 26(11):832–843, November 1983.

[14] L. Allen and O. Frieder. Exploiting Database Technology in Medical Arenas: A critical Assessment of PACS. *IEEE Engineering in Medicine and Biology*, 11(1):42–49, March 1992.

[15] R. Alonso and D. Barbara. Negotiating Data Access in Federated Database Systems. *Proceedings of the 5th International Conference on Data Engineering*, February 1989.

[16] R. Alonso, D. Barbara, and H. Garcia-Molina. Data Caching Issues in an Information Retrieval System. *ACM Transactions on Database Systems*, 15(3):359–384, September 1990.

[17] David P. Anderson and George Homsy. A Continuous Media I/O Server and Its Synchronization Mechanism. *IEEE Computer*, 24(10):51–57, October 1991.

[18] D.P. Anderson, S.Y. Tzou, R. Wahbe, R. Govindan, and M. Andrews. Support for Continuous Media in the Dash System. *Proc. 10th Intl. Conf. on Distributed Computing Systems*, pages 54–61, May 1990.

[19] Apollo Computer Inc., Chelmsford, MA. *Network Computing System: A Technical Overview*, 1989.

[20] W.G. Aref and H. Samet. Extending a DBMS with Spatial Operations. *Lecture notes in Computer science*, pages 299–318, August 1991. Vol. 525.

[21] G. Ariav. A Temporally Oriented Data Model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[22] H. Armbruster. Broadband ISDN - The Network of the Future: Applications and Compliance with user Requirements. *Globecom '86 (IEEE Global Telecommunications Conference Record 1986)*, pages 484–490, December 1986.

[23] N. Asokan, S. Ranka, and O. Frieder. High-quality and High Performance Full-Text Document Retrieval: the Paralell infoguide System. In *Proceedings of the IEEE International Conference on Parallel Architectures and Databases*, pages 519–521, 1990.

[24] B. R. Badrinath and K. Ramamritham. Semantics-Based Concurrency Control Beyond Commutativity. In *3rd IEEE International Conference on Data Engineering*, pages 304–311, 1987.

[25] Charles W. Bailey. Intelligent Multimedia Computer Systems: Emerging Information Resources in the Network Environment. *Library Hi Tech*, 8(1):29–41, 1990.

[26] S.C. Bailin. An Object Oriented Requirements Title Method. *Communications of the ACM 32,5*, 1989.

[27] B. Baker, E. Coffman, and R. Rivest. Orthogonal Packings in Two Dimensions. *SIAM Journal on Computing*, 9(4):846–855, November 1980.

[28] R. Balzer et al. Operational Specifications as the Basis for Rapid Prototyping. In *In Proceedings of ACM SIGSOFT '88: Third Symposium on Software Development Environments, Boston*, pages 3–16, December 1982. Software Engineering Notes 7(5).

[29] R. Balzer et al. Common Prototyping Working Group. Draft report on requirements for a common prototyping system, University of Maryland, 1988.

[30] Robert Balzer, Dewayne E. Perry, James Purtilo, Richard T. Snodgrass, Alexander L. Wolf, and Jon Ward. Technical Notes 1–9, September 1990–November 1991. Technical report, DARPA Module Interconnection Formalism Working Group, 1991.

[31] F. Bancilhon and R. Ramakrishnan. An Amateur's Introduction to Recursive Query Processing. In *ACM-SIGMOD*, Washington, D.C., 1986.

[32] Francois Bancilhon and Won Kim. Object-Oriented Database Systems: In Transition. *SIGMOD RECORD*, 19(4), December 1990.

[33] J. Banerjee, H.T. Chou, J.F Garza, and W. Kim. Data Model Issues for Object-Oriented Applications. *ACM Transactions on Office Information Systems*, 5(1):3–26, January 1987.

[34] D. Barbara and H. Garcia-Molina. The Demarkation Protocol: A Technique for Maintaining Arithmetic Constraints in Distributed Systems. In *Proc. of the Int'l Conference on Extending Data Base Technology*, Vienna, March 1992.

[35] Daniel Barbara and Chris Clifton. Information Brokers: Sharing Knowledge in a Heterogeneous Distributed System. Technical Report MITL-TR-31-92, Matsushita Information Technology Laboratory, 182 Nassau STreet, Princeton, NJ 08542, October 1992.

[36] R. Barrera and A. Buchmann. Schema Definition and Query Language for a Geographical Database System. *IEEE Transactions on Computer Architecture:Pattern Analysis and Image Database Management*, November 1981.

[37] C. K. Baru and O. Frieder. Database Operations in a Cube-Connected Multiprocessor System. *IEEE Transactions on Computers*, 38(6):920–927, June 1989.

[38] C. Batini, M. Lenzerini, and S.B. Navathe. A comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4), December 1986.

[39] N. Beckman and H.P. Kriegel. The R^*-Tree:An Efficient and Robust Access Method for Points and Rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, May 1990.

[40] C. Beeri and T. Milo. A Model for Active Object Oriented Database. In *Proc. of the 17th International Conf. on Very Large Databases*, pages 337–349, Sept. 1991.

[41] K. Belkhale and P. Banerjee. Approximate Scheduling Algorithms for the Partitionable Independent Task Scheduling Problem. In *Proceedings of the 1990 International Conference of Parallel Processing*, volume I, pages 72–75, August 1990.

[42] Nicholas J. Belkin and W. Bruce Croft. Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM*, 35(12):29–38, Dec. 1992.

[43] D. E. Bell and L. J. LaPadula. *Secure Computer Systems: Unified Exposition and Multics Interpretation.* The Mitre Corp., March 1976.

[44] Douglas L. Bell and Alan Kaplan. *PIC/ADA Toolset User Reference Manual.* University of Massachusetts, Software Development Laboratory, Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts 01003, arcadia design document um-91-04 edition, October 1991.

[45] D. Benson and G. Zick. Symbolic and Spatial Database for Structural Biology. *Proc. OOPSLA 91*, pages 329–339, 1991.

[46] B. Bergsten, M. Couprie, and M. Lopez. DBS3: A Parallel Database System for Shared Store. *Proceedings of the 2nd Intl. Conf. on Parallel and Distributed Information Systems*, pages 260–262, January 1993.

[47] T.J. Berners-Lee, R. Cailliau, R.-F. Groff, and B. Pollermann. World- Wide Web: An Information InfraStructure for High-Energy Physics. In *Proc. Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics*, La Londe-les-Maures, France, January 1992.

[48] P. Bernstein, M. Hsu, and B. Mann. Implementing Recoverable Requests Using Queues. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1990.

[49] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.

[50] P. Bruce Berra, C. Y. Roger Chen, Arif Ghafoor, Chin Chung Lin, Thomas D. C. Little, and Donghoon Shin. Architecture for Distributed Multimedia Database Systems. *Computer Communications*, 13(4):217–231, May 1990.

[51] P.B. Berra, S.M. Chung, and N. Hachem. Computer Architecture For a Surrogate File to a Very Large Data/Knowledge Base. *IEEE Computer*, 20(3):25–32, March 1987.

[52] Brian N. Bershad, Dennis T. Ching, Edward D. Lazowska, Jan Sanislo, and Michael Schwartz. A Remote Procedure Call Facility for Interconnecting Heterogeneous Computer Systems. *IEEE Transactions on Software Engineering*, 13(8), August 1987.

[53] E. Bertino. Data Hiding and Security in an Object-Oriented Database System. *Proc. Eighth IEEE International Conference on Data Engineering, Phoenix (Ariz.)*, Feb 1992.

[54] E. Bertino and S. Jajodia. Modeling MultiLevel Entities Using Single-Level Objects. Submitted for publication, August 1992.

[55] E. Bertino and L. Martino. Object-Oriented Database Management Systems: Concepts and Issues. *IEEE Computer*, 24(No.4):33–47, 1991.

[56] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-Oriented Query Languages: the Notion and the Issues. *IEEE Trans. on Knowledge and Data Engineering*, 4(3):223–237, 1992.

[57] E. Bertino and H. Weigand. An Approach to Authorization Modeling in Object-Oriented Database Systems. To appear in Data and Knowledge Engineering.

[58] Elisa Bertino and Won Kim. Indexing Techniques for Queries on Nested Objects. *IEEE Transactions on Knowledge and Data Engineering*, 1(2), June 1989.

[59] B. Bhargava, S. Browne, and J. Srinivasan. Composite Object Replication in Distributed Database Systems. In *Proc. International Conference on Information Systems and Management of Data, Bangalore, India*, July 1992.

[60] B. Bhargava, Prasun Dewan, James G. Mullen, and Jagannathan Srinivasan. Implementing Object Support in the RAID Distributed Database System. In *Proceedings Of The First International Conference on Systems Integration*, pages 368–377, April 1990.

[61] B. Bhargava and John Riedl. The RAID Distributed Database System. *IEEE Transactions on Software Engineering*, 16(6), June 1989.

[62] A. Bhide. An Analysis of Three Transaction Processing Architectures. *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 339–350, August 1988.

[63] A. Bhide. Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distributions of Column Values. *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 240–251, August 1988.

[64] A. Black, N. Hutchinson, E. Jul, and H. Levy. Object Structure in the Emerald System. Technical Report 86–04–03, University of Washington, University of Washington, Department of Computer Science, April 1986. 1986.

[65] D. C. Blair and M. E. Maron. An Evaluation of Retrieval Effectiveness for a Full-Text Document-Retrieval System. *Communications of ACM*, 28(3):289–299, May 1985.

[66] J.A. Blakely, P-A. Larson, and F. W. Tompa. Efficiently Updating Materialized Views. In *Proc. ACM-SIGMOD 1992 Int'l Conf. on Management of Data*, 1986.

[67] B. T. Blaustein and C. W. Kaufman. Updating Replicated Data During Communication Failures. In *Proceedings of the 11th International Conference on VLDB*, pages 49–58, 1985.

[68] J. Blazewicz, M. Drabowski, and J. Weglarz. Scheduling Multiprocessor Tasks to Minimize Schedule Length. *IEEE Transactions on Computers*, C-35(5):389–393, May 1986.

[69] S. H. Bokhari. On the Mapping Problem. *IEEE Transactions on Computers*, 30(3):207–214, March 1981.

[70] S. W. Bollinger and S. F. Midkiff. Processor and Link Assignment in Multicomputers Using Simulated Annealing. In *Proceedings of the 1988 International Conference on Parallel Processing*, pages 1–7, 1988.

[71] G. Booch. *Object-Oriented Design with Applications*. Benjamin-Cummings, 1990.

[72] H. Boral, W. Alexander, et al. Prototyping Bubba, A Highly Parallel Database System. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):4–24, March 1990.

[73] G. Bordogna, P. Carrara, and G. Pasi. Query Term Weights as Constraints in Fuzzy Information Retrieval. *Information Processing and Management*, 27(1):15–26, 1991.

[74] A. Borgida et al. Knowledge Representation as a Basis for Requirements Title. *IEEE Computers 18,4*, 1985.

[75] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas. An Overview of PCTE and PCTE+. *Communications of the ACM*, 1988.

[76] A. Bouguettaya, R. King, and K. Zhao. FINDIT: A Server Based Approach to Finding Information in Large Scale Heterogeneous Databases. In *First International Workshop on Interoperability in Multidatabase Systems, Kyoto, Japan*, pages 191–194, April 7–9 1991.

[77] M. Bouzeghoub and E. Metais. Semantic Modelling of Object-Oriented Databases. In *Proceedings of the 17th VLDB Conference*, September 1991.

[78] G. Brajnik, G. Guida, and C. Tasso. User Modeling in Expert Man - Machine Interfaces: A Case Study in Intelligent Information Retrieval. *IEEE Transactions on Systems, Man, and Cybernetics*, 20(1):166–185, January 1990.

[79] R. Breitl et al. The GemStone Data Management System. *Object-Oriented Concepts, Databases and Applications, W. Kim, and F. Lochovsky, eds., Addison-Wesley*, pages 283–308, 1989.

[80] M. Brodie and D. Ridjanovic. On the Design and Specification of Database Transactions. In M. Brodie, J. Mylopoulos, and Schmidtn J.W., editors, *On Conceptual Modeling*. Springer-Verlag, New York, 1983.

[81] Michael L. Brodie. The Promise of Distributed Computing and the Challenges of Legacy Information Systems. In *Proceedings of the IFIP DS–5 Semantics of Interoperable Systems Workshop*, page 31, November 16–20 1992.

[82] A. P. Buchmann. Modeling Heterogeneous Systems as an Active Object Space. Technical report, GTE Laboratories, Inc., 40 Sylvan Road, Waltham MA 02254, 1987.

[83] Paul Butterworth, Allen Otis, and Jacob Stein. The GemStone Object Database Management System. *Communicatons of the ACM*, 34(10), October 1991.

[84] Eick C., J. Liu, and P. Werstein. Integration of Rules into a Knowledge Base Management System. In *First International Conference on Systems Integration*, 1990.

[85] Ellis C.A. and Gibbs S.J. *Active Objects: Realities and Possibilities*. in Object-Oriented Concepts, Databases, and Applications, ACM Press, 1989.

[86] Michael Carey and Laura Haas. Extensible Database Management Systems. *SIGMOD RECORD*, 19(4), December 1990.

[87] Michael J. Carey, David J. DeWitt, Daniel Frank, Goetz Graefe, M. Muralikrishna, Joel E. Richardson, and Eugene J. Shekita. The Architecture of the EXODUS Extensible DBMS. In *International Workshop on Object-Oriented Database Systems*, pages 52–65, 1986.

[88] Yves Caseau. The LAURE System: Documentation. *Bellcore TM-ARH*, 1990.

[89] T. Catarci and G. Santucci. GRASP: A Graphical System for Statistical Databases. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science 420*. Springer-Verlag, 1990.

[90] S. Ceri and J. Widom. Deriving Production Rules for Constraint Maintenance. In *Proceedings of the Sixteenth International Conference of Very Large Data Bases, Brisbane, Australia*, pages 567–577, August 1990.

[91] S. Ceri and J. Widom. Production Rules in Parallel and Distributed Database Environme nts. In *18th VLDB*, pages 339–351, Vancouver, British Columbia, 1992.

[92] U.S. Chakravarthy. Rule Management and Evaluation: An Active DBMS Perspective. *SIGMOD RECORD*, 18(3):20–28, 1989.

[93] U.S. Chakravarthy and J. Minker. Multiple Query Processing in Deductive Database. In *12th International Conference on Very Large Databases*, Kyoto, Japan, 1986.

[94] L. Chambers and D. Cracknell. Parallel Features of NonStop SQL. *Proceedings of the 2nd Intl. Conf. on Parallel and Distributed Information Systems*, pages 69–70, January 1993.

[95] A.K. Chandra and D. Harel. Structure and Complexity of Relational Queries. *Journal of Computer Systems and Sciences*, 25(1):99–128, 1982.

[96] Chin-Chen Chang and Tzong-Chen Wu. Retrieving the Most Similar Symbolic Pictures from Pictorial Databases. *Information Processing and Management*, 28(5):581–588, 1992.

[97] E.E. Chang and R.H. Katz. Inheritance in Computer-Aided Design Databases - Semantics and Implementation Issues. *Computer-Aided Design*, 22(3):489–499, October 1990.

[98] N. S. Chang and K. S. Fu. Query by Pictorial Examples. In *Proc. Compsac*, 1979.

[99] N. S. Chang and K. S. Fu. Pictorial Query Languages for Pictorial Databases. *IEEE Computer*, 1981.

[100] N. S. Chang and K. S. Fu. Picture Indexing and Abstraction Techniques for Pictorial Databases. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1984.

[101] N.S. Chang and K.S. Fu. Query-by-Pictorial-Example. *IEEE Transactions on Software Engineering*, November 1980.

[102] N.S. Chang and K.S. Fu. Picture Query Languages for Pictorial Database Systems. *Computer.*, November 1981.

[103] S.K. Chang, C.W. Yan, Donald C. Dimitroff, and Timothy Arndt. An Intelligent Image Database System. *IEEE Trans. Software Engineering*, 14(5):681–688, May 1988.

[104] M.-S. Chen, H.-I. Hsiao, and P. S. Yu. Applying Hash Filters to Improving the Execution of Bushy Trees. *Proceedings of the 19th International Conference on Very Large Data Bases*, August 1993.

[105] M.-S. Chen, M.-L. Lo, P. S. Yu, and H. C. Young. Using Segmented Right-Deep Trees for the Execution of Pipelined Hash Joins. *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 15–26, August 1992.

[106] M.-S. Chen and K. G. Shin. Processor Allocation in an N-Cube Multiprocessor Using Gray Codes. *IEEE Transactions on Computers*, C-36(12):1396–1407, December 1987.

[107] M.-S. Chen, P. S. Yu, and K.-L. Wu. Scheduling and Processor Allocation for Parallel Execution of Multi- Join Queries. *Proceedings of the 8th International Conference on Data Engineering*, pages 58–67, February 1992.

[108] P.P. Chen. The Entity-Relationship Model-Toward a Unified View of Data. *ACM Transactions on Database Systems*, pages 9–36, 1976.

[109] Tsz S. Cheng, Shashi K. Gadia, and Sunil S. Nair. A seamless object oriented model for spatio-temporal databases. Technical Report TR-92-42, Iowa State University, 1992.

[110] M. Chock, A.F. Cardenas, and A. Klinger. Data Structure and Manipulation Capabilities of a Picture Database Management System (PICDMS). *IEEE Trans. on Pattern Analysis and Machine Intelligence*, PAMI-6(4):484–492, July 1984.

[111] S. Christodoulakis. Estimating Record Selectivities. *Information Systems*, 8(2):105–115, 1983.

[112] Alonzo Church. *The Calculi of Lambda-Conversion.* Princeton University Press, Princeton, 1941.

[113] David D. Clark, Scott Shenker, and Zhang. Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism. *Computer Communication Review (Proc. SIGCOMM '92)*, 22(4):14–26, October 1992.

[114] L. A. Clarke, J. C. Wileden, and A.L. Wolf. Object Management Support for Software Development Environments. In *Proceedings 1987 Appin Workshop on Persistent Object Stores*, pages 363–381, July 1987.

[115] Lori A. Clarke, Jack C. Wileden, and Alexander L. Wolf. GRAPHITE: A Meta-Tool For Ada Environment Development. COINS Technical Report 85–44, University of Massachusetts, Software Development Laboratory, Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts 01003, November 1985.

[116] J. Clifford. *A Logical Framework for the Temporal Semantics and Natural-Language Querying of Historical Databases.* PhD thesis, Dept. of Computer Science, SUNY at Stony Brook, December 1982.

[117] J. Clifford. A Model for Historical Databases. In *Proceedings of Workshop on Logical Bases for Data Bases*, Toulouse, France, December 1982.

[118] J. Clifford and A. Croker. On Consistent Extensions of the Relational Model of Data. Technical Report CRIS-#135, GBA-WP#86-100, Dept. of Information Systems, New York University, Leonard N. Stern School of Business, 1986.

[119] J. Clifford and A. Croker. The Historical Relational Data Model HRDM and Algebra Based on Lifespans. In *Proc. Third International Conference on Data Engineering*, pages 528–537, Los Angeles, February 1987. IEEE.

[120] J. Clifford, A. Croker, and A. Tuzhilin. On Completeness of Query Languages for Grouped and Ungrouped Historical Data Models. In A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass, editors, *Temporal Databases.* Press, 1993.

[121] J. Clifford and T. Isakowitz. On The Semantics of Transaction Time and Valid Time in BiTemporal Relations. Technical report, Center for Research on Information Systems, Stern School of Business, New York University, 1992.

[122] J. Clifford and A.U. Tansel. On an Algebra for Historical Relational Databases: Two Views. In S. Navathe, editor, *ACM SIGMOD*, pages 247–265, Austin, TX, May 1985. acm.

[123] J. Clifford and D. S. Warren. Formal Semantics for Time in Databases. *ACM Transactions on Database Systems*, 6(2):214–254, June 1983.

[124] P. Coad and E. Yourdon. Object-Oriented Analysis. *Prentice Hall, Englewood Cliffs, NJ*, 1991.

[125] E.F. Codd. Extending the Database Relational Model to Capture More Meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[126] E.F. Codd. A Relational Model of Data for Large Shared Data Banks", booktitle = " Readings in Database Systems. pages 5–15. Morgan Kaufman Publishers, Inc., 1990.

[127] E. Coffman, M. Garey, D. Johnson, and R. Tarjan. Performance Bounds for Level-Oriented Two-Dimensional Packing Algorithms. *SIAM Journal on Computing*, 9(4):808–826, November 1980.

[128] Douglas E. Comer and Rajendra S. Yavatkar. Flows: Performance Guarantees in Best Effort Delivery Systems. In *Proc. IEEE INFOCOM '89*, pages 100–109, April 1989.

[129] Intel Corporation. *IPSC/2 User's Guide.* Intel Corporation, March 1988.

[130] NCUBE Corporation. *NCUBE/ten: An Overview.* NCUBE Corporation, November 1985.

[131] J. K. Cringean, R. England, G. A. Manson, , and P. Willett. Processor and Link Assignment in Multicomputers Using Simulated Annealing. In *Proceedings of the 1990 ACM SIGIR*, pages 413–428, September 1990.

[132] Janey K. Cringean, Roger England, Gordon A. Manson, and Peter Willett. Network Design for the Implementation of Text Searching Using a Multicomputer. *Information Processing & Management*, 27(4):265–283, 1991.

[133] C. J. Crouch. An Approach to the Automatic Construction of Global Thesauri. *Information Processing & Management*, 26(5):629–640, 1990.

[134] Fish D. et al. Iris: An Object-Oriented Database Management System. *ACM Transactions on Office Information Systems*, 5(1):48–69, 1987.

[135] Maier D., Stein J., Otis A., and Purdy A. Development of an Object-Oriented DBMS. *ACM OOPSLA 1986 Conference*, 1986.

[136] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in Partitioned Networks. *ACM Computing Surveys*, 17(3):341–370, September 1985.

[137] M. Davis. *Computability and Unsolvability.* McGraw-Hill, 1958.

[138] U. Dayal, B. Blaustein, A. Buchmann, U. Chakravarthy, M. Hsu, R. Ladin, D. McCarthy, A. Rosenthal, and S. Sarin. The HiPAC Project: Combining Active Databases and Timing Constraints. *ACM-SIGMOD Record*, 17(1):51–70, March 1988.

[139] U. Dayal, M. Hsu, and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. In *Proceedings of the 1990 ACM SIGMOD Conference on Management of Data*, pages 204–214, Atlantic City, New Jersey, June 1990.

[140] U. Dayal, M. Hsu, and R. Ladin. A Transactional Model for Long-Running Activities. In *17th Int'l Conference on Very Large Data Bases*, pages 113–122, Barcelona, Spain, September 1991.

[141] D. De-Champeaux and W. Olthoff. Towards and Object-Oriented Analysis Technique. *Proceedings of the Pacific Northwest Quality Software Conference.*, September 1989.

[142] T. DeMarco. *Structured Analysis and System Specification.* Yourdon, 1978.

[143] Alan Demers, Srinivasan Keshav, and Scott Shenker. Analysis and Simulation of a Fair Queueing Algorithm. *Internetworking: Research and Experience*, 1:3–26, 1990.

[144] S. Demurjian and D. Hsiao. Towards a Better Understanding of Data Models Through the Multilingual Database System. *IEEE Transactions on Software Engineering*, 14(7), July 1988.

[145] Yi Deng and Shi-Kuo Chang. A Framework for the Modeling and Prototyping of Distributed Information Systems. *International Journal of Software Engineering and Knowledge Engineering*, 1(3):203–226, 1991.

[146] D. Denning. *Cryptography and Data Security.* Addison-Wesley, Reading, Mass., 1982.

[147] L. P. Deutsch. Levels of Reuse in the Smalltalk-80 Programming System. *Tutorial:Software Reusability, P. Freeman, Ed., IEEE Computer Society Press, Wash. D.C.*, 1987.

[148] L. P. Deutsch. Design Reuse and Frameworks in the Smalltalk-80 System. *Software Reusability, Vol 1, T. J. Biggerstaff and A. J. Perlis, Eds., ACM Press, pp-57-71*, 1989.

[149] Peter Deutsch. Resource Discovery in an Internet Environment–the Archie Approach. *Electronic Networking*, 2(1):45–51, Spring 1992.

[150] O. Deux et al. The Story of O_2. *IEEE Trans. on Knowledge and Data Engineering*, 2(1):91–108, 1990.

[151] O. Deux et al. The O_2 System. *Communicatons of the ACM*, 34(10), October 1991.

[152] P. Dewan. A Tour of the Suite User Interface Software. In *Proceedings of the 3rd ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 57–65, October 1990.

[153] P. Dewan, A. Vikram, and B. Bhargava. Engineering the Object- Relation Model in O- Raid. In *Proceedings of the International Conference on Foundations of Data Organization and Algorithms*, pages 389–403, June 1989.

[154] D. J. DeWitt and R. Gerber. Multiprocessor Hash-Based Join Algorithms. *Proceedings of the 11th International Conference on Very Large Data Bases*, pages 151–162, August 1985.

[155] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.I. Hsiao, and R. Rasmussen. The Gamma Database Machine Project. *IEEE Transactions on Knowledge and Data Engineering*, 2(1):44–62, March 1990.

[156] D. J. DeWitt, J. Naughton, and D. Schneider. Parallel External Sorting Using Probabilistic Splitting. *Proceedings of the 1st Conference on Parallel and Distributed Information Systems*, pages 280–291, December 1991.

[157] D. J. DeWitt, J. Naughton, D. Schneider, and S. Seshadri. Practical Skew Handling in Parallel Joins. *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 27–40, August 1992.

[158] D. M. Dias, B. R. Iyer, J. T. Robinson, and P. S. Yu. Integrated Concurrency-Coherency Controls for Data Sharing. *IEEE Transactions on Software Engineering*, SE-15(4):437–448, April 1989.

[159] T.E. Dielman. *Pooled Cross-Sectional and Time-Series Data Analysis.* Marcel Dekker, Inc., 1989.

[160] Pamela Drew, R. King, and Dennis Heimbigner. A Multi-Dimensional, Extensible Framework for the Integration of Next-Generation, Heterogeneous Database Management Systems. *VLDB Journal, Special Issue on Heterogeneous Databases*, 1992. To Appear 2Q.

[161] Pamela Drew, Roger King, and Jonathan Bein. A la Carte: An Extensible Framework for the Tailorable Construction of Heterogeneous Object Stores. In *Implementing Persistent Object Bases: Principles and Practice, The Fourth International Workshop on Persistent Object Systems.* Morgan Kaufmann Publishers, Inc., 1990.

[162] Pamela A. Drew. *A la Carte: An Implementation of a Toolkit for the Incremental Integration of Heterogeneous Database Management Systems.* PhD thesis, University of Colorado, Boulder, 1991.

[163] James R. Driscoll, David A. Rajala, William H. Shaffer, and Donald W. Thomas. The Operation and Performance of an Artificially Intelligent keywording System. *Information Processing and: Management*, 27(1):43–54, 1991.

[164] Batory D.S., Barnett J.R., Garza J.F., and Smith K.P. GENESIS: A Reconfigurable DBMS. Technical Report 86-07, Dept. of Computer Science, U.T. Austin, 1986.

[165] J. Du and L. Leung. Complexity of Scheduling Parallel Task Systems. *SIAM, Journal on Discrete Mathematics*, 2(4):473–487, November 1989.

[166] X. Du and F. J. Maryanski. Data Allocation in a Dynamically Reconfigurable Environment. In *Proceeding of the IEEE Fourth International Conference on Data Engineering*, pages 74–81, February 1988.

[167] Ken Dubose and Hyong S. Kim. An Effective Bit Rate/Table Lookup Based Admission Control Algorithm for the ATM B-ISDN. In *Proc. 17th Conf. on Local Computer Networks*, pages 20–29, Minneapolis, Minnesota, September 1992.

[168] Zbigniew Dziong, Jean Choquette, Ke-Qiang Liao, and Lorne G. Mason. Admission Control and Routing in ATM Networks. *Computer Networks and ISDN Systems*, 20(1–5):189–196, December 1990.

[169] M.T. Edmead. Interactive Applications Using CD-I. In *COMPCON Spring '87, 32nd IEEE Computer Society Intl. Conf.*, pages 410–412, San Francisco, CA, February 1987.

[170] F. Eliassen and R. Karlsen. Interoperability and Object Identity. *SIGMOD RECORD*, 20(4), December 1991.

[171] C. Elkan. A Decision Procedure for Conjunctive Query Disjointness. *8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 1989.

[172] A. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan-Kaufmann, February 1992.

[173] R. Elmasri, I. ElAssal, and V. Kouramajian. Semantics of Temporal Data in an Extended ER Model. In *9th Entity-Relationship Conference*, October 1990.

[174] R. Elmasri and V. Kouramajian. A Temporal Query Language Based on Conceptual Entities and Roles. In *11th Entity-Relationship Conference*, October 1992.

[175] R. Elmasri and Shamkant Navathe. *Fundamentals of Database Systems.* Benjamin Cummings, 1989.

[176] R. Elmasri, J. Weeldreyer, and A Hevner. The Category Concept: An Extension to the ER Model. *Data and Knowledge Engineering*, pages 75–116, June 1985.

[177] R. Elmasri and J. Wiederhold. GORDAS: A Formal High Level Query Language for the ER Model. *2nd Entity-Relationship Conference*, October 1981.

[178] R. Elmasri and G. Wuu. Index by Time Interval: An Efficient Access Structure for Temporal Queries. *Submitted for publication.*, 1990.

[179] R. Elmasri, G. Wuu, and V. Kouramajian. *A Temporal Model and Query Language for EER Databases.* Benjamin Cummings, 1993.

[180] E.A. Emerson. Temporal and Modal Logic. In Jan van Leeuwen., editor, *Handbook of Theoretical Computer Science*, page 995. Elsevier, New York, 1990.

[181] Julio Escobar, Debra Deutsch, and Craig Partridge. Flow Synchronization Protocol. BBN Systems and Technologies Division, Cambridge, MA, March 1992.

[182] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger. The Notions of Consistency and Predicate Locks in Database Systems. *Communications of ACM*, pages 624–633, november 1976.

[183] O. Etzion, A. Segev, and G. Avigdor. A Temporal Active Database Model. Technical Report LBL-32587, Lawrence Berkeley Lab, 1992.

[184] J.J. Ewing. An Object-Oriented Operating System Interface. *OOPSLA'86 Proceedings, SIGPLAN Not. (ACM) 21, 11 (Portland, Oregon*, November 1986.

[185] A. Farrag and M. Ozsu. Using Semantic Knowledge of Transactions to Increase Concurrency. *ACM Transactions on Database Systems*, 14(4):503–525, December 1989.

[186] Domenico Ferrari. Client Requirements for Real-Time Communication Services. *IEEE Communications Magazine*, 28(11):65–72, November 1990.

[187] R. G. Fichman and Chris F. Kemerer. Object-Oriented and Conventional Analysis and Design Methodologies. *IEEE Computer*, October 1992.

[188] D. Fishman, D Beech, H.P. Cate, and E.C. Chow. Iris: An Object-Oriented Database System. *ACM Transactions On Office Information Systems*, 5(1):48–69, January 1987.

[189] D. Fishman et al. Overview of the Iris DBMS. *Object-Oriented Concepts, Databases, and Applications, W. Kim, and F. Lochovsky, eds., Addison-Wesley*, pages 219–250, 1989.

[190] David W. Flater and Yelena Yesha. An Efficient Management of Read-Only Data in a Distributed Information System. Technical Report CS-92-04, University of Maryland Baltimore County, Baltimore, MD 21228, March 1992.

[191] David W. Flater and Yelena Yesha. Query Routing and Object Caching in a Large Distributed Information System. In *Proceedings of the ISMM First International Conference on Information and Knowledge Management*, pages 525–534, Baltimore, MD, U.S.A., November 1992. The International Society for Mini and Microcomputers.

[192] David W. Flater and Yelena Yesha. A Robust and Efficient Strategy for the Distributed Caching of Read-Only Data in a Large Networked Information System. *International Journal of Intelligent and Cooperative Information Systems*, 1993. Submitted.

[193] David W. Flater and Yelena Yesha. An Efficient Management of Read-Only Data in a Distributed Information System. *International Journal of Intelligent and Cooperative Information Systems, Special Issue on Information and Knowledge Management*, 1993. To appear.

[194] David W. Flater and Yelena Yesha. An Information Retrieval System for Network Resources. In *Proceedings of the International Workshop on Next Generation Information Technologies and Systems*, 1993. To appear.

[195] David W. Flater and Yelena Yesha. Properties of Networked Information Retrieval with Cooperative Caching. Technical Report CS-93-08, University of Maryland Baltimore County, Baltimore, MD 21228, April 1993.

[196] Committee for Advanced DBMS Function. Third-Generation Database System Manifesto. *SIGMOD RECORD*, 19(3):31–44, September 1990.

[197] O. Frieder. Multiprocessor Algorithms for Relational-Database Operations on Hypercube Systems. *IEEE Computer*, pages 13–28, November 1990.

[198] O. Frieder and C. K. Baru. Site Selection and Query Scheduling Policies for Multicomputer Database Systems. In *Proceeding of IEEE Transactions on Knowledge and Data Engineering*, pages 74–81, February 1994. To appear in 1994.

[199] O. Frieder, V. A. Topkar, R. K. Karne, and A. K. Sood. Experimentation with Hypercube Database Engines. *IEEE Micro*, 12(1):42–56, February 1992.

[200] R. Fujimoto. Parallel Discrete Event Simulation. *Communications of the ACM*, 33(10):31–53, 1990.

[201] S. Gadia and C. Yeung. A Generalized Model for a Temporal Relational Database. *ACM SIGMOD-88 Conference Proceedings.*, June 1988.

[202] S. K. Gadia and J. Vaishnav. A Query Language for a Homogeneous Temporal Database. In *Proc. of The Fourth Annual ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 51–56, 1985.

[203] Shashi K. Gadia. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, 13:418–448, 1988.

[204] Shashi K. Gadia and Sunil S. Nair. Temporal Databases: A Prelude to Parametric Data. In A. Tansel et al., editors, *Temporal Databases: Theory, Design and Implementation*. Benjamin Cummings Publishing Company, 1993.

[205] Shashi K. Gadia and Chuen-Sing Yeung. Inadequacy of Interval Timestamps in Temporal Databases. *Information Sciences*, 54:1–22, 1991.

[206] Z. Galil and N. Megiddo. A Fast Selection Algorithm and the Problem of Optimal Distribution of Effort. *Journal of the ACM*, pages 58–64, 1979.

[207] S. Gantimahapatruni and G. Karabatis. Enforcing Data Dependencies in Cooperative Information Systems. In *Proceedings of International Conference on Intelligent and Cooperative Information Systems*, Rotterdam, The Netherlands, May 1993.

[208] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[209] H. Garcia-Molina. Using Semantic Knowledge for Transaction Processing in a Distributed Database. *ACM Transactions on Database Systems*, 8(2):186–213, June 1983.

[210] H. Garcia-Molina, D. Gawlick, J. Klein, K. Kleissner, and K. Salem. Coordinating Multi-Transaction Activities. Technical Report CS-TR-247-90, Princeton University, February 1990.

[211] M. Garey and R. Graham. Bounds for Multiprocessor Scheduling with Resource Constraints. *SIAM Journal on Computing*, 4(2):187–200, June 1975.

[212] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[213] N. Gehani and W.D. Roome. *The Concurrent C Programming Language.* Silicon Press, 1989.

[214] N. H. Gehani. Capsules: a Shared Memory Access Mechanism for Concurrent C/C++. *To Be Published in IEEE Transactions on Parallel and Distributed Systems*, 1990.

[215] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Composite Event Specification in Active Databases: Model and Implementation. *Proc. of the 18th Int'l Conf. on Very Large Databases*, August 1992.

[216] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event Specification in an Active Object-Oriented Database. *Proc. ACM-SIGMOD 1992 Int'l Conf. on Management of Data*, 1992.

[217] N.H. Gehani and H. V. Jagadish. Ode as an Active Database: Constraints and Triggers. *Proc. 17th Int'l Conf. Very Large Data Bases*, pages 327–336, 1991.

[218] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On Serializability of Multidatabase Transactions through Forced Local Conflicts. In *Proceedings of the* 7^{th} *IEEE International Conference of Data E ngineering*, April 1991. Kobe, Japan.

[219] R. Gerber. Dataflow Query Processing Using Multiprocessor Hash-Partitioned Algorithms. Technical Report Tech. Rep. 672, Computer Science Department, University of Wisconsin-Madison, October 1986.

[220] S. Ghandeharizadeh, R. Hull, and Dean Jacobs. Implementation of Delayed Updates in Heraclitus. In *Proc. of Intl. Conf. on Extending Data Base Technology*, 1992. to appear.

[221] P.B. Gibbons. A Stub Generator for Multilanguage RPC in Heterogeneous Environments. *IEEE Transactions of Software Engineering*, 13(1):77–87, January 1987.

[222] Henry M. Gladney. A Model for Distributed Information NeTworks. Technical Report RJ5220, IBM Almaden Res. Lab, 650 Harry Road, San Jose, California 95120-6099, July 1986.

[223] A. Go et al. An Approach to implementing a Geo-Data System. *TR ERL-M529, University of California, Berkeley*, June 1975.

[224] D. E. Goldberg. *Genetic Algorithms in Search Optimization and Machine Learning.* Addison Wesley, 1989.

[225] Jonny Goldman. In response to questions sent Through e-mail.

[226] S. J. Golestani. Congestion-Free Transmission of Real-Time Traffic in Packet Networks. In *Proc. IEEE INFOCOM '90*, pages 527–536, San Francisco, CA, 1990.

[227] Goodchild. *Accuracy of Spatial Databases.* Taylor and Francis, 1989.

[228] R. Graham. Bounds on Multiprocessor Timing Anomolies. *SIAM Journal of Computing*, pages 416–429, 1969.

[229] O. Guenther and A. Buchmann. Research Issues in Spatial Databases. *SIGMOD RECORD*, December 1990.

[230] Oliver Gunther. Efficient Structures for Geometric Data Management. *Lecture notes in Computer science*, 1988. Vol. 337.

[231] Oliver Gunther. The Design of the Cell Tree: An Object Oriented Index Structure for Geometric Databases. In *Proc. of the Fifth IEEE International Conference on Data Engineering*, pages 598–605, 1989.

[232] Oliver Gunther and Eugene Wong. A Dual Space Representation for Geometric Data. In *Proceedings of the 13'th International Conference on Very Large Data Bases*, pages 501–506, 1987.

[233] M.S. Guo, S.Y.W. Su, and H. Lam. An Association Algebra for Processing Object-Oriented Databases. In *Proc. 7th IEEE Int'l Conf. on Data Engineering*, 1991.

[234] A. Gupta, C.L. Rorgy, A. Newell, and R. Wedig. Parallel Algorithms and Architectures for Rule-Based Systems. In *International Symposium on Computer Architecture*, 1986.

[235] Anoop Gupta, Charles Forgy, Allen Newell, and Robert Wedig. Parallel Algorithms and Architectures for Rule-Based System. In *ICPP*, 1986.

[236] S. Guptill and L. Starr. Making Maps with Computers. *American Scientist*, 76:136–142.

[237] R.H. Guting. Geo-Relational Algebra: A Model and Query Language for Geometric Database Systems. In *Proc. of the Conference on Extending Database Technology(EDBT '88)*, pages 506–527, 1988.

[238] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings of the ACM SIGMOD International Conference on Management of Data.*, 1984.

[239] Bernard J. Haan, Paul Kahn, Victor A. Riley, James H. Coombs, and Norman K. Meyrowitz. IRIS Hypermedia Services. *Communications of ACM*, 35(1):36–51, January 1992.

[240] T. Haerder and A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, 15(4), December 1983.

[241] Donna Harman, Wayne McCoy, Robert Toense, and Gerald Candela. Prototyping a Distributed Information Retrieval System That Uses Statistical Ranking. *Information Processing and Management*, 27(5):449–460, 1991.

[242] William Harrison. RPDE: A Framework for Integrating Tool Fragments. *IEEE Software*, November 1987.

[243] R. L Haskin. Special Purpose Processors for Text Retrieval. *Database Engineering*, 4:16–29, September 1981.

[244] R. Hayes, S.W. Manweiler, and R. D. Schlichting. A Simple System for Constructing Distributed, Mixed-Language Programs. *Software - Practice and Experience*, 18(7):641–660, July 1988.

[245] D. Heimbigner and D. McLeod. A Federated Architecture for Information Management. *ACM Transactions on Office Information Systems*, 3(3), 1985.

[246] Dennis Heimbigner. A Tower of Babel: Providing Flexible and Extensible Environment Services. Draft Technical Report CU–CS–580–92, University of Colorado, Boulder, University of Colorado at Boulder, Department of Computer Science, Campus Box 430, Boulder, CO 80309, 1992.

[247] Dennis Heimbigner. Triton Reference Manual. Technical Note CU–CS–483–92, University of Colorado, Computer Science Department, University of Colorado, Boulder, CO 80309–0430, May 5th 1992. Version 0.8.

[248] B. Henderson-Sellars and J.M. Edwards. The Object Oriented Systems Life Cycle. *Communications of the ACM*, September 1990.

[249] J. Herring et al. Extensions to the SQL Language to Support Spatial Analysis in a Topological Data Base. *GIS/LIS'88, San Antonio, Texas*, November 1988.

[250] R.G. Herrtwich. Time Capsules: An Abstraction for Access to Continuous-Media Data. *Proc. 11th Real-Time Systems Symp.*, pages 11–20, December 1990.

[251] Jerry Hobbs. Presentation on the use of a finite State Machine (FASTUS) to perform Information extraction from natural Language texts given in a panel session at the First International Conference on Information and Knowledge Management.

[252] M.E. Hodges, R.M. Sasnett, and M.S. Ackerman. A Construction Set for Multimedia Applications. *IEEE Software*, pages 37–43, January 1989.

[253] Petra Hoepner. Synchronizing the Presentation of Multimedia Objects – ODA Extensions. *ACM SIGOIS Bulletin*, 12(1):19–32, July 1991.

[254] W. Hong. Exploiting Inter-Operator Parallelism in XPRS. *Proceedings of ACM SIGMOD*, pages 19–28, June 1992.

[255] W. Hong and M. Stonebraker. Optimization of Parallel Query Execution Plans in XPRS. *Proceedings of the 1st Conference on Parallel and Distributed Information Systems*, pages 218–225, December 1991.

[256] W.C. Hou, G. Ozsoyoglu, and B.K. Taneja. Processing Aggregate Relational Queries with Hard Time Constraints. *Proc. ACM SIGMOD Intl. Conf. on Management of Data*, June 1989.

[257] M. Hsu, R. Ladin, and D. McCarthy. An Execution Model for Active Data Base Management Systems. In *Proceedings of the* 3^{rd} *International Conference on Data and Knowledge Bases*, June 1988.

[258] M. Hsu and A. Silberschatz. Persistent Transmission and Unilateral Commit- A Position Paper. In *Workshop on Multidatabases and Semantic Interoperability*, Tulsa, OK, October 1990.

[259] K. A. Hua and C. Lee. Handling Data Skew in Multiprocessor Database Computers Using Partition Tuning. *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 525–535, September 1991.

[260] K. A. Hua, Y.-L. Lo, and H. C. Young. Including the Load Balancing Issue in the Optimization of Multi-Way Join Queries for Shared-Nothing Database Computers. *Proceedings of the 2nd Conference on Parallel and Distributed Information Systems*, pages 74–83, January 1993.

[261] G. David Huffman, Dennis A. Vital, and Royal G. Bivins, Jr. Generating Indices with Lexical Association Methods: Term Uniqueness. *Information Processing and Management*, 26(4):549–558, 1990.

[262] Joseph Y. Hui. Resource Allocation for Broadband Networks. *IEEE Journal on Selected Areas in Communications*, 6(9):1598–1608, December 1988.

[263] J. Hull. *Options, Futures and Other Securities.* Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[264] R. Hull. *A Survey of Theoretical Research on Typed Complex Database Objects, J. Paredaens Editor.* Databases. Academic Press, 1987.

[265] R. Hull and D. Jacobs. On the Semantics of rules in Database Programming Languages, In Next Generation Information System Technology: volume 504. In *Proc. of the First International East/West Workshop, Kiev, USSR*, pages 59–85. Springer-Verlag LNCS, October 1990. ed. by J. Schmidt and A. Stogny.

[266] R. Hull and R. King. Semantic Database Modeling: Survey, Application, and Research Issues. *ACM Computing Surveys*, 19(3):201–258, September 1987.

[267] Richard Hull and Dean Jacobs. Language Constructs for Programming Active Databases. In *Proceedings of the 17th International Conference on Very Large Data Bases, Barcelona*, pages 455–468, September 1991.

[268] S. Hummel, E. Schonberg, and L. Flynn. Factoring: A Method for Scheduling Parallel Loops. *Communications of the ACM*, pages 90–101, August 1992.

[269] Steve Huseth, Jonathan Krueger, Aaron Larson, and J. M. Purtilo. The Common Prototyping Language, A Module Interconnection Approach. Technical Report UMIACS–TR–90–50, S–TR–2449, University of Maryland, Department of Computer Science, University of Maryland, College Park, MD 20742, May 1990.

[270] T. Ibaraki and N. Katoh. *Resource Allocation Problems: Algorithmic Approaches.* The MIT Press, Cambridge, Massachusetts, 1988.

[271] T. Imielinski. Abstraction in Query Processing. *JACM*, 38(3), 1991.

[272] T. Imielinski. Incomplete Deductive Databases. *Annals of Mathematics and Artificial Intelligence, was Available as an Unpublished Manuscript in 1988 Spring.*, 3:259–294, 1991.

[273] T. Imielinski, R. van der Meyden, and K. Vadaparty. Complexity Tailored Design – A New Database Design Methodology. Technical Report LCSR 171, Rutgers University, Department Of Computer Science, New Brunswick, NJ08903, December 1991.

[274] T. Imielinski, S. Naqvi, and K. Vadaparty. Incomplete Objects – A Data Model for Design and Planning Applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Denver, Colorado, May 1991.

[275] T. Imielinski, S. Naqvi, and K. Vadaparty. Querying Design and Planning Databases. Submitted to Second International Conference on Deductive and Object Oriented Databases (DOOD), March 1991.

[276] T. Imielinski and K. Vadaparty. Complexity of Querying Databases with OR-Objects. In *Proceedings of the Eighth ACM Symposium on Principles of Databases Systems (PODS)*, 1989.

[277] T. Imielinski, K. Vadaparty, and R. van der Meyden. Complexity Tailored Design – A New Database Design Methodology. Technical Report LCSR 171, Rutgers University, Department Of Computer Science, New Brunswick, NJ08903, December 1991.

[278] Ontologic Inc. *ONTOS 2.1 Product Description.* MA., 1991.

[279] Interactive Software Engineering, 270 Storke Road, Suite 7, Goleta CA 93117, U.S.A. *The Environment, The Language, The Libraries.* Version 2.3.

[280] Y. E. Ioannidis and Y. C. Kang. Left-Deep vs. Bushy Trees: An Analysis of Strategy Spaces and its Implication for Query Optimization. *Proceedings of ACM SIGMOD*, pages 168–177, May 1991.

[281] Judith H. Irven, Margaret E. Nilson, Thomas H. Judd, John F. Patterson, and Yoshitaka Shibata. Multi-Media Information Services: A Laboratory Study. *IEEE Communications Magazine*, 26(6):27–44, June 1988.

[282] Fan I.S. and Sackett P.J. A PROLOG Simulator for Interactive Flexible Manufacturing System Control. *Simulation*, 50(6):239–247, 1988.

[283] T. Ishida. Parallel Rule Firing in Production Systems. *IEEE Transactions on Knowledge and Data Engineering*, pages 11–17, 1991.

[284] B. Iyer, G. Ricard, and P. Varman. Percentile Finding Algorithm for Multiple Sorted Runs. *Proceedings of the 15th International Conference on Very Large Data Bases*, pages 135–144, August 1989.

[285] Widom J., Cochrane R., and B.G. Lindsay. Implementing Set-Oriented Production Rules as an Extension to Starburst. In *17th International Conference on Very Large Databases*, 1991.

[286] Dean Jacobs and Richard Hull. Database Programming with Delayed Updates. In *Intl. Workshop on Database Programming Languages*, San Mateo, California, 1991. Morgan-Kaufmann, Inc. to appear.

[287] H. V. Jagadish and O. Shmueli. Synchronizing Trigger Events in a Distributed Object-Oriented Database. *Proc. Int'l Workshop on Distributed Object Management*, August 1992.

[288] S. Jajodia and B. Kogan. Integrating an Object-Oriented Data Model with MultiLevel Security. *Proc. IEEE Symposium on Security and Privacy, Oakland, California*, pages 76–85, May 1990.

[289] S. Jajodia, B. Kogan, and R. Sandhu. A MultiLevel-Secure Object-Oriented Data Model. *Technical Report, George Mason University*, 1992.

[290] S. Jajodia and D. Mutchler. Dynamic Voting. In *ACM SIGMOD Conference*, pages 227–238, 1987.

[291] S. Jajodia and D. Mutchler. A Pessimistic Consistency Control Algorithm for Replicated Files which Achieves High Availability. *IEEE Transactions on Software Engineering*, 15(1):39–46, Jan 1989.

[292] S. Jajodia and R. Sandhu. Toward a MultiLevel Secure Relational Data Model. In *Proc. ACM SIGMOD Int'l. Conf. on Management of Data*, pages 50–59, Denver, Colorado, May 1991.

[293] C. Jensen. *Towards the Realization of Transaction Time Database Systems.* PhD thesis, University of Maryland, December 1990.

[294] C.S. Jenson, J. Clifford, S.K. Gadia, A. Segev, and R.T. Snodgrass. A Glossary of Temporal Database Concepts. *ACM SIGMOD Record*, 21(3):35–43, 1992.

[295] Y. Jiang. Indexing Support for Distributed Object-Oriented Database Systems. Preliminary Ph.D. exam, Purdue University, 1992.

[296] R.E. Johnson and B. Foote. Designing Reusable Classes. *J. of Object-Oriented Programming*, June/July 1988.

[297] L.L. Jones and S.G. Smith. Can Multimedia Instruction Meet Our Expectations. *Educom Review*, 27(1):39–43, January/February 1992.

[298] Rick Jones. Moving up to Eiffel 3. *JOOP*, September 1992.

[299] T. Joseph and A. F. Cardenas. PICQUERY: A High Level Language for Pictorial Database Management. *IEEE Transactions on Software Engineering*, 14(5):630–638, May 1988.

[300] Marsha Jovanovic. Bringing It Together for Global Change Research. *San Diego Supercomputer Center Gather/Scatter*, 8(9-10), Sept. - Oct. 1992.

[301] Stanley M. Sutton Jr., Dennis Heimbigner, and Leon J. Osterweil. Language Constructs for Managing Change in Process-Centered Environments. In *The Fourth Symposium on Practical Software Development Environments*, Irvine, CA., 1990.

[302] Stanley M. Sutton Jr., Hadar Ziv, Dennis Heimbigner, Harry E. Yessayan, Mark Maybee, Leon J. Osterweil, and Xiping Song. Programming a Software Requirements-Specification Process. *IEEE Software*, 1991.

[303] A. Juels. Personal Communication.

[304] E. Jungert et al. Vega – A Geographical Information System. *Proc. Scandinavian Research Conf. on Geographical Information Systems*, June 1985.

[305] Brewster Kahle. think.com:/public/wais/README, September 1991.

[306] H. Kang and S.H. Son. A Hierarchical Export/Import Scheme for Data Sharing in a Federated Distributed Database System. *International Symposium on Database Systems for Advanced Applications*, 1991.

[307] G. Kappel and M. Schrefl. Object/Behavior Diagrams. In *Proc. 7th IEEE Int'l Conf. on Data Engineering*, pages 530–539, 1991.

[308] P. Karger. Limiting the Damage Potential of Discretionary Trojan Horses. In *Proc. of the 1987 IEEE Symposium on Research in Security and Privacy*, Oakland (Calif.), May 1987.

[309] A. Karmouch, L. Orozco-Barbosa, N.D. Georganas, and M. Goldberg. A Multimedia Medical Communications System. *IEEE Journal on Selected Areas in Communications*, 8(3):325–339, April 1990.

[310] R. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4), December 1990.

[311] R. H. Katz. *Information Management for Engineering Design*. Springer-Verlag, 1985.

[312] T. F. Keefe and W. T. Tsai. Prototyping the SODA Security Model. *Database Security, III: Status and Prospects, David L. Spooner and Carl Landwehr, eds., North-Holland, Amsterdam*, pages 211–235, 1990.

[313] T. F. Keefe, W. T. Tsai, and M. B. Thuraisingham. A MultiLevel Security Model for Object-Oriented System. In *Proc. 11th National Computer Security Conference*, pages 1–9, October 1988.

[314] R. K. Keller, M. Cameron, R. N. Taylor, and D. B. Troup. User Interface Development and Software Environments: The Chiron-1 System. In *Proceedings of the 13th International Conference on Software Engineering, Austin, Texas*, pages 208–218, May 13–16 1991.

[315] W. Kent. The Breakdown of the Information Model in Multi-Database Systems. *SIGMOD RECORD*, 20(4), December 1991.

[316] B. W. Kernighan and D. M. Ritchie. *The C Programming Language (2nd Ed.).* Prentice-Hall, 1988.

[317] M.A. Ketabchi and V. Berzins. Mathematical Model of Composite Objects and Its Application for Organizing Engineering Databases. *IEEE Trans. on Software Engineering*, 14(1):71–84, January 1988.

[318] W. Kim. Research directions for integrating heterogeneous databases. *SIGMOD RECORD*, December 1989.

[319] W. Kim, N. Ballou, H. T. Chou, J. Garza, and D. Woelk. Features of the ORION Object-Oriented Database System. *Object-Oriented Concepts, Databases, and Applications, W. Kim, and F. Lochovsky, eds., Addison-Wesley*, pages 251–282, 1989.

[320] W. Kim, E. Bertino, and J. Garza. Composite Objects Revisited. In *Proc. of ACM-SIGMOD Conference on Management of Data*, Portland (Oregon), May 29-June 3 1989.

[321] Won Kim. Object-Oriented Databases: Definition and Research Directions. *IEEE Transactions on Knowledge and Data Engineering*, 2(3), September 1990.

[322] John Kimball, Tim King, Aaron Larson, and Jon Ward. ProtoTech Phase I, Final Technical Report. Draft Technical Report CS–C92–002, Honeywell/University of Maryland, 1989. December 8th 1989 – April 24th 1992.

[323] M. Kitsuregawa, M. Nakano, L. Hirada, and M. Takagi. Performance Evaluation of Functional Disk System with Nonuniform Data Distribution. *Proceedings of the 2nd International Symposium on Databases in Parallel and Distributed Systems*, pages 80–89, July 1990.

[324] M. Kitsuregawa, M. Nakayama, and M. Takagi. The Effect of Bucket Size Tuning in Dynamic Hybrid GRACE Hash Join Method. *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 257–266, August 1988.

[325] M. Kitsuregawa and Y. Ogawa. Bucket Spreading Parallel Hash: A New, Robust Parallel Hash Join Method for Data Skew in the Super Database Computer (SDC). *Proceedings of the 16th VLDB Conference*, pages 23–34, August 1990.

[326] Morse K.L. Parallel Distributed Simulation in ModSim. In *International Conference on Parallel Processing*, 1990.

[327] J. Klein and A. Reuter. Migrating Transactions. In *Future Trends in Distributed Computing Systems in the 90's*, Hong Kong, 1988.

[328] L.R. Klien. *A Textbook of Economics.* Evanston: Row Peterson and Co., 1953.

[329] D. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley, MA, 1973.

[330] H. Korth and A. Silberschatz. *Database System Concepts.* McGraw-Hill Book Company, 1991.

[331] H.F. Korth, E. Levy, and A. Silberschatz. A Formal Approach to Recovery by Compensating Transactions. In *Proceedings of the 16th International Conference on VLDB*, pages 95–106, 1990.

[332] R. Kowalski. Algorithm = Logic + Control. *Communications of ACM*, pages 424–475, July 1979.

[333] N. Krishnakumar and A. Bernstein. Bounded Ignorance in Replicated Systems. *Proc. of the Symposium on Principles of Database Systems*, May 1991.

[334] R. Krishnamurthy, H. Boral, and C. Zaniolo. Optimization of Non-Recursive Queries. *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 128–137, August 1986.

[335] R. Krishnamurthy, W. Litwin, and W. Kent. Language Features for Interoperability of Databases with Schematic Discrepancies. *ACM SIGMOD International Conference on Management of Data*, 1991.

[336] R. Krishnamurti and E. Ma. An Approximation Algorithm for Scheduling Tasks on Varying Partition Sizes in Partitionable Multiprocessor Systems. *IEEE Transactions on Computers*, 41(12):1572–1579, December 1992.

[337] A. Kumar and M. Stonebraker. Semantics Based Transaction Management Techniques for Replicated Data. In *Proceedings of ACM SIGMOD*, pages 117–125, 1988.

[338] C.H. Kung. Conceptual Modeling in the Context of Software Development. *IEEE Transactions on Software Engineering*, 15(10):1175–1187, October 1989.

[339] G. Kuper. *The Logical Data Model.* PhD thesis, Stanford University, Dept.of Computer Science, 1985.

[340] Forgy Charles L. RETE : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*, 19, 1982.

[341] Raschid L., T. Sellis, and C.-C. Lin. Exploiting Concurrency in a DBMS Implementation for Production Systems. In *The First Symposium on Databases in Parallel and Distributed Systems*, 1988.

[342] Rowe L. and Stonebraker M. The POSTGRES Data Model. In *International Conference on VLDB*, 1987.

[343] Bellcore Information Networking Research Laboratory. The Touring Machine System. *Communications of ACM*, 36(1):68–77, January 1993.

[344] M. S. Lakshmi and P. S. Yu. Effectiveness of Parallel Joins. *IEEE Transactions on Knowledge and Data Engineering*, 2(4):410–424, December 1990.

[345] H. Lam, S. Su, and A. Alashqur. Integrating the Concepts and Techniques of Semantic Modeling and the Object-Oriented Paradigm. In *Proc. 13th Int'l Computer Software and Applications Conference (COMPSAC)*, pages 209–217, 1989.

[346] H. Lam, S. Su, et al. GTOOLS: An Active Graphical User Interface Toolset for an Object-Oriented KBMS. In *International Journal of Computer Science and Engineering*, volume 7, pages 69–85, 1992.

[347] Charles Lamb, Gordon Landis, Jack Orenstein, and Dan Weinreb. The ObjectStore Database System. *Communicatons of the ACM*, 34(10), October 1991.

[348] L. Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of ACM*, 21(7), July 1978.

[349] J. Lassila. Preliminary Data in Econometric Databases. In G. Goos and J. Hartmanis, editors, *Lecture Notes in Computer Science 420.* Springer-Verlag, 1990.

[350] B. Leban, D. McDonald, and D. Forster. A Representation for Collections of Temporal Intervals. In *Proceedings of the AAAI-1986 5th Int. Conf. on Artificial Intelligence*, pages 367–371, 1986.

[351] C. Lecluse, P. Richard, and F. Velez. O_2, an Object-Oriented Data Model. In *Proc. ACM-SIGMOD 1988 International Conference on Management of Data*, pages 424–433, June 1988.

[352] S.-Y. Lee and J. K. Aggarwal. A Mapping Strategy for Paralell Processing. *IEEE Transactions on Computers*, 36(4):433–442, April 1987.

[353] C. E. Leiserson et al. The Network Architecture of the Connection Machine CM-5. *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 272–285, June 1992.

[354] D. Lewis. General Semantics. In Barbara H. Partee, editor, *Montague Grammar*, pages 1–50. Academic Press, Inc., New York, 1976.

[355] K. J. Lieberherr et al. Object-Oriented Programming: An Objective Sense of Style. *OOPSLA'88 Proceedings, SIGPLAN Not. (ACM) 23, 11, (San Diego, CA*, September 1988.

[356] K. J. Lieberherr et al. From Objects to Classes: Algorithms for Object-Oriented Design. *Tech. Report Demeter-3, Northeastern University*, January 1990.

[357] K. J. Lieberherr and I. Holland. Assuring Good Style for Object-Oriented Programs. *IEEE Software*, (September) 1989.

[358] K. J. Lieberherr and I. Holland. Tools for Preventive Software Maintenance. *Proceedings of the Conference on Software Maintenance, IEEE Press, Miami Beach, FL.*, October 1989.

[359] K. J. Lieberherr and A. J. Riel. Demeter: A CASE Study of Software Growth through Parameterized Classes. *Jrnl. of Object-Oriented Programming. 1,3*, Aug/Sept 1988.

[360] B. Linder. Informix Parallel Data Query. *Proceedings of the 2nd Intl. Conf. on Parallel and Distributed Information Systems*, pages 71–72, January 1993.

[361] B. Linder. Oracle Parallel RDBMS on Massively Parallel Systems. *Proceedings of the 2nd Intl. Conf. on Parallel and Distributed Information Systems*, pages 67–68, January 1993.

[362] B. Lindsay, L. Haas, C. Mohan, H. Pirahesh, and P. Wilms. A Snapshot Differential Update Algorithm. *ACM SIGMOD International Conference on Management of Data*, 1986.

[363] A. Lippman and W. Bender. News and Movies in the 50 Megabit Living Room. *Globecom '87*, pages 1976–1981, November 9187.

[364] Andrew Lippman and William Butera. Coding Image Sequences for Interactive Retrieval. *Communications of ACM*, 32(7):852–860, July 1989.

[365] Jr. Lipski. On Semantic Issues Connected with Incomplete Information Databases. *ACM Trans. on Database Systems*, 4(3):262–296, 1979.

[366] B. Liskov, T. Bloom, D. Gifford, R. Scheifler, and W. Weihl. Communication in the Mercury System. Memo 59–1, Programming Methodology Group, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1988.

[367] T.D.C. Little and A. Ghafoor. Spatio-Temporal Composition of Distributed Multimedia Objects for Value-Added Networks. *Computer*, 24(10):42–50, October 1991.

[368] Thomas D. C. Little and Arif Ghafoor. Network Considerations for Distributed Multimedia Object Composition and Communication. *IEEE Network Magazine*, pages 32–49, November 1990.

[369] Thomas D. C. Little and Arif Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, April 1990.

[370] Thomas D. C. Little and Arif Ghafoor. Multimedia Synchronization Protocols for Broadband Integrated Services. *IEEE Journal on Selected Areas in Communication*, 9(9):1368–1381, December 1991.

[371] Thomas D. C. Little, Arif Ghafoor, and C. Y. Roger Chen. Conceptual Data Models for Time-Dependent Multimedia Data. In *Proc. International Workshop on Multimedia Information Systems (MMIS '92)*, pages 86–110, February 1992.

[372] W. Litwin. MALPHA: A Relational Multidatabase Manipulation Language. *1st IEEE Conference on Data Engineering*, 1984.

[373] W. Litwin et al. SIRIUS Systems for Distributed Data Management. pages 311–366. North-Holland Publishing, 1982.

[374] W. Litwin and H. Tirri. Flexible Concurrency Control Using Value Dates. Technical Report 845, INRIA, May 1988.

[375] Delcambre L.M.L and J.N. Etheredge. The Relational Production Language: A Production Language for Relational Databases. In *Second International Conference of Expert Database Systems*, 1988.

[376] M.-L. Lo, M.-S. Chen, C. V. Ravishankar, and P. S. Yu. On Optimal Processor Allocation to Support Pipelined Hash Joins. *Proceedings of ACM SIGMOD*, May 1993.

[377] G. M. Lohman et al. The POSTGRES Next Generation Database Management System. *Communications of ACM*, 34(10):78–92, October 1991.

[378] G. M. Lohman, B. Lindsay, H. Pirahesh, and K. B. Schiefer. Extensions to Starburst: Objects, Types, Functions, and Rules. *Communications of ACM*, 34(10):94–109, October 1991.

[379] Guy M. Lohman, Bruce Lindsay, Hamid Pirahesh, and K. Bernhard Schiefer. Extensions to Starburst: Objects, Types, Functions, and Rules. *Communicatons of the ACM*, 34(10), October 1991.

[380] T. Lovett and S.S. Thakkar. The Symmetry Multiprocessor System. *Proc. of International Conference on Parallel Processing*, August 1988.

[381] H. Lu, M.-C. Shan, and K.-L. Tan. Optimization of Multi-Way Join Queries for Parallel Execution. *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 549–560, September 1991.

[382] David C. Luckham and Friedrich W. von Henke. An Overview of Anna, a Specification Language for Ada. *IEEE Software*, March 1985.

[383] L.F. Ludwig. Integration of CAD/CAE with Multimedia Teleconferencing and Messaging via Broadband Networks and Shared Resource Servers. *Proc. 1st Intl. Conf. Systems Integration*, pages 136–143, April 1990.

[384] Stonebraker M. and Rowe L.A. The Design of POSTGRES. In *ACM-SIGMOD*, 1986.

[385] Yoelle S. Maarek, Daniel M. Berry, and Gail E. Kaiser. An Information Retrieval Approach for Automatically Constructing Software Libraries. *IEEE Transactions on Software Engineering*, 17(8):800–813, August 1991.

[386] Wendy E. Mackay and Glorianna Davenport. Virtual Video Editing in Interactive Multimedia Applications. *Communications of ACM*, 32(7):802–809, July 1989.

[387] S. Manivannan and C.D. Pegden. A Rule-Based Simulator for Modeling Just-in-Time Manufacturing System (JITSAI). *Simulation*, 52(8):109–117, 1990.

[388] Frank A. Manola and Jack A. Towards a General Spatial Model for Object Oriented DBMS. In *Proceedings of the 12'th International Conference on Very Large Data Bases*, 1986.

[389] V.M. Markowitz. Representing Processes in the Extended Entity-Relationship Model. In *Proc. 6th IEEE Int'l Conf. on Data Engineering*, pages 103–110, 1990.

[390] Y. Masunaga. Design Issues of OMEGA: An Object-Oriented Multimedia Database Management System. *Journal of Information Processing*, 14(1):60–74, 1991.

[391] Yoshifumi Masunaga. Design Issues of OMEGA: An Object-Oriented Multimedia Database Management System. *Journal of Information Processing*, 14(1):60–74, 1991.

[392] M. Maybee, L.J. Osterweil, and S.D. Sykes. Q: A Multi-lingual Interprocess Communications System for Software Environment Implementation. Technical Report CU–CS–476–90, University of Colorado, Department of Computer Science, Campus Box 430, University of Colorado, Boulder, CO 80309–0430, 1990.

[393] Mark Maybee. Q: A Multi-lingual Interprocess Communications System. Reference Manual. Technical report, University of Colorado, Department of Computer Science, Campus Box 430, University of Colorado, Boulder, CO 80309–0430, 1990.

[394] James Mayfield and Charles Nicholas. SNITCH: Augmenting Hypertext Documents with a Semantic Net. In *Proceedings of the ISMM First International Conference on Information and Knowledge Management*, pages 146–152, Baltimore, MD, U.S.A., November 1992. The International Society for Mini and Microcomputers.

[395] D. R. McCarthy and U. Dayal. The Architecture of an Active Database Management System. *Proc. ACM-SIGMOD 1989 Int'l Conf. Management of Data*, pages 215–224, May 1989.

[396] E. McKenzie and R. Snodgrass. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, 23(4), December 1991.

[397] C. Meadows and C. Landwehr. Designing a Trusted Application in an Object-Oriented Data Model. *Research Directions in Database Security, Teresa F. Lunt, ed., Springer-Verlag, Berlin*, 1992.

[398] Arnold C. Meltzer and Gerald Kowalski. Text Searching Using an Inversion Database Consisting of Trigrams. In *IEEE Proceedings of the Second International Conference on Computers and Applications*, pages 65–69, 1987.

[399] R. Merton. Theory of Rational Option Pricing. *The Bell J. of Econ. and Mgmt. Sc*, 4(1), 1973.

[400] B. Meyer. Object-Oriented Software Construction. *Prentice Hall*, 1988.

[401] B. Meyer. An Eiffel Collection. Technical Report TR–EI–20/EC, ISE, 270 Storke Road, Suite 7, Goleta CA 93117, U.S.A, November 1990.

[402] B. Meyer. Eiffel: The Language. Object-Oriented Series, New York, 1992.

[403] Sun MicroSystems. XDR: External Data Representation Standard. Technical Report RFC–1014, Sun MicroSystems. Inc., June 1987.

[404] Sun MicroSystems. RPC: Remote Procedure Call Protocol Specification. Technical Report RFD–1057, Sun MicroSystems. Inc., June 1988.

[405] M. S. Miller et al. The Application Accelerator Illustration System. *OOPSAL'86 Proceedings SIGPLAN Not. (ACM) 2,11 (Portland, Oregon)*, November 1986.

[406] H.D. Mills. Stepwise Refinement and Verification in Box-Structured Systems. *IEEE Computer*, pages 23–36, June 1988.

[407] J. Mills. Semantic Integrity of the Totality of Corporate Data. In *Proceedings of the* 1^{st} *International Conference on Systems Integration*, April 1990.

[408] J. Minker and A Rajasekhar. A FixPoint Semantics for Disjunctive Logic Programs. *Journal of Logic Programming*, 1990.

[409] P. Mishra and M.H. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, 24(1):63–113, Mar 1992.

[410] C. Mohan, I. Narang, and S. Silen. Solutions to Hot Spot Problems in a Shared Disks Transaction Environment. *Proceedings of 4th International Workshop on High Performance Transaction Systems*, September 1991.

[411] L. Mohan and R. Kashyap. An Object Oriented Knowledge Representation for Spatial Information. *IEEE Transactions on Software Engineering*, 1988.

[412] L. Mohan and R.L. Kashyap. A Visual Query Language for Graphical Interaction with Schema-Intensive Database. *To appear on IEEE Trans. on Data and Knowledge Engineering.*

[413] D.E. Monarchi and G.I. Puhr. A Research Typology for Object-Oriented Analysis and Design. *Communications of the ACM*, September 1992.

[414] R. Montague. Pragmatics. In R. Klibansky, editor, *Contemporary Philosophy: A Survey*, pages 102–122. La Nuova Italia Editrice, Florence, 1968.

[415] A. Montgomery, D. D'Souza, and S. Lee. The Cost of Relational Algebraic Operations on Skewed Data: Estimates and Experiments. *Information Processing 83, IFIP*, pages 235–241, 1983.

[416] D.J. Moore. Multimedia Presentation Development Using the Audio Visual Connection. *IBM Systems Journal*, 29(4):494–508, 1990.

[417] C. Mosher. *Postgres Reference Manual.* Unpublished Manuscript, 1991.

[418] J. Eliot B. Moss and Tony Hosking. *Managing Persistent Data With Mneme: User's Guide to the Client Interface.* University of Massachusetts, Object Oriented Systems Laboratory, Department of Computer and Information Science, University of Massachusetts, Amherst, MA 01003.

[419] A. Motro and P. Buneman. Constructing Superviews. *ACM SIGMOD International Conference on Management of Data*, 1981.

[420] James G. Mullen, Jagannathan Srinivasan, Prasun Dewan, and B. Bhargava. Supporting Queries in the O-Raid Object-Oriented Database System. In *Proceedings of the International Computer Software and Applications Conference*, 1990. to appear.

[421] I. S. Mumick, H. V. Jagadish, and O. Shmueli. Events with Attributes in an Active Database. Technical report, 1993.

[422] Philip Musgrove. The General Theory of Gerrymandering. *American Political Series, Ed. R.B.Ripley. Sage Publications Inc.*, 1977.

[423] N. Naffah. Multimedia Applications. *Computer Communications*, 13(4):243–249, May 1990.

[424] G. Nagy and S. Wagle. Geographic Data Processing. *Computing Surveys*, June 1979.

[425] Ataru T. Nakagawa and Kokichi Futatsugi. Software Process a la Algebra: OBJ for OBJ. *IEEE Transactions on Software Engineering*, 1990.

[426] I. Nassi and B. Shneiderman. Flowchart Techniques for Structured Programming. *SIGPLAN Notices*, 8(8), 1973.

[427] S. Navathe and R. Ahmed. TSQL - A Language Interface for Temporal Databases. In *Proceedings of Temporal Aspects of Information Systems*, 1987.

[428] S. Navathe and R. Ahmed. Version Control and Management in CAD Databases. Technical report, Univ of Florida, 1990.

[429] S.B. Navathe and Rafi Ahmed. A Temporal Relational Model and a Query Language. *International Journal of Information Sciences*, March 1989.

[430] Shamkant B. Navathe. Evolution of Data Modeling for Databases. *Communications of the ACM*, 35(9), September 1992.

[431] J. M. Nerson. Object-Oriented Analysis and Design in the 'Business Class' Project. *OOPSLA, Phoenix, Arizona*, 1991. Submitted to the Workshop on Object-Oriented (Domain) Analysis.

[432] J. M. Nerson. Applying Object-Oriented Analysis and Design. *Communications of the ACM*, 35(9), September 1992.

[433] New York University, New York, NY 10012. *GRIFFIN: Language Reference Manual*, October 1992. Version 1.09.

[434] T. L. Nixon, K. Lubenow, J. Srinivasan, P. Dewan, and B. Bhargava. Building a User-Interface for the O-Raid Database System Using the Suite System. In *Proceedings of the Second International Conference on Systems Integration*, June 1992.

[435] E. Nordhagen. Generic Object-Oriented Systems. *Proceedings of TOOLS'89, Paris, France.*, November 1989.

[436] F. Olken. *Random Sampling from Databases.* Unpublished Manuscript, 1992.

[437] E. Omiecinski and E. T. Lin. The Adaptive-Hash Join Algorithm for a Hypercube Multicomputer. *IEEE Transactions on Parallel and Distributed System*, 3(3):334–349, May 1992.

[438] B.C. Ooi, R.S. Davis, and K.J. McDonell. Extending a DBMS for Geographic Applications. In *Proceedings of the Fifth IEEE International Conference on Data Engineering*, pages 590–597, 1989.

[439] P. V. Oosterom and J. Bos. An Object Oriented Approach to Designing a Geographical Information Systems. *Comp. and Graphics*, 14, 1989.

[440] Jack A. Orenstein and F. A. Manola. An Efficient Pictorial Database System for PSQL. *IEEE Transactions on Software Engineering*, 14(5):639–650, May 1988.

[441] Jack A. Orenstein and Frank A. Manola. PROBE: Spatial Data Modeling and Query Processing in an Image Database Application. *IEEE Transactions on Software Engineering*, 14(5):611–629, May 1988.

[442] Bernstein P., V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison Wesley, 1987.

[443] Chris D. Paice. Constructing Literature Abstracts by Computer: Techniques and Prospects. *Information Processing and Management*, 26(1):171–186, 1990.

[444] Chris D. Paice. A Thesaural Model of Information Retrieval. *Information Processing and Management*, 27(5):433–447, 1991.

[445] Abhay K. Parekh and Robert G. Gallager. A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single Node Case. In *Proc. IEEE INFOCOM '92*, 1992.

[446] C. Parent and S. Spaccapietra. An Entity-Relationship Algebra. *1st IEEE International Conference on Data Engineering*, 1984.

[447] C. Parent and S. Spaccapietra. ERC+: An Object Based Entity-Relationship Approach. In P.Loucopoulos and R.Zicari, editors, *Conceptual Modelling, Databases and CASE: An Integrated View of Information Systems Development.* John Wiley, 1992.

[448] Gurudatta M. Parulkar. The Next Generation of Internetworking. *Computer Communications Review*, 20(1):18–43, January 1990.

[449] Roy D. Pea. The Collaborative Visualization Project. *Communications of ACM*, 36(5):60–63, May 1993.

[450] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[451] Y. Perl, J. Geller, E.J. Neuhold, and V. Turau. The Dual Model for Object-Oriented Databases. Technical report, 1991. Research Report CIS-91-30.

[452] J.L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice Hall, 1981.

[453] A. Poggio, J.J. Garcia-Luna-Aceves, E.J. Craighill, D. Moran, L. Aguilar, D. Worthington, and J. Hight. CCWS: A Computer-Based Multimedia Information System. *Computer*, 18(10):92–103, October 1985.

[454] C. A. Pogue, E. M. Rasmussen, and P. Willett. Searching and Clustering of Databases Using the icl Distributed Array Processor. *Parallel Computing*, 8:399–407, October 1988.

[455] C. A. Pogue and P. Willett. Use of Text Signatures for Document Retrieval in a Highly Paralell Environment. *Parallel Computing*, 4:259–268, June 1987.

[456] C. Pu and A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. *ACM SIGMOD*, May 1991.

[457] J. M. Purtilo. The Polylith Software Bus. Technical Report UMIACS–TR–90–65, CS–TR–2469, University of Maryland, Department of Computer Science, University of Maryland, College Park, MD 20742, May 1990.

[458] J. M. Purtilo and Joanne M. Atlee. Module Reuse by Interface Adaptation. Technical report, University of Maryland, Computer Science Department, University of Maryland, College Park, Maryland 20742, August 1989.

[459] Y. Rabinovich, A. Sinclair, and A. Wigderson. Quadratic Dynamical Systems. In *Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science*, pages 304–313, September 1992.

[460] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A Model of Authorization for Next-Generation Database Systems. *ACM Trans. on Database Systems*, 16(1):88–131, March 1991.

[461] F. Rabitti and P. Savino. Automatic Image Indexation to Support Content-Based Retrieval. *Information Processing and Management*, 28(5):547–565, 1992.

[462] E. Rahm. Use of Global Extended Memory for Distributed Transaction Processing. *Proceedings of 4th International Workshop on High Performance Transaction Systems*, September 1991.

[463] K. Ramamritham. Real-Time Databases. *Journal of Distributed and Parallel Database Systems*, 1(1), 1992.

[464] P. Venkat Rangan, Harrick M. Vin, and Srinivas Ramanathan. Communication Architectures and Algorithms for Media Mixing in Multimedia Conferences. *IEEE/ACM Transactions on Networking*, 1(1):20–30, February 1993.

[465] Erwin P. Rathgeb. Modeling and Performance Comparison of Policing Mechanisms for ATM Networks. *IEEE Journal on Selected Areas in Communications*, 9(3):325–334, April 1991.

[466] O. Ratib and H.K. Huang. Low Cost Desktop Image Analysis Workstation with Enhanced Interactive User Interface. *Proc. SPIE Medical Imaging III: Image Capture and Display*, 1091:184–190, January 1989.

[467] Lee Ratzan. Building an Internet Browser. *UNIX Review*, 10(1):25–, January 1992.

[468] T. Reenskaug and E. Nordhagen. The Description of Complex Object-Oriented Systems: Version 1. *Senter for Industriforskning, Oslo, Norway*, 1989.

[469] D. Reiner. The Kendall Square Query Decomposer. *Proceedings of 2nd Intl. Conf. on Parallel and Distributed Information Systems*, pages 36–37, January 1993.

[470] T. Risch. Monitoring Database Objects. In *Proceedings of the 15th International Conference on Very Large Databases*, pages 445–453, 1989.

[471] S. E. Robertson. The Methodology of Information Retrieval Experiment. In K. Sparck Jones, editor, *Information Retrieval Experiment*, pages 9–31. Butterworths, 1981.

[472] B. Vincent Robinson. Interactive Machine Acquisition of a Fuzzy Spatial Relation. *Computers and Geosciences*, Vol 16, No 6 1990.

[473] G. C. Roman. Formal Specifications of Geographical Data Processing Requirements. In *Proceedings of the Fifth IEEE International Conference on Data Engineering*, 1986.

[474] N. Roussopoulos, C. Faloutsos, and T. Sellis. An Efficient Pictorial Database System for PSQL. *IEEE Transactions on Software Engineering.*, May 1988.

[475] N. Roussopoulos and H. Kang. A Pipeline N-Way Join Algorithm Based on the 2-Way Semijoin Program. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, December 1991.

[476] N. Roussopoulos and D. Leifker. Direct Spatial Search on Pictorial Databases Using Packet R-Trees. *Proc. '85 SIGMOD*, pages 17–31, May 1985.

[477] W.B. Rubenstein. A Database Design for Musical Information. *Proc. ACM SIGMOD Conf. on Management of Data*, pages 479–490, May 1987.

[478] Gerda Ruge. Experiments on Linguistically-Based Term Associations. *Information Processing and Management*, 28(3):317–332, 1992.

[479] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying Interdatabase Dependencies in a Multidatabase Environment. *IEEE Computer*, 24(12):46–52, December 1991.

[480] G. Salton and C. Buckley. Parallel Text Search Methods. *Communications of ACM*, 31(2):202–215, February 1988.

[481] G. Salton and C. Buckley. Term Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(3):513–524, 1988.

[482] G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw Hill, 1983.

[483] H. Samet. The Quadtree and Other Hierachical Data Structures. *ACM Computing surveys*, 16:187–260, May 1984.

[484] H. Samet. The Quadtree and Related Data Structures. *Computing Surveys*, 16(2):187–260, June 1984.

[485] H. Samet. Applications of Spatial Data Structures. *Addison Wesley Publications*, 1989.

[486] H. Samet. The Design and Analyses of Spatial Data Structures. *Addison Wesley Publications.*, 1989.

[487] R. S. Sandhu, R. Thomas, and S. Jajodia. Supporting Timing Channel Free Computations in MultiLevel Secure Object-Oriented Databases. *Database Security V: Status and Prospects, (C. E. Landwehr and S. Jajodia, eds.), North-Holland*, pages 297–314, 1992.

[488] N. Sarda. Extensions to SQL for Historical Databases. *IEEE Tran. on Knowledge and Data Engineering*, 2(2):220–230, 1990.

[489] S. Sarin, M. DeWitt, and R. Rosenburg. Overview of SHARD: A System for Highly Available Replicated Data. Technical Report CCA-88-01, Computer Corporation of America, May 1988.

[490] S. K. Sarin. Robust Application Design in Highly Available Databases. In *5th Symposium on Reliability in Distributed Systems*, pages 87–94, 1986.

[491] S. K. Sarin and N. A. Lynch. Discarding Obsolete Information in a Replicated Database System. *IEEE Transactions on Software Engineering*, SE-13(1):39–47, Jan 1987.

[492] R.M. Sasnett. Reconfigurable Video. *M.S. Thesis, MIT*, 1986.

[493] H.-J. Schek, H.-B. Paul, and G. Weikum. The DASDBS Project: Objectives, Experiences, and Future Prospects. In *IEEE Transactions on Knowledge and Data Engineering*, volume 2, 1990.

[494] H.-J. Schek and M.H. Scholl. The Relational Model with Relation-Valued Attributes. *Information Systems*, 11(2), 1986.

[495] Peter Scheuermann and Clement Yu. Report on the Workshop on Heterogeneous Database Systems Held at Northwestern University, Evanston, Illinois, December 11–13, 1989. *SIGMOD RECORD*, 19(4), December 1990.

[496] A. Schilberschatz, M. Stonebraker, and J.D. Ullman. Database Systems: Achievements and Opportunities. *SIGMOD RECORD*, 19(4), December 1990.

[497] D. Schneider. Complex Query Processing in Multiprocessor Database Machines. Technical Report Tech. Rep. 965, Computer Science Department, University of Wisconsin- Madison, September 1990.

[498] D. Schneider and D. J. DeWitt. Tradeoffs in Processing Complex Join Queries via Hashing in Multiprocessor Database Machines. *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 469–480, August 1990.

[499] M. Scholl and A. Voisard. Thematic Map Modeling. *Lecture Notes in Computer Science*, pages 167–190, 1989. Vol. 409.

[500] M.H. Scholl, C. Laasch, and M. Tresh. Updatable Views in Object-Oriented Databases. In *2nd International Conference on Deductive and Object-Oriented Databases*, December 1991.

[501] M.H. Scholl, H.-B. Paul, and H.-J. Schek. Supporting Flat Relations by Nested Relational Kernel. In *13th International Conference on Very Large Data Bases*, September 1987.

[502] M.H. Scholl, H.-J. Schek, and M. Tresh. Object Algebra and Views for Multi-Objectbases. In *International Workshop on Distributed Object Management*, August 1992.

[503] Michael F. Schwartz. The NeTworked Resource Discovery Project: Goals, Design, and Research Efforts. Technical Report CU-CS-387-88, University of Colorado, Boulder, Colorado 80309, May 1988.

[504] Michael F. Schwartz. A Scalable, Non-Hierarchical Resource Discovery Mechanism Based on Probabilistic Protocols. Technical Report CU-CS-474-90, University of Colorado, Boulder, Colorado 80309, June 1990.

[505] Michael F. Schwartz, Darren R. Hardy, William K. Heinzman, and Glenn C. Hirschowitz. Supporting Resource Discovery Among Public Internet Archives Using a Spectrum of Information Quality. In *11th International Conference on Distributed Computing Systems*, May 1991.

[506] B. Seeger and H.P. Kriegel. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In *Proceedings of the 16'th International Conference on Very Large Data Bases*, pages 590–601, 1986.

[507] A. Segev and H. Gunadhi. Event-Join Optimization in Temporal Relational Databases. In *Proceedings of the Int. Conf. on Very Large Databases*, pages 205–215, August 1989.

[508] A Segev and A. Shoshani. Logical Modeling of Temporal Data. *Proceedings of the ACM-SIGMOD International Conference on Management of Data.*, 1987.

[509] A. Segev and A. Shoshani. The Representation of a Temporal Data Model in the Relational Environment. In M. Rafanelli, J.C. Klensin, and P. Svensson, editors, *Lecture Notes in Computer Science No. 339*, pages 39–61. Springer-Verlag, 1988.

[510] A. Segev and J.L. Zhao. Data Management for Large Rule Systems. In *Proceedings of the Int. Conf. on Very Large Databases*, pages 297–307, September 1991.

[511] P. G. Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access Path Selection in a Relational Database Management System. *Proceedings of ACM SIGMOD*, pages 23–34, 1979.

[512] T. Sellis, N. Roussopoulos, and C. Faloutsos. The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems. In *Proceedings of the 13'th International Conference on Very Large Data Bases*, pages 507–518, 1987.

[513] R. Sharma. A Generic Machine for Paralell Information Retrieval. *Information Processing and Management*, 25(3):223–235, 1989.

[514] A. Sheth and G. Karabatis. Multidatabase Interdependencies in Industry. *ACM SIGMOD*, May 1993.

[515] A. Sheth and J. Larson. Federated Database Systems for Managing Distributed Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), September 1990.

[516] A. Sheth, J. Larson, A. Cornelio, and S. Navathe. A Tool for Integrating Conceptual Schemas and User Views. In *4th IEEE International Conference on Data Engineering*, February 1988.

[517] A. Sheth, Y. Leu, and A. Elmagarmid. Maintaining Consistency of Interdependent Data in Multidatabase Systems. Technical Report CSD-TR-91-016, Computer Sciences Department, Purdue University, March 1991.

[518] A. Sheth and M. Rusinkiewicz. Management of Interdependent Data: Specifying Dependency and Consistency Requirements. In *Proceedings of the Workshop on the Management of Replicated Data*, Houston, TX, November 1990.

[519] A. Sheth, M. Rusinkiewicz, and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In A. Elmagarmid, editor, *Database Transaction Models for Advanced Applications*, chapter 14. Morgan-Kaufmann, February 1992.

[520] D. Shmoys, J. Wein, and D. Williamson. Scheduling Parallel Machines On-Line. In *Proceedings of the 32nd Annual Symposium on the Foundations of Computer Science*, pages 131–140, October 1991.

[521] B. Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *IEEE Computer*, 16:57–69, 1983.

[522] Y.M. Shyy. *The Design and Implementation of a Knowledge Base Programming Language for Evolutionary Prototyping of Software Systems.* PhD thesis, University of Florida, Computer and Information Science Department, 1992.

[523] Y.M. Shyy and S.Y.W. Su. K: a High-Level Knowledge Base Programming Language for Advanced Database Applications. In *ACM SIGMOD*, pages 338–347, Denver, Colorado, May 29-31 1991.

[524] A. Silberschatz, M. Stonebraker, and J. Ullman. Database Systems: Achievements and Opportunities. *CACM*, 34(10):110–120, October 1991.

[525] E. Simon and P. Valduriez. Integrity Control in Distributed Database Systems. In *Proceedings of the* 20^{th} *Hawaii International Conference on System Sciences*, 1986.

[526] D. Sleator. A 2.5 Times Optimal Algorithm for Packing in Two Dimensions. *Information Processing Letters*, 10(1):37–40, February 1980.

[527] J. Smith and C Smith. Database Abstraction: Aggregation and Generalization. *ACM Transactions on Database Systems*, 2(2), June 1976.

[528] R. Snodgrass. The Temporal Query Language TQuel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[529] R. Snodgrass. The Temporal Query Language TQuel. *ACM Trans. on Database Systems*, 12(2):247–298, June 1987.

[530] R. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of ACM-SIGMOD 1985 International Conference on Management of Data*, pages 236–246, Austin, TX., May 1985.

[531] R. Snodgrass, S. Gomez, and L.E. Jr. McKenzie. Aggregates in the Temporal Query Language TQUEL. Technical Report Technical Report TR89-26, University of Arizona, 1992.

[532] A. Snyder. An Abstract Model for Object-Oriented Systems. *STL-90-22., Software Technology Laboratory, HP Labs, Palo Alto, CA.*, 1990.

[533] W. Kirk Snyder. *The SETL2 Programming Language.* Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, September 9 1990.

[534] W. Kirk Snyder. *The SETL2 Programming Language: Update On Current Developments.* Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, September 7 1990.

[535] Snodgrass Soo, M., C. R., Dyreson, C.S. Jensen, and N. Kline. Architectural Extensions to Support Multiple Calendars. Technical Report TempIS Technical Report 32, University of Arizona, 1992.

[536] S. Spaccapietra and C. Parent. Conflicts an Correspondence Assertions in Interoperable Databases. *SIGMOD RECORD*, 20(4), December 1991.

[537] S. Spaccapietra, C. Parent, and Y. Dupont. Model-Independent Assertions for Integration of Heterogeneous Schemas. *Very Large Data Base Jopurnal*, 1(1), July 1992.

[538] S. Spaccapietra, C. Parent, K. YtongNon, and M. S. Abaidi. Generalizations: A Formal and Flexible Approach. In *1st Conference on Management of Data*, November 1989.

[539] D. L. Spooner. The Impact of Inheritance on Security in Object-Oriented Database Systems. *Database Security, II: Status and Prospects, Carl E. Landwehr, ed., North-Holland, Amsterdam*, pages 141–160, 1989.

[540] J. Srinivasan, Y. Jiang, Y. Zhang, and B. Bhargava. Experiments on O-Raid Distributed Object-Oriented Database System. In *Proceedings of the First International Conference on Information and Knowledge Management*, November 1992.

[541] Jagannathan Srinivasan. *Replication and Fragmentation of Composite Objects in Distributed Database Systems.* PhD thesis, Purdue University, Aug 1992.

[542] S. Sripada. A Uniform Model for Temporal Object-Oriented Databases. In *International Conference on Very Large Databases*, 1988.

[543] William Stallings, editor. *Advances in ISDN and Broadband ISDN.* IEEE Computer Society Press, 1992.

[544] C. Stanfill. Partitioned Posting Files: A Paralell Inverted File Structure for Information Retrieval. In *Proceedings of the 1990 ACM SIGIR*, pages 413–428, September 1990.

[545] C. Stanfill and B. Kahle. Parallel Free-Text Search on the Connection Machine System. *Communications of ACM*, 29(12):1229–1239, December 1986.

[546] C. Stanfill, R. Thau, and D. Waltz. A Paralell Indexed Algorithm for Information Retrieval. In *Proceedings of the 1989 ACM SIGIR*, pages 88–97, June 1989.

[547] Jr. Stanley M. Sutton. *APPL/A: A Prototype Language for Software-Process Programming.* PhD thesis, University of Colorado, August 1990.

[548] R. Steinmetz. Synchronization Properties in Multimedia Systems. *IEEE Journal on Selected Areas in Communications*, 8(3):401–412, April 1990.

[549] H. Stone, J. L. Wolf, and J. Turek. Optimal Partitioning of Cache Memory. *IEEE Transactions on Computers*, 41(9):1054–1068, 1992.

[550] H. S. Stone. Parallel Querying of Large Databases: A Case Study. *IEEE Computer*, 20(10):11–21, October 1987.

[551] M. Stonebraker. The Case for Shared Nothing. *IEEE Database Engineering*, 9(1), 1986.

[552] M. Stonebraker. Inclusion of New Types in Relational Database Systems. In M. Stonebraker, editor, *Readings in Database Systems*, pages 480–487. Morgan Kaufman Publishers, Inc., 1990.

[553] M. Stonebraker, E. Hanson, and S. Potamianos. The POSTGRES Rule Manager. *IEEE Transactions on Software Engineering*, 14(7):897–907, July 1988.

[554] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On Rules, Procedures, Caching and Views in Data Base Systems. In *Proceedings of ACM SIGMOD International Conference on the Management of Data*, June 1990.

[555] M. Stonebraker, R. Katz, D. Patterson, and J. Ousterhout. The Design of XPRS. *Proceedings of the 14th International Conference on Very Large Data Bases*, pages 318–330, 1988.

[556] M. Stonebraker and G. Kemnitz. The POSTGRES Next-Generation Database Management System. *CACM*, 34(10):78–93, October 1991.

[557] Michael Stonebraker and Greg Kemnitz. The POSTGRES Next-Generation Database Management System. *Communicatons of the ACM*, 34(10), October 1991.

[558] P. D. Stotts and R. Furuta. Petri-Net-Based Hypertext : Document Structure with Browsing Semantics. *ACM Trans. on Office Automation Systems*, 7(1):3–29, January 1989.

[559] J. P. Strickland, P. P. Uhrowczik, and V. L. Watts. IMS/VS: An Evolving System. *IBM Systems Journal*, V-21:490–510, 1982.

[560] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, Reading, Mass., 1986.

[561] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1986.

[562] B. Stroustrup. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, 1990.

[563] B. Stroustrup. *The C++ Programming Language (2nd Ed.).* Addison-Wesley, 1991.

[564] S. Su, V. Krishnamurthy, and H. Lam. An Object-Oriented Semantic Association Model OSAM*. In S. Kumara et al., editors, *Artificial Intelligence Manufacturing Theory and Practice*, pages 463–494. American Inst. of Indus. Engr., 1989.

[565] S.Y.W. Su. SAM*: A Semantic Association Model for Corporate and Scientific-Statistical Databases. *Information Sciences*, 29:151–199, 1983.

[566] Y.H.S. Su and H. M. Chen. A Temporal Knowledge Representation Model *OSAM**/T and its Query Language OQL/T. In *Proceedings of the Int. Conf. on Very Large Databases*, pages 431–442, September 1991.

[567] R. Subramanian. *Object Oriented Modeling and Intelligent Query Processing in Spatial Databases.* PhD thesis, Rutgers University, 1992.

[568] J.S. Sventek. An Architecture for Supporting Multi-Media Integration. *Proc. IEEE Computer Society Office Automation Symposium*, pages 46–56, April 1987.

[569] A. Swami. Optimization of Large Join Queries: Combining Heuristics with Combinatorial Techniques. *Proceedings of ACM SIGMOD*, pages 367–376, 1989.

[570] A. Swami and A. Gupta. Optimization of Large Join Queries. *Proceedings of ACM SIGMOD*, pages 8–17, 1988.

[571] Sellis T. and et. el. Implementing Large Production Systems in a DBMS Environment. In *ACM-SIGMOD*, 1988.

[572] Thomasma T., Y. Mao, and Ulgen O.M. Manufacturing Simulation in Smalltalk. In *SCS Multiconference on Object-Oriented Simulation*, 1990.

[573] A.U. Tansel, M.E. Arkun, and G. Ozsoyoglu. Time-by-Example Query Language for Historical Databases. *IEEE Trans. on Software Engineering*, 15(4):464–478, April 1989.

[574] A. Tantawi, D. Towsley, and J. Wolf. Optimal Allocation of Multiple Class Resources in Computer Systems. *Proceedings of the ACM Sigmetrics Conference*, pages 253–260, May 1988.

[575] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michael Young. Foundations for the Arcadia Environment Architecture. In *In Proceedings of ACM SIGSOFT '88: Third Symposium on Software Development Environments, Boston*, pages 1–13, November 1988. Appeared as Sigplan Notices 24(2) and Software Engineering Notes 13(5).

[576] Richard N. Taylor, Frank C. Belz, Lori A. Clarke, Leon Osterweil, Richard W. Selby, Jack C. Wileden, Alexander L. Wolf, and Michael Young. Issues Encountered in Building a Flexible Software Development Environment: Lessons Learned from the Arcadia Project. In *Proceedings SDE5*, December 1992.

[577] Teradata. DBC/1012 Database Computer System Manual Release 2.0. Technical Report Document C10-0001-02, Teradata Corperation, November 1985.

[578] Robert B. Terwilliger and R. H. Campbell. PLEASE: a Language for Incremental Software Development. In *Proceedings of the 4th International Workshop on Software Specification and Design*, pages 249–256, April 1987.

[579] Robert B. Terwilliger and Roy H. Campbell. PLEASE: Executable Specifications for Incremental Software Development. *Journal of Systems and Software*, 10:97–112, 1989.

[580] R. Tewari. *Robustness in Replicated Databases.* PhD thesis, Rutgers University, 1990.

[581] R. Tewari and N.R. Adam. Using Semantic Knowledge of Transactions to Improve Recovery and Availability of Replicated Data. *Information Systems*, 17(2):477–490, Nov-Dec 1992.

[582] M. B. Thuraisingham. A MultiLevel Secure Object-Oriented Data Model. In *Proc. 12th National Computer Security Conference*, pages 579–590, October 1989.

[583] M. B. Thuraisingham. Mandatory Security in Object-Oriented Database System. In *Proc. Conf. on Object-Oriented Programming: Systems, Languages,and Applications*, pages 203–210, October 1989.

[584] H. Tirri, J. Srinivasan, and B. Bhargava. Transactions for Fragmented Composite Objects. Technical report, Purdue University, 1991.

[585] H. Tirri, J. Srinivasan, and B. Bhargava. Integrating Distributed Data Sources Using Federated Objects. In *International Workshop on Distributed Object Management, Edmonton, Canada*, July 1992.

[586] F.W. Tompa. A Data Model for Flexible Hypertext Database Systems. *ACM Trans. on Information Systems*, 7(1):85–100, January 1989.

[587] S.T. Treves, E.S. Hashem, B.A. Majmudar, K. Mitchell, and D.J. Michaud. Multimedia Communications in Medical Imaging. *IEEE Journal on Selected Areas in Communications*, 10(7):1121–1132, September 1992.

[588] J. Turek, J. Wolf, K. Pattipati, and P. S. Yu. Scheduling Parallelizable Tasks: Putting it all on the Shelf. *Proceedings of the ACM Sigmetrics Conference, Newport, RI*, pages 225–236, June 1992.

[589] J. Turek, J. Wolf, and P. Yu. Approximate Algorithms for Scheduling Parallelizable Tasks. In *Proceedings of the 4th Annual Symposium on Parallel Algorithms and Architectures, San Diego, CA*, pages 323–332, June 1992.

[590] H. R. Turtle and W. B. Croft. Evaluation of an Inference Network-Based Retrieval Model. *ACM Transactions on Information Systems*, 3:187–222, 1991.

[591] A. Turtur et al. IDB: An Image Database System. *IBM Journal of Research and Development*, 35(1):88–95.

[592] Dayal U., Buchmann A., Goldhirsch D., and Heiler S. PROBE - A Research Project in Knowledge Directed DBMS. Technical Report CCA-85-03, Computer Corporation of America, 1986.

[593] J. D. Ullman. *Principles of Database and Knowledge-Base Systems.* Computer Science Press, 1988.

[594] J.D. Ullman. *Principles of Database Systems.* Computer Science Press, Potomac, MD, 1980.

[595] U.S.Dept. of Commerce, Bureau of the Census. TIGER/Line Precensus Files. 1990.

[596] P. Valduriez and G. Gardarin. Join and Semijoin Algorithms for a Multiprocessor Database Machine. *ACM Transactions on Database Systems*, 9(1):133–161, March 1984.

[597] P. Van Hentenryck. *Constraint Satisfaction in Logic Programming.* Logic Programming. MIT Press, 1989.

[598] M. Vardi. The Complexity of Relational Query Languages. In *Proceedings of the 14th STOC*, pages 137–146, 1982.

[599] Y. Vassiliou. Null Values in Database Management : A Denotational Semantics Approach. In *Proceedings of ACM SIGMOD International Conference on the Management of Data*, May 1979.

[600] J. Verso. Verso: A Database Machine Based on Non-1NF Relations. Technical Report 523, INRIA, 1986.

[601] Q. Wang and H. Cheng. A Heuristic of Scheduling Parallel Tasks and it Analysis. *SIAM Journal on Computing*, 21(2):281–294, April 1992.

[602] Jon Ward. A Candidate HUMD ProtoTech Demonstration Vehicle: ProtoTech Extensions to a Structured Design Methodology. Honeywell Technical Report CS–R92–015, Honeywell Systems and Research Center, July 7th 1992. DRAFT.

[603] Peter Wegner. Dimensions of Object-Oriented Modeling. *IEEE Computer*, 25(10):12–20, October 1992.

[604] W. Weihl. Commutativity-Based Concurrency Control for Abstract Data Types. In *21st Hawaii International Conference on System Sciences*, pages 205–214, 1988.

[605] K.H. Weiss. Formal Specification and Continuous Media. *Proc. 1st Intl. Workshop on Network and Operating Support for Digital Audio and Video*, November 1990.

[606] David L. Wells, Craig W. Thompson, and Jose A. Blakeley. DARPA Open Object-Oriented Database System. Technical report, Texas Instrument Incorporated, P.O. Box 655474, MS 238, Dallas, Texas 75265, U.S.A., 1992.

[607] Marek Wernik, Osama Aboul-Magd, and Henry Gilbert. Traffic Management for B-ISDN Services. *IEEE Network*, 6(5):10–19, September 1992.

[608] R.A. Whitehurst. Integrating Object-Oriented and Rule-Based Approaches with the Simulation Environment. In *30th Annual Army Operations Research Society Symposium*, 1991.

[609] G. Wiederhold and X. Qian. Modeling Asynchrony in Distributed Databases. In *Intl. Conf. on Data Engineering*, February 1987.

[610] Gio Wiederhold. Intelligent Integration of Diverse Information. In *Proceedings of the ISMM First International Conference on Information and Knowledge Management*, pages 1–7, Baltimore, MD, U.S.A., November 1992. The International Society for Mini and Microcomputers.

[611] Jack C. Wileden, Alexander L. Wolf, Charles D. Fisher, and Peri L. Tarr. PGRAPHITE: An experiment in persistent typed Object Management. In *In Proceedings of ACM SIGSOFT 88: Third Symposium on Software Development Environments, Boston*, pages 130–142, November 1988.

[612] Jack C. Wileden, Alexander L. Wolf, William R. Rosenblatt, and Peri L. Tarr. Specification Level Interoperability. In *In Proceedings of the Twelfth International Conference on Software Engineering, Nice*, March 1990.

[613] P. Willett. Recent Trends in Hierarchic Document Clustering: A critical review. *Information Processing and Management*, 24(5):577–597, 1984.

[614] A. Wilschut and P. Apers. Dataflow Query Execution in Parallel Main-Memory Environment. *Proceedings of the 1st Conference on Parallel and Distributed Information Systems*, pages 68–77, December 1991.

[615] R. Wirfs-Brock et al. Designing Object-Oriented Software. *Prentice Hall.*, 1990.

[616] Rebecca Wirfs-Brock. Object Oriented Design: A Responsibility Driven Approach. *OOPSLA'89 Proceedings*, 1989.

[617] Rebecca Wirfs-Brock and Ralph E. Johnson. Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, September 1990.

[618] N. Wirth. *Programs = Algorithms + Data Structures.* Prentice Hall, 1976.

[619] D. Woelk, W. Luther, , and W. Kim. Multimedia Applications and Database Requirements. *IEEE Office Automation Symposium*, pages 180–189, April 1987.

[620] Alexander L. Wolf. The AdaPIC Tool Set: Supporting Interface Control and Analysis Throughout the Software Development Process. *IEEE Transactions on Software Engineering*, 15(3), March 1989.

[621] Alexander L. Wolf and Alan Kaplan. *PIC/ADA Language Reference Manual.* Software Development Laboratory, Computer and Information Science Department, University of Massachusetts, Amherst, Massachusetts 01003, December 1991. Arcadia Design Document UM–91–10.

[622] J. L. Wolf, D. M. Dias, and P. S. Yu. An Effective Algorithm for Parallelizing Sort Merge Joins in the Presence of Data Skew. *Proceedings of the 2nd International Symposium on Databases in Parallel and Distributed Systems*, pages 103–115, July 1990.

[623] J. L. Wolf, D. M. Dias, and P. S. Yu. A Parallel Sort Merge Join Algorithm for Managing Data Skew. *IEEE Transactions on Parallel and Distributed Systems*, pages 70–86, January 1993.

[624] J. L. Wolf, D. M. Dias, P. S. Yu, and J. Turek. An Effective Algorithm for Parallelizing Hash Joins in the Presence of Data Skew. *Proceedings of the 7th International Conference on Data Engineering*, pages 200–209, April 1991.

[625] J. L. Wolf, D. M. Dias, P. S. Yu, and J. Turek. Comparative Performance of Parallel Join Algorithms. *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems*, pages 78–88, December 1991.

[626] J. L. Wolf, B. Iyer K. Pattipati, and J. Turek. Optimal Buffer Partitioning for the Nested Block Join Algorithm. *Proceedings of the 7th International Conference on Data Engineering*, pages 510–519, April 1991.

[627] J. L. Wolf, J. Turek, and P. S. Yu. A Robust Method for Performing a Parallel Query in an Environment with Partial Sharing. *IBM Technical Disclosure Bulletin*, October 1992.

[628] J. L. Wolf, J. T. Turek, M.-S. Chen, and P. S. Yu. On the Optimal Scheduling of Multiple Queries in a Parallel Processing Database Environment. *IBM Research Report, RC 18298*, September 1992.

[629] Wai Yee Peter Wong and Dik Lun Lee. Signature File Methods for Implementing a Ranking Strategy. *Information Processing and Management*, 26(5):641–653, 1990.

[630] Gillian M. Woodruff and Rungroj Kositpaiboon. Multimedia Traffic Management Principles for Guaranteed ATM Network Performance. *IEEE Journal on Selected Areas in Communications*, 8(3):437–446, April 1990.

[631] K. Wu, P. Yu, and C. Pu. Divergence Control for Epsilon Serializability. *Proc. of the 8th Int'l Conference on Data Engineering*, February 1992.

[632] G.T.J Wuu and U. Dayal. A Uniform Model for Temporal Object-Oriented Databases. In *8th Int. Conf. on Data Engineering*, pages 584–593, May 1979.

[633] N. Yankelovich, B.J. Hann, N.K. Meyrowitz, and S.M. Drucker. Intermedia: The Concept and the Construction of a Seamless Information Environment. *IEEE Computer*, 21(1):81–96, January 1988.

[634] R. Yassen, S.Y.W. Su, and H. Lam. An Extensible Kernel Object Management System. *ACM OOPSLA 1991 Conference*, pages 247–263, 1986.

[635] Raj Yavatkar. MCP: A Protocol for Coordination and Temporal Synchronization in Multi-media Collaborative Applications. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, June 1992.

[636] Rajendra S. Yavatkar. Communication Support for Collaborative Multimedia Applications. Technical Report 181-91, Department of Computer Science, University of Kentucky, January 1991.

[637] C. Yu et al. Efficient Placement of Audio Data on Optical Disks for Real-Time Applications. *Communications of the ACM*, 32(7):862–871, July 1989.

[638] P. S. Yu, M.-S. Chen, H. Heiss, and S. H. Lee. On Workload Characterization of Relational Database Environments. *IEEE Transactions on Software Engineering*, 18(4):347–355, April 1992.

[639] P. S. Yu and A. Dan. Effect of System Dynamics on Coupling Architectures for Transaction Processing. *Proceedings of the 8th International Conference on Data Engineering*, pages 458–469, February 1992.

[640] P. S. Yu and A. Dan. Impact of Workload Partitionability on the Performance of Coupling Architectures for Transaction Processing. *Proceedings of 4th IEEE Symposium on Parallel and Distributed Processing*, December 1992.

[641] P. S. Yu, D. M. Dias, J. T. Robinson, B. R. Iyer, and D. W. Cornell. On Coupling Multi-Systems Through Data Sharing. *Proceedings of the IEEE*, 75(5):573–587, May 1987.

[642] S. Zdonik. Object-Oriented Type Evolution. *Advances in Database Programming Languages, F.Bancilhon, and P.Buneman, eds., Addison-Wesley*, pages 277–288, 1990.

[643] S. B. Zdonik and P. Wegner. Language and Methodology for Object-Oriented Database Environments. In *Proc. 19th Annual Hawaii International Conference on System Sciences*, pages 378–387, 1986.

[644] K. Zhao. *Panorama: Dynamic View Construction in Large Multidatabase Systems*. PhD thesis, University of Colorado, Department of Computer Science, University of Colorado, Boulder, CO 80309, 1992.

[645] K. Zhao, R. King, and A. Bouguettaya. Incremental Specification of Views Across Databases. In *First International Workshop on Interoperability in Multidatabase Systems, Kyoto, Japan*, pages 187–190, April 7–9 1991.

[646] G. Zipf. *Human Behavior and the Principle of Least Effort.* Addison-Wesley, MA, 1949.

Lecture Notes in Computer Science

For information about Vols. 1–685
please contact your bookseller or Springer-Verlag

Vol. 686: J. Mira, J. Cabestany, A. Prieto (Eds.), New Trends in Neural Computation. Proceedings, 1993. XVII, 746 pages. 1993.

Vol. 687: H. H. Barrett, A. F. Gmitro (Eds.), Information Processing in Medical Imaging. Proceedings, 1993. XVI, 567 pages. 1993.

Vol. 688: M. Gauthier (Ed.), Ada-Europe '93. Proceedings, 1993. VIII, 353 pages. 1993.

Vol. 689: J. Komorowski, Z. W. Ras (Eds.), Methodologies for Intelligent Systems. Proceedings, 1993. XI, 653 pages. 1993. (Subseries LNAI).

Vol. 690: C. Kirchner (Ed.), Rewriting Techniques and Applications. Proceedings, 1993. XI, 488 pages. 1993.

Vol. 691: M. Ajmone Marsan (Ed.), Application and Theory of Petri Nets 1993. Proceedings, 1993. IX, 591 pages. 1993.

Vol. 692: D. Abel, B.C. Ooi (Eds.), Advances in Spatial Databases. Proceedings, 1993. XIII, 529 pages. 1993.

Vol. 693: P. E. Lauer (Ed.), Functional Programming, Concurrency, Simulation and Automated Reasoning. Proceedings, 1991/1992. XI, 398 pages. 1993.

Vol. 694: A. Bode, M. Reeve, G. Wolf (Eds.), PARLE '93. Parallel Architectures and Languages Europe. Proceedings, 1993. XVII, 770 pages. 1993.

Vol. 695: E. P. Klement, W. Slany (Eds.), Fuzzy Logic in Artificial Intelligence. Proceedings, 1993. VIII, 192 pages. 1993. (Subseries LNAI).

Vol. 696: M. Worboys, A. F. Grundy (Eds.), Advances in Databases. Proceedings, 1993. X, 276 pages. 1993.

Vol. 697: C. Courcoubetis (Ed.), Computer Aided Verification. Proceedings, 1993. IX, 504 pages. 1993.

Vol. 698: A. Voronkov (Ed.), Logic Programming and Automated Reasoning. Proceedings, 1993. XIII, 386 pages. 1993. (Subseries LNAI).

Vol. 699: G. W. Mineau, B. Moulin, J. F. Sowa (Eds.), Conceptual Graphs for Knowledge Representation. Proceedings, 1993. IX, 451 pages. 1993. (Subseries LNAI).

Vol. 700: A. Lingas, R. Karlsson, S. Carlsson (Eds.), Automata, Languages and Programming. Proceedings, 1993. XII, 697 pages. 1993.

Vol. 701: P. Atzeni (Ed.), LOGIDATA+: Deductive Databases with Complex Objects. VIII, 273 pages. 1993.

Vol. 702: E. Börger, G. Jäger, H. Kleine Büning, S. Martini, M. M. Richter (Eds.), Computer Science Logic. Proceedings, 1992. VIII, 439 pages. 1993.

Vol. 703: M. de Berg, Ray Shooting, Depth Orders and Hidden Surface Removal. X, 201 pages. 1993.

Vol. 704: F. N. Paulisch, The Design of an Extendible Graph Editor. XV, 184 pages. 1993.

Vol. 705: H. Grünbacher, R. W. Hartenstein (Eds.), Field-Programmable Gate Arrays. Proceedings, 1992. VIII, 218 pages. 1993.

Vol. 706: H. D. Rombach, V. R. Basili, R. W. Selby (Eds.), Experimental Software Engineering Issues. Proceedings, 1992. XVIII, 261 pages. 1993.

Vol. 707: O. M. Nierstrasz (Ed.), ECOOP '93 – Object-Oriented Programming. Proceedings, 1993. XI, 531 pages. 1993.

Vol. 708: C. Laugier (Ed.), Geometric Reasoning for Perception and Action. Proceedings, 1991. VIII, 281 pages. 1993.

Vol. 709: F. Dehne, J.-R. Sack, N. Santoro, S. Whitesides (Eds.), Algorithms and Data Structures. Proceedings, 1993. XII, 634 pages. 1993.

Vol. 710: Z. Ésik (Ed.), Fundamentals of Computation Theory. Proceedings, 1993. IX, 471 pages. 1993.

Vol. 711: A. M. Borzyszkowski, S. Sokołowski (Eds.), Mathematical Foundations of Computer Science 1993. Proceedings, 1993. XIII, 782 pages. 1993.

Vol. 712: P. V. Rangan (Ed.), Network and Operating System Support for Digital Audio and Video. Proceedings, 1992. X, 416 pages. 1993.

Vol. 713: G. Gottlob, A. Leitsch, D. Mundici (Eds.), Computational Logic and Proof Theory. Proceedings, 1993. XI, 348 pages. 1993.

Vol. 714: M. Bruynooghe, J. Penjam (Eds.), Programming Language Implementation and Logic Programming. Proceedings, 1993. XI, 421 pages. 1993.

Vol. 715: E. Best (Ed.), CONCUR'93. Proceedings, 1993. IX, 541 pages. 1993.

Vol. 716: A. U. Frank, I. Campari (Eds.), Spatial Information Theory. Proceedings, 1993. XI, 478 pages. 1993.

Vol. 717: I. Sommerville, M. Paul (Eds.), Software Engineering – ESEC '93. Proceedings, 1993. XII, 516 pages. 1993.

Vol. 718: J. Seberry, Y. Zheng (Eds.), Advances in Cryptology – AUSCRYPT '92. Proceedings, 1992. XIII, 543 pages. 1993.

Vol. 719: D. Chetverikov, W.G. Kropatsch (Eds.), Computer Analysis of Images and Patterns. Proceedings, 1993. XVI, 857 pages. 1993.

Vol. 720: V.Mařík, J. Lažanský, R.R. Wagner (Eds.), Database and Expert Systems Applications. Proceedings, 1993. XV, 768 pages. 1993.

Vol. 721: J. Fitch (Ed.), Design and Implementation of Symbolic Computation Systems. Proceedings, 1992. VIII, 215 pages. 1993.

Vol. 722: A. Miola (Ed.), Design and Implementation of Symbolic Computation Systems. Proceedings, 1993. XII, 384 pages. 1993.

Vol. 723: N. Aussenac, G. Boy, B. Gaines, M. Linster, J.-G. Ganascia, Y. Kodratoff (Eds.), Knowledge Acquisition for Knowledge-Based Systems. Proceedings, 1993. XIII, 446 pages. 1993. (Subseries LNAI).

Vol. 724: P. Cousot, M. Falaschi, G. Filè, A. Rauzy (Eds.), Static Analysis. Proceedings, 1993. IX, 283 pages. 1993.

Vol. 725: A. Schiper (Ed.), Distributed Algorithms. Proceedings, 1993. VIII, 325 pages. 1993.

Vol. 726: T. Lengauer (Ed.), Algorithms – ESA '93. Proceedings, 1993. IX, 419 pages. 1993

Vol. 727: M. Filgueiras, L. Damas (Eds.), Progress in Artificial Intelligence. Proceedings, 1993. X, 362 pages. 1993. (Subseries LNAI).

Vol. 728: P. Torasso (Ed.), Advances in Artificial Intelligence. Proceedings, 1993. XI, 336 pages. 1993. (Subseries LNAI).

Vol. 729: L. Donatiello, R. Nelson (Eds.), Performance Evaluation of Computer and Communication Systems. Proceedings, 1993. VIII, 675 pages. 1993.

Vol. 730: D. B. Lomet (Ed.), Foundations of Data Organization and Algorithms. Proceedings, 1993. XII, 412 pages. 1993.

Vol. 731: A. Schill (Ed.), DCE – The OSF Distributed Computing Environment. Proceedings, 1993. VIII, 285 pages. 1993.

Vol. 732: A. Bode, M. Dal Cin (Eds.), Parallel Computer Architectures. IX, 311 pages. 1993.

Vol. 733: Th. Grechenig, M. Tscheligi (Eds.), Human Computer Interaction. Proceedings, 1993. XIV, 450 pages. 1993.

Vol. 734: J. Volkert (Ed.), Parallel Computation. Proceedings, 1993. VIII, 248 pages. 1993.

Vol. 735: D. Bjørner, M. Broy, I. V. Pottosin (Eds.), Formal Methods in Programming and Their Applications. Proceedings, 1993. IX, 434 pages. 1993.

Vol. 736: R. L. Grossman, A. Nerode, A. P. Ravn, H. Rischel (Eds.), Hybrid Systems. VIII, 474 pages. 1993.

Vol. 737: J. Calmet, J. A. Campbell (Eds.), Artificial Intelligence and Symbolic Mathematical Computing. Proceedings, 1992. VIII, 305 pages. 1993.

Vol. 738: M. Weber, M. Simons, Ch. Lafontaine, The Generic Development Language Deva. XI, 246 pages. 1993.

Vol. 739: H. Imai, R. L. Rivest, T. Matsumoto (Eds.), Advances in Cryptology – ASIACRYPT '91. X, 499 pages. 1993.

Vol. 740: E. F. Brickell (Ed.), Advances in Cryptology – CRYPTO '92. Proceedings, 1992. X, 593 pages. 1993.

Vol. 741: B. Preneel, R. Govaerts, J. Vandewalle (Eds.), Computer Security and Industrial Cryptography. Proceedings, 1991. VIII, 275 pages. 1993.

Vol. 742: S. Nishio, A. Yonezawa (Eds.), Object Technologies for Advanced Software. Proceedings, 1993. X, 543 pages. 1993.

Vol. 743: S. Doshita, K. Furukawa, K. P. Jantke, T. Nishida (Eds.), Algorithmic Learning Theory. Proceedings, 1992. X, 260 pages. 1993. (Subseries LNAI)

Vol. 744: K. P. Jantke, T. Yokomori, S. Kobayashi, E. Tomita (Eds.), Algorithmic Learning Theory. Proceedings, 1993. XI, 423 pages. 1993. (Subseries LNAI)

Vol. 745: V. Roberto (Ed.), Intelligent Perceptual Systems. VIII, 378 pages. 1993. (Subseries LNAI)

Vol. 746: A. S. Tanguiane, Artificial Perception and Music Recognition. XV, 210 pages. 1993. (Subseries LNAI).

Vol. 747: M. Clarke, R. Kruse, S. Moral (Eds.), Symbolic and Quantitative Approaches to Reasoning and Uncertainty. Proceedings, 1993. X, 390 pages. 1993.

Vol. 748: R. H. Halstead Jr., T. Ito (Eds.), Parallel Symbolic Computing: Languages, Systems, and Applications. Proceedings, 1992. X, 419 pages. 1993.

Vol. 749: P. A. Fritzson (Ed.), Automated and Algorithmic Debugging. Proceedings, 1993. VIII, 369 pages. 1993.

Vol. 750: J. L. Díaz-Herrera (Ed.), Software Engineering Education. Proceedings, 1994. XII, 601 pages. 1994.

Vol. 751: B. Jähne, Spatio-Temporal Image Processing. XII, 208 pages. 1993.

Vol. 752: T. W. Finin, C. K. Nicholas, Y. Yesha (Eds.), Information and Knowledge Management. Proceedings, 1992. VII, 142 pages. 1993.

Vol. 753: L. J. Bass, J. Gornostaev, C. Unger (Eds.), Human-Computer Interaction. Proceedings, 1993. X, 388 pages. 1993.

Vol. 754: H. D. Pfeiffer, T. E. Nagle (Eds.), Conceptual Structures: Theory and Implementation. Proceedings, 1992. IX, 327 pages. 1993. (Subseries LNAI).

Vol. 755: B. Möller, H. Partsch, S. Schuman (Eds.), Formal Program Development. Proceedings. VII, 371 pages. 1993.

Vol. 756: J. Pieprzyk, B. Sadeghiyan, Design of Hashing Algorithms. XV, 194 pages. 1993.

Vol. 757: U. Banerjee, D. Gelernter, A. Nicolau, D. Padua (Eds.), Languages and Compilers for Parallel Computing. Proceedings, 1992. X, 576 pages. 1993.

Vol. 758: M. Teillaud, Towards Dynamic Randomized Algorithms in Computational Geometry. IX, 157 pages. 1993.

Vol. 759: N. R. Adam, B. K. Bhargava (Eds.), Advanced Database Systems. XV, 451 pages. 1993.

Vol. 760: S. Ceri, K. Tanaka, S. Tsur (Eds.), Deductive and Object-Oriented Databases. Proceedings, 1993. XII, 488 pages. 1993.

Vol. 761: R. K. Shyamasundar (Ed.), Foundations of Software Technology and Theoretical Computer Science. Proceedings, 1993. XIV, 456 pages. 1993.

Vol. 762: K. W. Ng, P. Raghavan, N. V. Balasubramanian, F. Y. L. Chin (Eds.), Algorithms and Computation. Proceedings, 1993. XIII, 542 pages. 1993.

Vol. 765: T. Helleseth (Ed.), Advances in Cryptology – EUROCRYPT '93. Proceedings, 1993. X, 467 pages. 1994.