

NO-A195 853

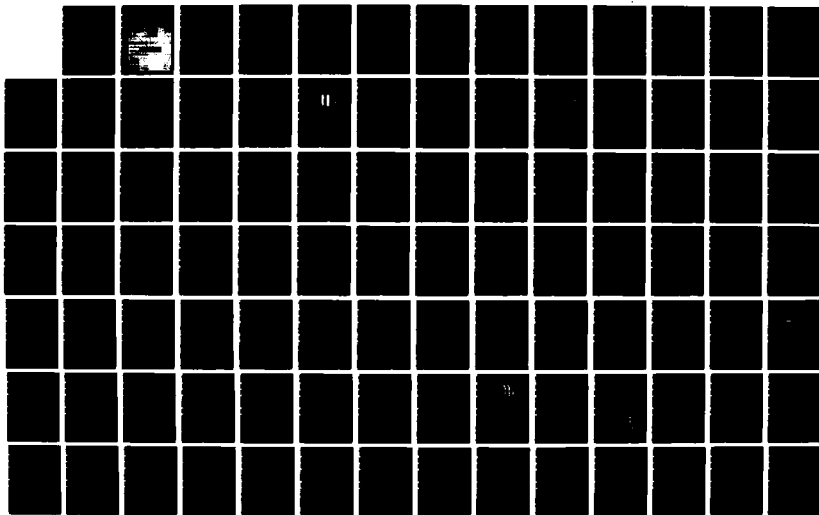
OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE
SEMANTICS VOLUME 4(U) MASSACHUSETTS INST OF TECH
CAMBRIDGE A GUPTA ET AL DEC 87 MIT-KBIISE-4
DTR557-85-C-00083

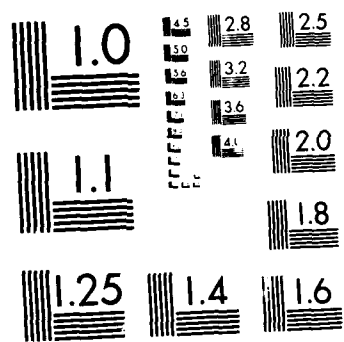
1/4

UNCLASSIFIED

F/G 12/7

NL





DTIC FILE COPY



Massachusetts
Institute of
Technology
Knowledge-Based
Integrated Information
Systems Engineering
(KBIISE) Project
Volume 4

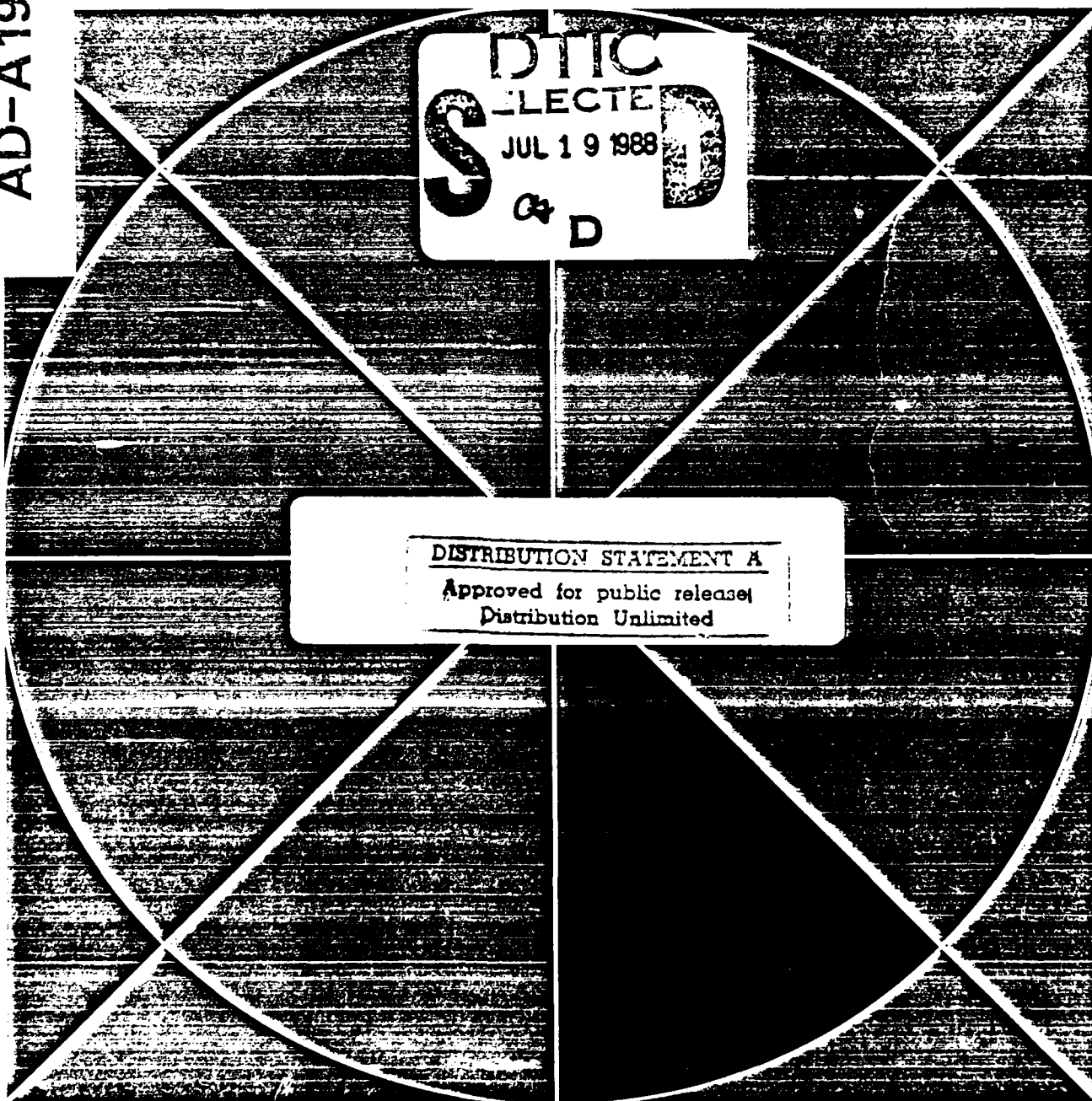
Object-Oriented
Approach to Integrating
Database Semantics

2

AD-A195 853

Amar Gupta
Stuart Madnick

SERIES EDITORS



- 10 7 18 006

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER KBIISE-4	2. GOVT ACCESSION NO. ADA 855 820	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Object-Oriented Approach to Integrating Database Semantics		5. TYPE OF REPORT & PERIOD COVERED Part of Final Report; Sept. 86 - Jan. 88
		6. PERFORMING ORG. REPORT NUMBER KBIISE-4
7. AUTHOR(s) Amar Gupta and Stuart Madnick (editors)		8. CONTRACT OR GRANT NUMBER(s) DTRS57-85-C-00083
9. PERFORMING ORGANIZATION NAME AND ADDRESS Massachusetts Institute of Technology Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Transportation Systems Center, Broadway, MA 02142 (In conjunction with U.S. Air Force)		12. REPORT DATE December 1987
		13. NUMBER OF PAGES 296 pages
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Transportation Systems Center, Broadway, MA 02142		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for Public Release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES This volume is one of eight volumes prepared by M.I.T. for Department of Transportation and Department of Defense (U.S. Air Force).		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Integration, knowledge, databases, systems engineering, methodologies, information modeling, heterogeneous database systems		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This volume presents the idea of using an object-oriented rule-based approach to integrating database semantics. It is divided into three parts. The first part, "An Actor's Role in Integrating Expert and Database Systems," presents arguments for various approaches for using expert system techniques in combination with database management systems and concludes by recommending an object-oriented approach.		

The second part, "Interfacing Objects and Databases," describes the motivation, development, and usage of a prototype Knowledge Oriented Representation Language (KOREL) that uses an object-oriented approach for interfacing to existing database systems. KOREL is implemented in common LISP. One of the important features of KOREL is that it automatically retrieves data as needed from the underlying databases rather than "snapshot" the entire database. This improves performance and minimizes the impact of skew. A sample application, based upon information which might exist at a university, is used to demonstrate the ability to superimpose extensional information upon database.

The third part, "A Knowledge-Based System for Resolving Semantic Conflicts: A Problem of Integrating Heterogeneous Database Management Systems," presents a comprehensive example of the integration of multiple databases. The application analyzed involves multiple tour guide databases (i.e., Michelin, Fieldings, Official Hotel and Restaurant Guide) which have overlapping information that is often ambiguous, contradictory, and incomplete. Various types of semantic conflicts are identified and approaches to their resolution are presented. A prototype system, based on the Knowledge Engineering Environment (KEE) software, is described.

OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE SEMANTICS

Amar Gupta
Stuart Madnick
Series Editors

Knowledge-Based Integrated Information Systems
Engineering (KBIISE) Report: Volume 4

Massachusetts Institute of Technology



Accession For	
NTIS	CRA&I ☒
DTIC	TAB ☐
Unannounced	☐
Justification	
By	
Distribution	
Availability Codes	
Dist	Avail and/or Special
A-1	

Copyright © 1987 by
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
77 Massachusetts Avenue
Cambridge, MA 02139
All rights reserved.

This book is one of eight volumes published by M.I.T. as part of the *Knowledge-Based Integrated Information Systems Engineering (KBIISE) Report*. The contents of these eight volumes reflect the work performed by M.I.T. employees and students, as well as by a number of contractors and subcontractors. All requests for permission for photocopying and distribution of these volumes and individuals parts should be directed to the series' editors at M.I.T.

OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE SEMANTICS

About This Volume

This volume presents the idea of using an object-oriented rule-based approach to integrating database semantics. It is divided into three parts. The first part, "*An Actor's Role in Integrating Expert and Database Systems*," presents arguments for various approaches for using expert system techniques in combination with database management systems and concludes by recommending an object-oriented approach.

The second part, "*Interfacing Objects and Databases*," describes the motivation, development, and usage of a prototype Knowledge Oriented Representation Language (KOREL) that uses an object-oriented approach for interfacing to existing database systems. KOREL is implemented in common LISP. One of the important features of KOREL is that it automatically retrieves data as needed from the underlying databases rather than "snapshot" the entire database. This improves performance and minimizes the impact of skew. A sample application, based upon information which might exist at a university, is used to demonstrate the ability to superimpose extensional information upon the database.

The third part, "*A Knowledge-Based System for Resolving Semantic Conflicts: A Problem of Integrating Heterogeneous Database Management Systems*," presents a comprehensive example of the integration of multiple databases. The application analyzed involves multiple tour guide databases (i.e., Michelin, Fieldings, Official Hotel and Restaurant Guide) which have overlapping information that is often ambiguous, contradictory, and incomplete. Various types of semantic conflicts are identified and approaches to their resolution are presented. A prototype system, based on the Knowledge Engineering Environment (KEE) software, is described.

Table of Contents

	Page
SERIES EDITORS' NOTE	1
AN ACTOR'S ROLE IN INTEGRATING EXPERT AND DATABASE SYSTEMS (Technical Report #20)	5
INTERFACING OBJECTS AND DATABASES (Technical Report #9)	27
KNOWLEDGE-BASED SYSTEM FOR RESOLVING SEMANTIC CONFLICTS: A PROBLEM OF INTEGRATING HETEROGENEOUS DATABASE MANAGEMENT SYSTEMS (Technical Report #13)	171

DEDICATED
TO
THE
NEXT
GENERATION
OF
PROFESSIONALS

SERIES EDITORS' NOTE

This book is one of eight volumes published by MIT as part of the Knowledge-Based Integrated Information Systems Engineering Project (KBIISE). In order to appreciate the papers in this book, it is necessary to be aware about the theme of the KBIISE project, its major objectives, and the different documents that summarize the research accomplishments to date.

Goal

The primary goal of the KBIISE project is to integrate islands of disparate information systems that characterize virtually all large organizations. The number and the size of these islands has grown over years and decades as organizations have invested in an increasing number of computer systems to support their growing reliance on computerized data. This has made the problem of integration more pronounced, complex, and challenging.

The need for multiple systems in large organizations is dictated by a combination of technical reasons (such as the desired level of processing power and the amount of storage space), organizational reasons (such as each department obtaining its own computer based on its function), and strategic reasons (such as the level of reliability, connectivity, and backup capabilities). Further, underlying trends in the information technology area have led to a situation where most organizations now depend on a portfolio of information processing machines, ranging from mainframes to minicomputers and from general purpose workstations to sophisticated CAD/CAM systems, to support their computational requirements. The tremendous diversity and the large size of the different systems make it difficult to integrate these systems.

Key Participants

The above problem is becoming increasingly evident in all large government agencies and in large development programs. In the fall of 1986, the U.S. Air Force (USAF) and the Transportation Systems Center (TSC) of the U.S. Department of Transportation approached M.I.T. to conduct and to coordinate research activity in this area in order "to develop the framework for a comprehensive methodology for large scale distributed, heterogeneous information systems which will provide: (i) the necessary structure and standards for an evolving top down global framework; (ii) simultaneous bottom up systems development; and (iii) migratory paths for existing systems."

Both USAF and TSC provided sustained assistance to members of our research team. In addition, Citibank and IBM provided some funds for research in very specific areas. One advantage of our corporate links was the opportunity to analyze and to generate case studies of actual decentralized organizational environments.

The research sponsors and MIT agreed that in order to deal with the heterogeneity issue in a meaningful way, it was important that a critical mass of influential individuals participate in the development of solutions. Only through widespread discussion and acceptance of a proposed strategy would it become feasible to deal with the major problems. For these reasons, a Technical Advisory Panel (TAP) was constituted. Nominees to the TAP included experts from academic and research organizations, government agencies, computer companies, and other corporations. In addition, several subcontractors, the primary one being Texas A&M University, provided assistance in specific areas.

Technical Outputs

The scope of the work included (i) technical issues; (ii) organizational issues; and (iii) strategic issues. On the basis of exploratory research efforts in all these areas, 24 technical reports were prepared. Eighteen of these reports were generated by MIT research personnel, and their respective areas of investigation are summarized in the figure on the opposite page.

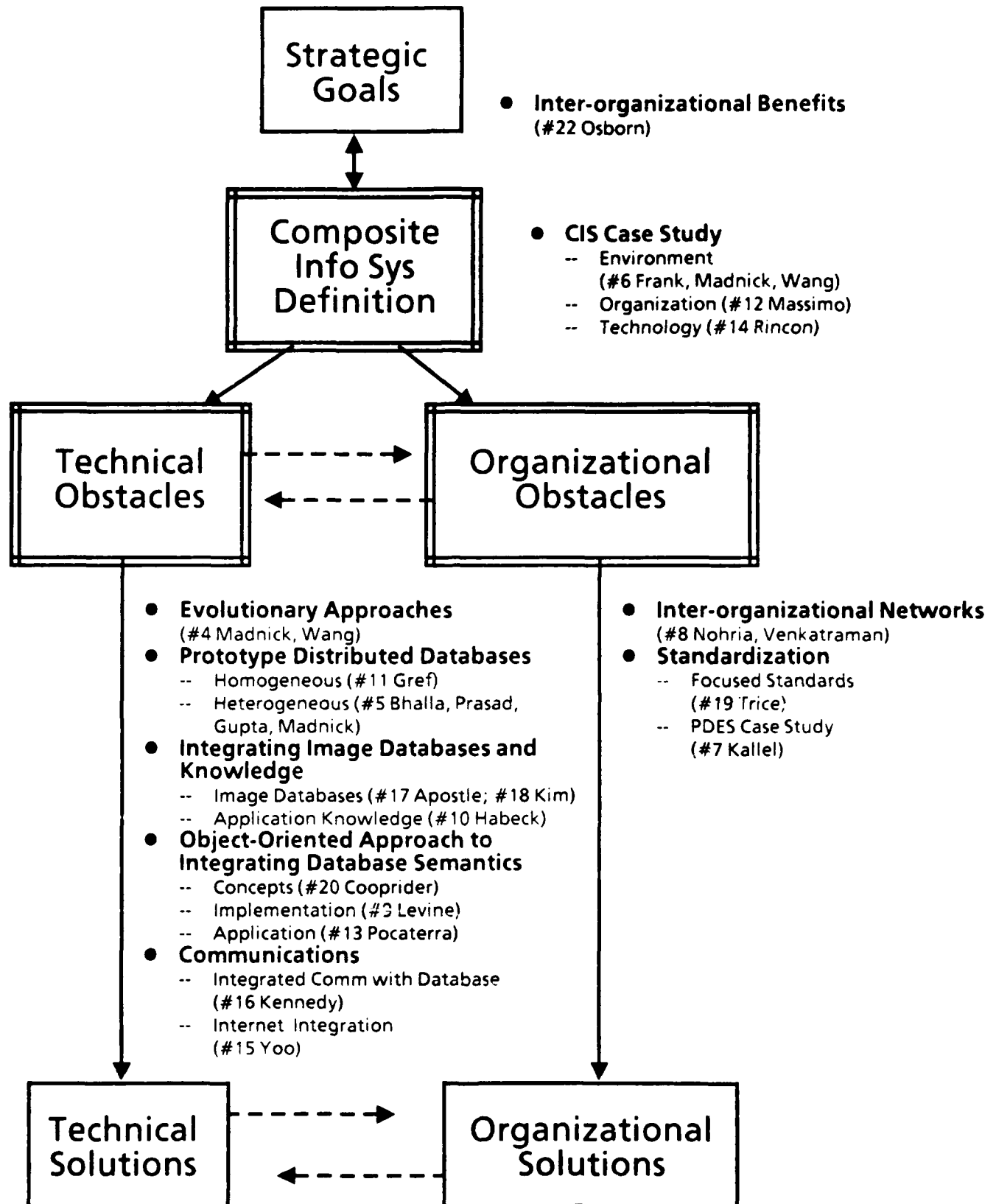
The five technical reports, not represented in the figure, are as follows:

- #1. Summary.
- #2. Record of discussions held at the first meeting of the Technical Advisory Panel (TAP) on February 17, 1987.
- #3. Consolidated report submitted by Texas A&M University.
- #21. Annotated Bibliography.
- #23. Record of discussions held at the second meeting of the Technical Advisory Panel (TAP) on May 21 and 22, 1987.
- #24. Contributions received from members of the TAP highlighting their views on various aspects of the problem.

All the 24 technical reports have been edited and reorganized as an eight-volume set. The titles of the different volumes are as under:

- 1. KNOWLEDGE-BASED INTEGRATED INFORMATION SYSTEMS ENGINEERING-HIGHLIGHTS AND BIBLIOGRAPHY
- 2. KNOWLEDGE-BASED INTEGRATED INFORMATION SYSTEMS DEVELOPMENT METHODOLOGIES PLAN
- 3. INTEGRATING DISTRIBUTED HOMOGENEOUS AND HETEROGENEOUS DATABASES - PROTOTYPES
- 4. OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE SEMANTICS
- 5. INTEGRATING IMAGES, APPLICATIONS, AND COMMUNICATIONS NETWORKS
- 6. STRATEGIC, ORGANIZATIONAL, AND STANDARDIZATION ASPECTS OF INTEGRATED INFORMATION SYSTEMS
- 7. INTEGRATING INFORMATION SYSTEMS IN A MAJOR DECENTRALIZED INTERNATIONAL ORGANIZATION
- 8. TECHNICAL OPINIONS REGARDING KNOWLEDGE-BASED INTEGRATED INFORMATION SYSTEMS ENGINEERING

Volume 2 contains the report submitted by Texas A&M and Volume 8 highlights the views of members of the TAP. Activities described in the other 6 volumes have been conducted at MIT.



Acknowledgments

Funds for this project have been provided by U.S. Air Force, U.S. Department of Transportation (Contract Number DTRS57-85-C-00083), IBM, and Citibank. We thank all these organizations and their representatives for their support. In particular, we are indebted to Major Paul Condit of U.S. Air Force for his initiative in sponsoring this project, to Dr. Frank Hassler, Bud Giangrande, and Bob Berk of the Transportation Systems Center (TSC) for their support and assistance, to Professor Joseph Sussman, Director, Center for Transportation Studies (CTS) at MIT for his help and encouragement, and to all the individuals whose results have been published in this book.

We would welcome receiving feedback from readers of this book.

Amar Gupta and S.E. Madnick
Massachusetts Institute of Technology
Cambridge, Massachusetts.

AN ACTOR'S ROLE IN INTEGRATING EXPERT AND DATABASE SYSTEMS

JAY COOPRIDER

The two disciplines of expert systems (ES) and database (DBMS) have so far evolved largely independently of each other. Currently, expert systems offer more powerful knowledge representation schemes such as production rules, first order logic, semantic networks, and frames that their database counterparts. Conversely, database management systems are characterized by higher levels of sophistication in areas such as concurrency control, data security, and optimized data access. All these strengths can be integrated together in Expert Database Systems (EDBS).

Alternative approaches to designing EDBS are as follows:

- (a) By adding inference/deduction capabilities to a DBMS. This can potentially increase the overall functionality and efficiency;
- (b) By adding limited or full DBMS capabilities to an ES inference engine. Efforts are still at an infant stage in this area; or
- (c) By implementing an ES-DBMS interface.

The level of coupling can be very loose, very tight, or somewhere in between. The third alternative deserves elaboration. A loosely coupled system can lead to very inefficient processing. Tight coupling, on the other hand, requires the ES to know exactly when and how to consult the DBMS, and how to interpret the responses. The best option appears to be the use of a "metalevel component" between ES and DBMS; in such a system, the processing is distributed, but control resides with the independent subsystem.

The above type of system involves use of concepts to linking elements. An actor (or object) is a computational entity that combines in a single unit both program and data. Actors include procedures, functions, and all kinds of data structures. They consist of an interface part (which is public) and an implementation part (which is private). Computation is performed only by sending messages.

Examples of actor-oriented (or object-oriented) approaches are PRISM, POSTGRES, and NIAM. NIAM has been designed with the basic assumption that common data is shared between many users of a single system. This necessitates some common understanding of the information represented by the data. This understanding places a need for a common grammar. The concept of NIAM is further guided by the rule that relevant general static and dynamic aspects should be described in the conceptual schema.

A candidate architecture for an ES-DBMS integration is explored. This uses an object-oriented intermediate language and an object-oriented representation for dictionary/directory (DD/D) information. In addition, there is a query optimization module. The candidate architecture allows existing databases to function normally, while requiring external users to access information via a standard expert system interface.

1. Introduction

The fields of Expert Systems (ES) and Database Management Systems (DBMS) have their own well-defined architectures and concepts. Each has developed on its own road -- and each has developed its own strengths and weaknesses. At the present time they have reached points where each can be helped by the other. In terms of data representations and power/efficiency trade-offs, each field has positive aspects to offer the other. Additionally, each has begun to realize problems that can be addressed by the other. Because of this, much attention has recently been given to looking for ways of merging the strongest parts of each area -- creating a new type of system to address the information system needs of the future.

In [2], Smith defines such a merged system -- referred to as an Expert Database System (EDBS) -- as "a system for developing applications requiring knowledge-directed processing of shared information." In such a system, the motivations for combining the technologies are important, and the way they are combined can impact the system greatly.

Although a formal theory of expert systems is yet to be developed [9], there are some key common features of ES technology that can impact this issue. It is probably reasonable to say that, in general, artificial intelligence knowledge representation schemes (production rules, first-order logic, semantic networks, frames, etc.) are more powerful than their database counterparts -- particularly the three popular data models: hierarchical, network, and relational. The primary example of this is that the ES representations embed inferencing capabilities. However, the systems that have been developed for the manipulation of basic knowledge representation objects lack most of the secondary storage management facilities commonly offered in database management systems (concurrency control, data security, and optimized data access). A method of combining the power and flexibility of ES/knowledge representations and the efficiency and control of DBMS systems would be very useful.

2. Some Problems

From an expert system point of view, [9] gives three issues concerning data access:

1. Data Volume: The size of the knowledge base.
2. Database Origin: Is the data for the expert system taken from an internal knowledge base or from a link to an external DBMS?
3. Determinism in Data Access Requirements: If only a portion of the relevant database can reside in main storage at any one time, how well-determined are the actual data requirements?

The way these factors interact can greatly influence the type of ES-DBMS interface that one might desire. From a database point of view [5], ES inference and recursive capabilities are of great potential value. However, [2] and [5] emphasize that a poorly combined system can result

in a multiplicative explosion in processing time. The typical inference method of examining one tuple/rule at a time can lead to a very inefficient overall system, and any loosely-coupled system (see below) has the possible problems of inconsistency, incompatibility and redundancy to face.

3. Some Approaches

In designing an EDBS, there are a number of possible approaches. It is possible to add inference/deduction capabilities to a DBMS to create an intelligent database system, improving functionality (and possibly the efficiency). Such an approach would seem to be favored by [5]. Another alternative would be to add file-handling or full DBMS capabilities to an ES inference engine [14]. Efforts in this area are still very early in development, though they may have great benefits in the long run.

Particularly in a Composite Information System (CIS) context, the most promising method of ES-DBMS interface would be through inter-system communication. This allows the two to co-exist as independent systems and provides some form of communication between them -- allowing the DBMS to possibly operate as a totally separate system with its own set of DB users. One method of organization for such an ES-DBMS system would be loosely coupled (from a control standpoint -- not a shared memory standpoint), with a "snapshot" of the required data from the DBMS being extracted when the ES begins to work on a set of related problems. This portion of the database is stored as part of the ES's own database. In such an organization, the non-determinism of data access requirements can lead to the very inefficient processing described in [2] and [5]. For example, if the ES sends the DBMS a large number of isolated information requests without the overall goal of these

requests having been determined, the system has no choice but to respond to them individually -- forfeiting all opportunities for global optimization and creating the possibility of a multiplicative explosion in processing time.

A more interesting way to organize an ES-DBMS system might be to tightly couple an ES with an external database. In this alternative, the ES plays the role of an "intelligent" user of the DB. A tightly-coupled scenario requires an online communication system between the ES and DBMS (as did the loosely-coupled scenario). Queries can be generated and transmitted to the DBMS dynamically, and answers can be received and transformed into the internal knowledge representation. Therefore, in tight coupling, the ES must know when and how to consult the DBMS, and must be able to understand the answers -- the external database becomes an "extension" of the ES's knowledge base [8]. As Table 1 illustrates, this system organization is very applicable to the typical CIS environment. A Composite Information Systems environment often involves (perhaps many) pre-existing large databases, and efficiency (and therefore determinism) is usually a major issue. Such an environment favors a tight-coupled approach to address the inherent problems of performance and flexibility required that they must face.

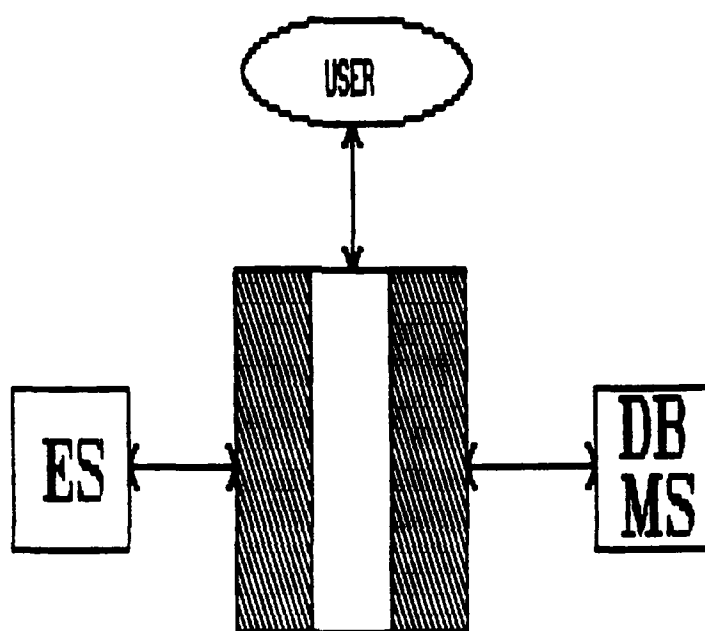
To attain tight coupling, particular care has to be taken to minimize the use of the communication system, and to the data representation and language translation problems. To address these problems, [11] recommends the use of a "metalevel" component between the ES and DBMS (see Figure 1). In such a system, the system processing is distributed, but control resides with an independent subsystem (as opposed to having control reside in either the ES or the DBMS alone). That is, the ES and the DBMS systems can each function (or process) independently of each other, but the

RULES					
CONDITIONS	1	2	3	4	5
Is there a very large data volume?	N	N	Y	Y	(Y)
Is there an existing database?	-	-	N	Y	(Y)
Are the data access requirements determined during the session?	N	Y	-	N	(Y)
ACTIONS					
Strategy 1: Elementary Data Access	X	X			
Strategy 2: Extension of ES to Generalized DBMS		X	X		
Strategy 3: Loose Coupling with External DBMS	X			X	
Strategy 4: Tight Coupling with External DBMS			X		(X)

CIS CHARACTERISTICS
 CIS STRATEGY

DECISION TABLE FOR THE DETERMINATION
OF AN ES ARCHITECTURE

TABLE 1 - SOURCE: [9]



"Meta-level Component"

Figure 1.  - Interface Routine

communication and data translation between the two is controlled by a third independent system. This is the general approach that this paper will take in building its candidate architecture.

4. Actor or Object-Oriented Systems

Typical data processing applications are implemented to model some real-world system [7]. Classic accounting applications model an account's paper ledgers and journals. Electronic mail systems model the paper-based U.S. Postal Service. One of the measures of a data processing tool's goodness is how easily the tool can be made to model the real world. The distance between the tool and the world is frequently referred to as the "semantic gap". The smaller the semantic gap, the easier the tool is to use.

During the last ten years, there has been much research on a new kind of computational entity, variously known as "actors", "objects", or "abstract data types". An actor (or object) is a computational entity that combines in a single unit both program and data [10]. Actors therefore include procedures, functions, and all kinds of data structures. They consist of an interface part (which is public) and an implementation part (which is private) [3]. Computation is performed only by sending messages. It is not possible to reach inside an actor or change it without sending that actor a message requesting such an operation. This guarantees the integrity of the objects of computation. Systems designed using actor or object-oriented (O-O) principles should depend only upon the behavior of the modules and not upon their physical representation.

Actors (objects) are organized into classes that contain the methods (sometimes called "scripts") that the objects use to respond to messages. Classes are organized in a strict

hierarchy, so that they can inherit the structure and methods of their superclasses.

The advantages of object-oriented systems result from their being based on simulation paradigms. As described in [6], a simulation paradigm can be summarized as follows:

- Computer objects correspond to external objects.
- The behavior of computer objects models the behavior of external objects.
- Objects are classified according to their behavior.

By following such simulation paradigms, object-oriented systems can have relatively small semantic gaps. In particular, they are well-suited to deal with time and changes of state in time, as well as for describing parallel or distributed processing [10] -- all important issues for CIS environments. There is often a close correspondence between thinking about computer objects (actors) and real-world objects. In many situations, the object-oriented system is almost directly derivable from the real-world situation it is intended to model [7].

5. Object-Oriented Approaches to KDBS

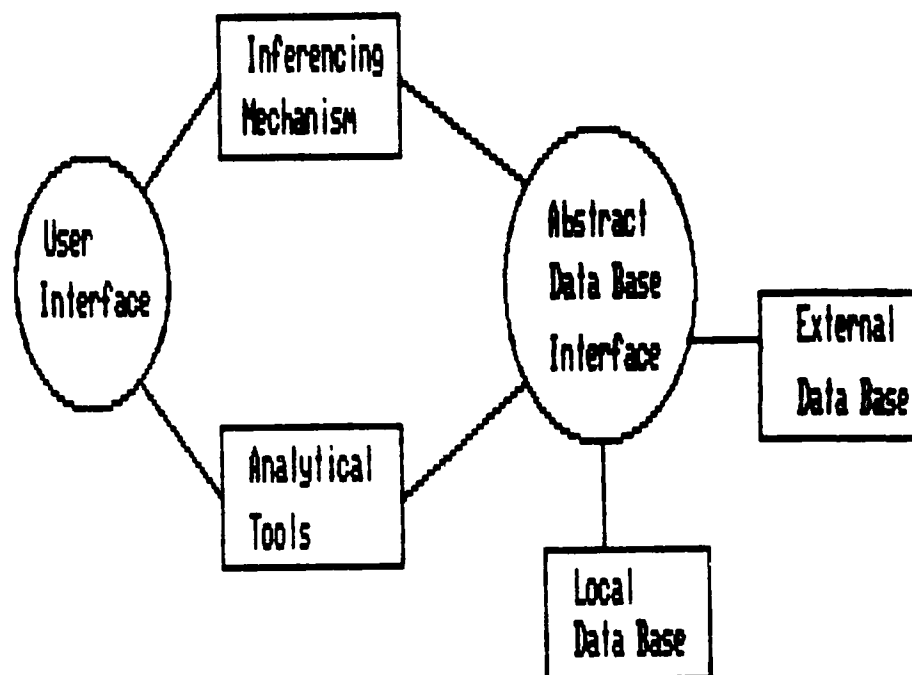
There have been a few object-oriented approaches to system design that might help provide insight to our problem. [3] and [4] describe PRISM, which uses an object-oriented dictionary facility to represent and describe a data dictionary schema. The entire specification of the system is stored in a knowledge base of "constraints". The concept is similar to that of semantic modelling. The primary problem with the PRISM design is that the data dictionary scheme controls processes within the expert system itself, but doesn't help external communication with other systems [11]. In other words, since it is primarily designed as an object-oriented approach to meld logic programming and database systems (but not to merge external ES and DBMS

systems), some of the critical issues to be faced in ES-DBMS integration in a CIS environment (communication control, language translation, query optimization, etc.) are not addressed.

POSTGRES [17] is similar in many respects to PROBE at CCA [18] and GENESIS at UT, Austin [19]. The philosophy behind it is that the many data models developed over the last decade exhibit widely varying concepts and unifying these constructs into a single data model will be impossible. Its goal is to use an O-O approach to simulate all the various constructs. However, it addresses none of the ES and corresponding communication issues.

An intelligent data base presented in [8] uses NIAL and the roster data model designed by Dr. Fleming Schmidt of IBM Denmark. It lacks an interface to an external expert system, but its structure (see figure 2) is promising to our problem. In particular, it addresses communication control in an external database interface, using predicates (methods, in O-O terms) in the data model to signal needed communication.

In the roster model, a data base is viewed as having two major parts: a head and a body. Conceptually, the body is a table that stores the data of the roster as the items (zones) of the table (very similar to a standard relational model). The head of a roster describes its structure, and has three required fields: "Labels", "Predicates", and "Keyflags". Of special note in our (object-oriented) context is the Predicate, which names an operation ("method") used to do a domain check on the corresponding zone. The roster model is more general than the relational model in that the unit of information stored as the items of the table can be any data structure. This generalization permits a roster to hold aggregated data in a convenient form for later analysis. It also allows data structures



Nial architecture for intelligent DSS.
Figure 2.

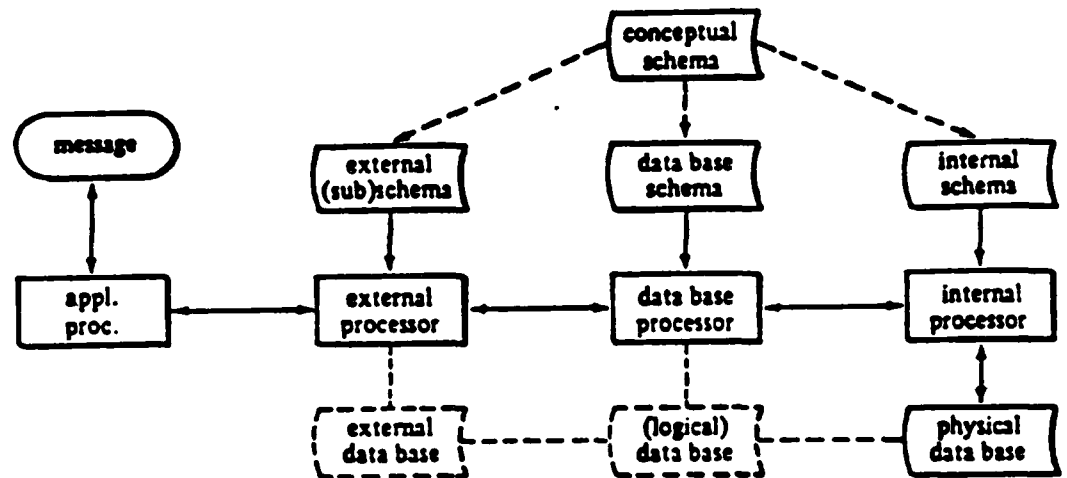
that are used to build components of a knowledge base to be accessed easily, making it particularly applicable as an example of an intermediate data representation for our model.

5.1 NIAM -- Object-Oriented Systems Design

NIAM is a systems design methodology described in [12] and [13]. The details of the methodology are not important here, but the general concepts are useful. For NIAM, the prime characteristic of the data base environment is that common data is shared between many users of a single system. By sharing common data, these users establish a dialogue with each other through the system. If this communication is to be useful and reliable, there must be some common understanding among the users of the information represented by the data. This common understanding must be recorded and in order to establish a dialogue a common predefined established grammar is needed. The things and happenings to which the common understanding of the represented information refers can be termed the "universe of discourse" [14].

NIAM considers that there are in fact two systems of interest: The universe of discourse and the information system which contains a linguistic representation of that universe of discourse. It uses a conceptual schema (see Figure 3) to describe which entities can possibly exist in the universe of discourse. For this paper, NIAM has two relevant primary principles:

THE 100 PERCENT PRINCIPLE: All relevant general static and dynamic aspects (all rules, laws, etc.) of the universe of discourse should be described in the conceptual scheme.



NIAM OVERVIEW

FIGURE 3

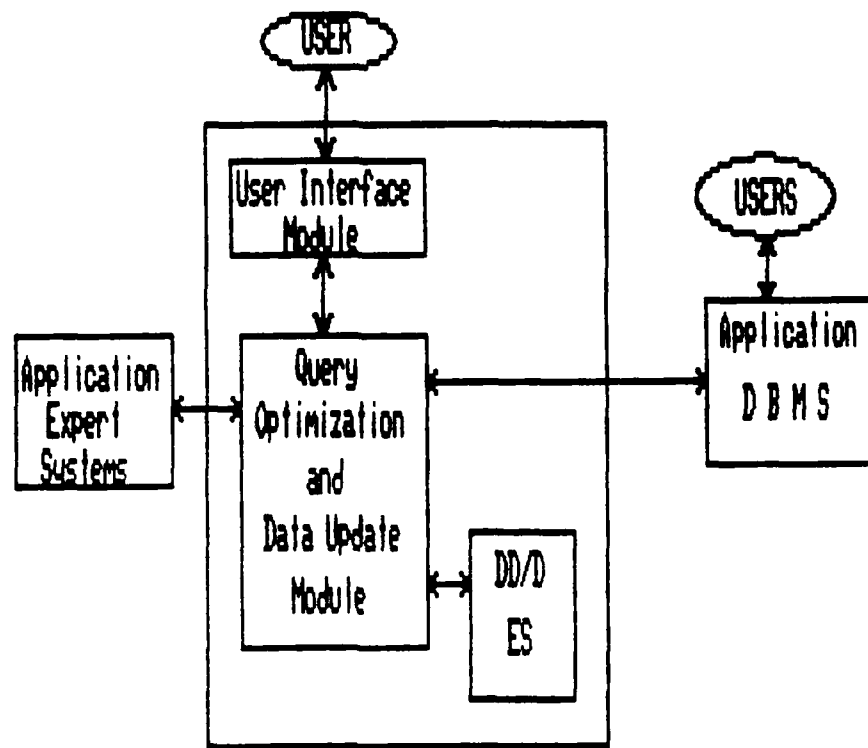
CONCEPTUALIZATION PRINCIPLE: A conceptual schema should only include conceptually relevant aspects, both static and dynamic, of the universe of discourse, thus excluding all aspects of (external or internal) data representation, physical data organization and access as well as all aspects of particular external user representation such as message formats, data structures, etc.

From an object-oriented point of view, these two principles have some profound implications. Basically, the 100 PERCENT PRINCIPLE says that anything that needs to be represented in an information system can be represented as an "object" in a conceptual schema. The CONCEPTUALIZATION PRINCIPLE says that only the interface details are important, not the implementation -- a perfect object-oriented concept. An object-oriented data system is directly derivable from a NIAM conceptual schema -- in fact, there is software available to translate the final NIAM conceptual grammar description and diagrams into system code.

6. A Candidate Architecture for ES - DBMS Integration

Building on many of the designs described above, Figure 4 shows a proposed architecture for the integration of expert systems and database management systems. Some description of the architecture is in order.

A query optimization module is included in a separate control subsystem. This optimization module can be divided into three parts [9]. First, a preprocessing mechanism collects ES requests for data while simulating the ES deduction process. As soon as a database related request is encountered in the expert system, it stops its reasoning process and gives control to a higher (metalanguage) program, which simulates the continuation of the database-related reasoning to collect similar database requests. In



Proposed EDBS Architecture

Figure 4.

effect, this mechanism delays the submission of individual tuple-oriented queries by converting them into more "optimizable" set-oriented ones, expressed in the (object-oriented) intermediate language. In the second step, the intermediate language expressions are optimized (semantic query simplification and common subexpression in recursive calls). The optimized query is then translated to the DBMS query language and executed by the DBMS. The answer is loaded into the internal ES database. After all this has been completed, the meta-level evaluation stops, and control returns to the expert system.

Rather than the manually-maintained database format that [11] uses for its data dictionary/directory (DD/D), the proposed model uses an object-oriented scheme (as in PRISM's object-oriented data dictionary suggested in [3]) controlled by an expert system. The DD/D ES is used to determine whether a user query will require use of the database, expert system, both or neither. This is done through the data directory services, which track which data is kept in the ES and which in the DBMS. It will also determine whether a proposed modification to the database or its schema (DD/D) is consistent, etc., using the object prototype and database sample methods outlined in [15], in which a sample of a new or changed data type is certified as meeting all semantic constraints before an actual operation is allowed. Semantic control of the entire system can be maintained by the system in this way. A full data model [16] would require schemata for:

1. Object Classification Schema -- Entities
2. Object Structure and Relationship Schema --
Relations
3. Operation Classification and Structure Schema --
Methods
4. Behaviour Schema -- Model over time

Such a model will involve a great deal of knowledge -- the internal DD/D ES will be needed to quickly find the right

parts of the models (meta-control) for processing and maintenance -- it would be an overwhelming manual task.

The format of the intermediate language will be O-O, similar to that presented in [8], with something similar to the roster data model. This enables a structure very easily (efficiently) translatable into a relational structure for SQL query translation, and yet it supports "predicates" (methods, in O-O terms) to support better semantic integrity of more involved structures (even ES frames).

Communication to the external database should probably be SQL queries, as the emerging standard in the industry. This would give the widest possible range of systems to interface to (again, an important factor for CIS). Note that while users can access the application database outside of this integrated system (particularly for those DB systems already installed), the ES interface is strictly through the interface system. This assumes that few pre-existing ES's will be involved, and it removes the need of providing a good user interface from the ES (the standard DBMS typically includes a good query interface as part of the package). It should also be relatively straightforward to have the system interface to application programs directly, in addition to responding to user requests.

7. Further Research

Many issues are yet to be resolved for systems similar to that described here. Note that an object-oriented approach to the interface has many benefits, but it is not without its costs. For a typical DBMS, the cost of executing different operations is usually known and can be incorporated into a cost model -- permitting the query optimizer to estimate the relative costs of various strategies. For an actor, the system is ignorant of the

details of physical representation and of the algorithms used to implement its operation. To enable the optimizer to construct efficient strategies using actors, it is important that the properties of the actors operations and its implementation parameters (like response time and volume of data returned) are specified [5].

At the present time, there is no coherent object-oriented data or knowledge model with a mathematical foundation (such as relational calculus). This can lead to several problems, such as difficulty in specifying queries and ensuring the database is consistent. Research into efficient storage structures for complex actors is in the very early stages. Use of actors from a conventional data-processing language such as COBOL is difficult because they lack the very concept of an actor. Particularly in a distributed heterogeneous environment, the system details and performance need to be tested carefully. Data modification is a particularly difficult issue, but [15] gives a good starting point for maintaining an EDBS in this type of environment.

Even with these potential problems, the ideas presented here would seem to have merit. The merging of expert system and database technologies is the natural result of the union of strong concepts from both fields. Added with an object-oriented interface, the result can be an efficient, powerful hybrid EDBS, able to handle the changing information system needs of business in the future.

References

- [1] Kerschberg, L. "Preface." In Kerschberg, L. (editor), Expert Database Systems, Benjamin/Cummings Publishing Co., 1986.
- [2] Smith, J. M. "Expert Database Systems: A Database Perspective." In Kerschberg, L. (editor), Expert Database Systems, Benjamin/Cummings Publishing Co., 1986.
- [3] Zaniolo, C., H. Ait-Kaci, D. Beech, S. Cammarata, L. Kerschberg, D. Maier, "Object Oriented Database Systems and Knowledge Systems." In Kerschberg, L. (editor), Expert Database Systems, Benjamin/Cummings Publishing Co., 1986.
- [4] Shepherd, A., L. Kerschberg. "Constraint Management in Expert Database Systems." In Kerschberg, L. (editor), Expert Database Systems, Benjamin/Cummings Publishing Co., 1986.
- [5] Dayal, H., H. Hwang, F. Manola, A. Rosenthal, J. Smith, "Knowledge-Oriented Database Management. Phase I Final Technical Report." CCA-84-02, Computer Corporation of America, 1984.
- [6] MacLennan, B., "A Simple Software Environment Based on Objects and Relations." in Proceedings of the ACM SIGPLAN 85 Symposium on Language Issues in Programming Environments, ACM, 1985.
- [7] Peterson, R., "Object-Oriented Data Base Design." AI Expert, March, 1987.
- [8] Jenkins, M., "Intelligent Data Bases & Nial." AI Expert, March, 1987.
- [9] Vassiliou, Y., J. Clifford, M. Jarke, "Database Access Requirements of Knowledge-Based Systems." In Kem, W., D. Reiner, D. Batory (editors), Query Processing in Database Systems, Springer-Verlag, 1985.
- [10] Kahn, K., "Intermission -- Actors in Prolog." in Tarnlund (editor), Logic Programming, 1983.
- [11] Al-Zobaidie, A., J. Grimson, "Expert Systems and Database Systems: How Can They Serve Each Other?" in Second International Expert Systems Conference, Learned Information Ltd., 1986.
- [12] Nijssen, G., "An Exercise in Conceptual Schema Design.", Working Paper IFIP WG 2.6 (Databases), 1976.

REFERENCES (Continued)

- [13] Maddison, R., Information System Methodologies, Wiley Heyden Ltd., 1983.
- [14] Jardine, D., "Semantic Agreement and the Communication of Knowledge." in Steel, T., and R. Meersman (editors), Database Semantics (DS-1), Elsevier Science Publishers, 1986.
- [15] Nguyen, C., "Object Prototypes and Database Samples for Expert Database Systems." in Kerschberg, L., Expert Database Systems, Proceedings of the First International Conference on Expert Database Systems, University of South Carolina, 1986.
- [16] Neuhold, E., "Objects and Abstract Data Types in Information Systems." in Steel, T., and R. Meersman (editors), Database Semantics (DS-1), Elsevier Science Publishers, 1986.
- [17] Stonebraker, M., "Object Management in POSTGRES Using Procedures." In 1986 International Workshop on Object-Oriented Database Systems, Pacific Grove, Calif., Sept. 1986.
- [18] Dayal, U., A. Buchmann, D. Goldhirsch, S. Heiler, F. Manola, J. Orenstein, and A. Rosenthal. "PROBE -- A Research Project in Knowledge Directed Database Management: Preliminary Analysis." Technical Report CCA-85-03, Computer Corporation of America, 1985.
- [19] Batory, D., J. Barnett, J. Garza, K. Smith, K. Tsukuda, B. Twitchell, and T. Wise. "GENESIS: A Reconfigurable Database Management System." Technical Report 86-07, Dept. of Computer Science, Univ. of Texas at Austin, 1986.

INTERFACING OBJECTS AND DATABASES

SAMUEL P. LEVINE

There are many large databases in existence. Unfortunately, serious problems arise when multiple databases need to be used in combination, especially by naive users.

In this research effort the knowledge representation of objects is explored as a more intuitive, yet powerful, way to interface to multiple databases. Furthermore, using objects, it is possible to superimpose upon the databases extensional information, such as heuristic knowledge, procedures, and constraints. This interface allows an object to reference information already captured in existing databases, as well as extend that data using the capabilities of an object-oriented representation.

As part of this effort an object-oriented language named KOREL (Knowledge Oriented Representation Language) and an easy-to-use database management system named ADBMS (Abstract DBMS) that utilizes KOREL have been implemented. A sample application has been demonstrated using KOREL and ADBMS to access information from databases which might exist at a university.

This system allows information to be retrieved and combined from multiple databases by either:

1. concatenation - similar information about same class of object from multiple databases (e.g., merge students from two departments).
2. relational join - different information about same class of object from multiple databases (e.g., merge list of students from department database with grade for each such student from registrar's database).

In addition, to providing convenient simple retrieval (from a single database) and complex retrieval (combinations from multiple databases), this system provides the additional capabilities of:

1. data mapping - to automatically convert data to reconcile differences in units (e.g., faculty salary is monthly, staff salary is bi-weekly).
2. procedural - to solve the least fixed point problem (e.g., "who is the lowest common manager of Stu and Amar?")
3. heuristic - to resolve contradictions in the data and/or incomplete information (e.g., "who are all the students eligible to graduate?")
4. meta-level - to explain how data is obtained or derived (e.g., "how did you conclude that Sam cannot graduate?")

One of the important features of KOREL is that it automatically retrieves data as needed from the underlying databases. This means that it is not necessary to "snapshot" the entire database thereby improving performance and minimizing the impact of skew.

1 Overview

1.1 Abstract

Currently, many large corporate and public databases exist and are in wide use. Unfortunately, many problems arise when the data in these databases needs to be used by another application or by a user, including:

- Current data models are limited in what information they can represent about real-world objects.
- Database query languages are inadequate to perform many potentially desirable operation on retrieved data.
- The way objects and relationships are represented in a database is non-intuitive for naive users.
- It is difficult to access and combine information in multiple databases.

In this report, the Artificial Intelligence knowledge representation methodology of *objects* is proposed as a more intuitive model of items in the real world. By providing an interface between objects and existing databases, the limitations discussed above can be removed. Using objects, it is possible to store not only data, but also extensional information, including heuristic knowledge, procedures, and constraints. An interface between objects and existing databases would allow an object to reference the information already captured in a database, as well as extend that data using the capabilities of an object-oriented representation. The interface which is proposed also allows the combination of information retrieved from multiple databases.

The implementation of an object-oriented language called KOREL is discussed, along with a set of tools to aid in the definition of an interface between KOREL objects and existing databases. KOREL runs in Gold Hill Common Lisp on an IBM PC/AT. An example in which KOREL objects are used to access information from databases which might exist at a university is also provided.

1.2 Relational DBMS Limitations

Although databases are widely used for storing and sharing information, the expressive power of the underlying data model is limited. In discussing this point further, I will look at the relational data model as an example of a powerful and extensively used database technology.

There are several types of limitations which arise when using data which is stored in a database:

- **Limitations of the database philosophy:** Often, commonly desired operations are inappropriate for the typical functionality of a query language:

We also feel that the role of a query language should be primarily the selection of data from a database, rather than arithmetic computation on the data. The computational capability, if desired, should be separate from the retrieval capability.

[AHO79]

As a result of this viewpoint, current Relational Database Managements Systems (RDBMS) tend to support queries which are only relations-to-relation mappings on the contents of the database. Thus, procedural capabilities fall outside the role of most query languages, although mechanisms

like embedding the query language in a procedural language are sometimes provided.

- **Limitations of the data model:** Another measure of the *completeness* of a relational database language arises by comparison to Codd's relational algebra and calculus. However, even languages which are complete in this respect cannot compute the family of "least fixed point" operations, as these cannot be expressed in relational algebra or calculus. These problems arise when trying to find the lowest common node for a set of objects arranged in a hierarchy. For example, it has been proven [AHO79] that it is impossible to determine the lowest-level manager for a group of employees by using relational algebra or calculus.
- **Limitations of the data description:** Finally, the actual presentation of data is often seen as non-intuitive.

If relational databases are to be accessed by people who have no experience with computer methods, then it is essential that the data description at the user interface have a clear correspondence with the subject matter in the real world. The fact that records are convenient and economical units of information for data storage and access is of little concern to such inexperienced users. A major problem with the table view of relations is that it is not very explicit about the nature of the real world information represented in a relation [PIR79]

1.3 Retrieving Data from Multiple Databases

In the real world, situations often arise when data must be combined from multiple sources. There are two main ways that this information can be combined:

- **Concatenation:** When information about the same conceptual object is contained in multiple databases, then it is often desirable to put together data from several sources. For example, if two corporations merge, it may be desirable to ask questions like "Throughout the new company, how many widgets do we have?" where information about widgets is kept in separate databases by each of the old companies.
- **Relational Joining:** At other times, it is useful to relationally join information from several databases. This would be necessary when there are centralized or public databases, and localized or private ones, each of which contains only a part of the desired information. For example, there is a central database listing the positions of all railroad boxcars in the United States. A single railroad company might want to relationally join a list of their boxcars (retrieved from their private database) with the information stored in the national database in order to locate their boxcars.

In practice, it is difficult to combine information stored in multiple databases, as the data may be returned in different formats. A possible solution is to create a higher level application which knows how to access each database and combine the results, but this requires finding out enough about each DBMS to be able to hard-wire the interface.

1.4 Interfacing Objects and Databases

There are currently two main approaches for creating an interface between an object-oriented system and an existing database:

- **Snapshot technique:** One widely used technique is to take a "snapshot" of the current state of the database, and translate that into an object

representation. For example, to represent the stock market history of a company, a query might be executed to return a table of the data for that company. The results of that query would then be processed to create a set of objects. This technique is used by the KEEConnection from Intellicorp.

While this approach is straightforward, it has many significant disadvantages. Frequently, the snapshot is very large, so large that it may be extremely inefficient to represent it using objects. For example, a query which returned a million entries would become a million objects. Operations like searches, which are optimized in a DBMS, are more inefficient when performed by an object system.

A potentially more serious problem is *skew*. Skew occurs when other users of the central database change the data, making the data which is stored in the objects obsolete. To avoid making decisions based on old information, steps must be taken to ensure that the data presented to the users is current.

- **Hard-wired interface:** Using this approach, a developer will give each class of objects procedural knowledge about how to access the desired information from the database. Typically, this procedure contains a set of queries to be executed at the appropriate time, with knowledge about how to connect to the database, as well as how to reformat and interpret the returned table. For example, to connect a stock market database with an object using this technique, procedures would be written which would directly access the stock market database, execute a pre-defined query, and translate the results into an object-compatible form, like a LISP list structure.

This is an extremely cumbersome and time consuming process, as the developer must be concerned not only with the domain knowledge, but also with the database language to access it, and the details of connecting to the database.

In many situations such as the stock market application presented above, the developer doesn't have complete freedom over the format of the information stored in the database. This is because other users and applications might rely upon the existing structure of the database. This constraint prevents the developer from modifying the structure of the database to simplify access. Simply stated, this requires the new application to interact with the database just as any other user might.

1.5 KOREL: Objects Interfaced with Databases

KOREL is an object-oriented language along with a set of tools to connect KOREL objects to databases. The way KOREL and these tools interact is pictured in Figure 1 and described here:

- **KOREL** is an object-oriented representation language, with extensions to simplify constraint and knowledge representation, as well as well-defined mechanisms for interfacing with databases. Mechanisms are also provided to build, relate, and display objects. KOREL is discussed in chapter 2.
- **ADBMS** is an Abstract DBMS. The basic query for the ADBMS is called a multi-query. When passed a multi-query, the ADBMS breaks it down into individual queries, each of which is passed to the corresponding *actual DBMS*. Retrieved values are returned to the ADBMS in a common format and combined appropriately. The ADBMS is discussed in chapter 3.

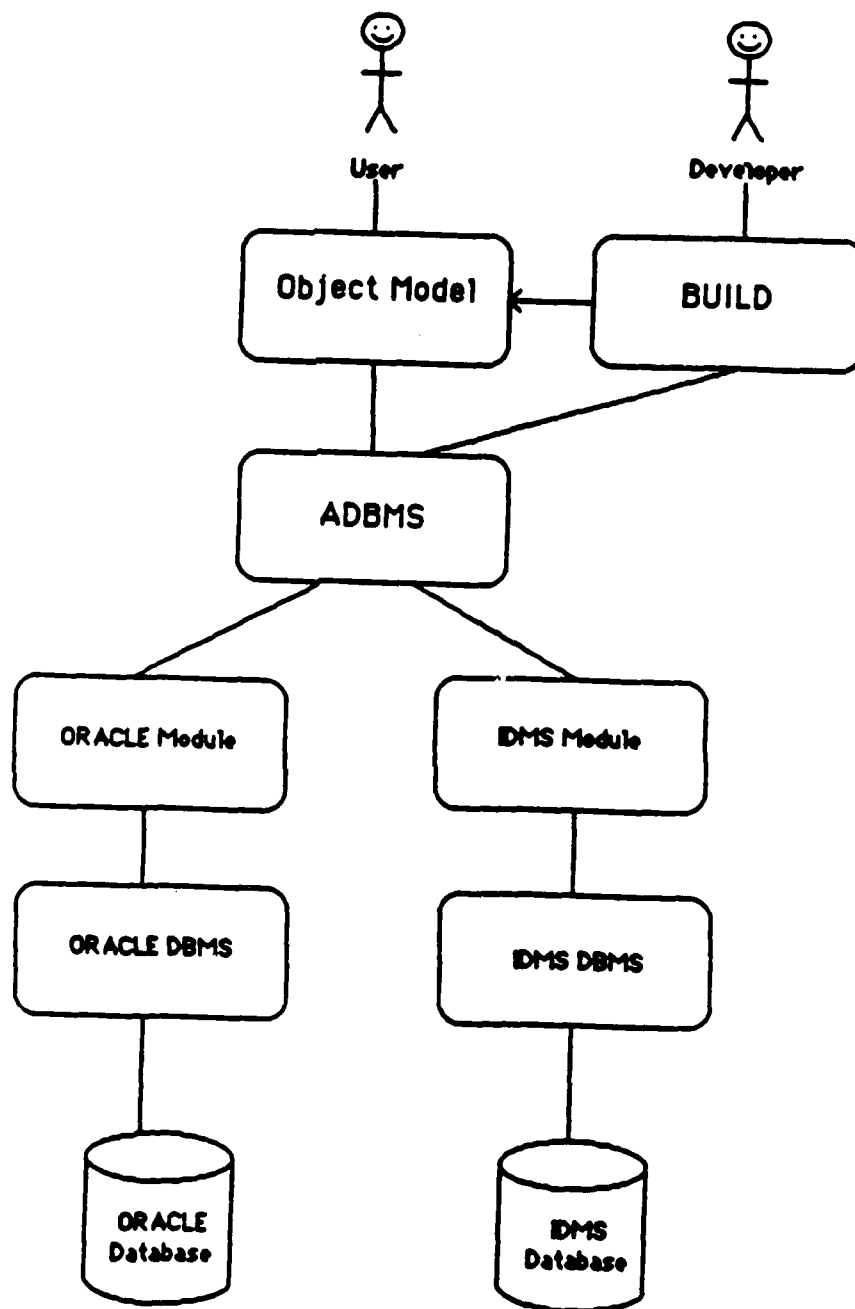


Figure 1: Major Components of the Object-Database Interface

- **Actual DBMS Modules** interface the ADBMS with a commercial DBMS. Specifically, these modules process single-query messages to generate an actual DBMS query. After a connection to a database is established, this actual query is executed, and its results are transformed into a common format and passed back to the ADBMS. The details of what a DBMS module must do are contained in chapter 4.
- **BUILD** is a tool which can be used by an application builder to define multi-queries and single-queries and attach them to KOREL objects. BUILD interacts with the specific DBMS modules to get information about the existing databases, which simplifies the definition of single-queries. Chapter 5 describes the functionality and use of BUILD.

These tools are used in two distinct ways:

- *Definition phase:* The tools are used to create an object model and assist the system builder in defining queries.
- *Use phase:* Next, the tools are used at run time, to process the previously defined queries.

Chapters 6 and 7 discuss these phases.

Chapter 8 presents an extended example of the use of these tools on the databases which might exist at a university.

Finally, Chapter 9 presents topics for further research.

1.6 The Advantages of KOREL

- The interface to the DBMS is clearly defined. This means that once an application (like KOREL) is integrated with the ADBMS, then that ap-

plication can use any of the physical DBMSs for which an actual DBMS module is defined.

- The object model is implemented by using objects. Since objects can be created and manipulated independently of an application, the data model can be used by more than one application, or as a higher level database interface.
- The ADBMS queries are defined using the existing structure of the database. This means that current databases can be easily accessed via an object model.
- The query accepted by the ADBMS can be made of multiple sub-queries, joined by either concatenating the result of each sub-query or performing a relational join. This allows an object to have information which is stored in multiple tables, or even in multiple databases.

1.7 Related Research Efforts

There have been previous efforts in extending the representational capabilities of databases by interfacing them with expert systems. Other research has specifically addressed increasing the storage efficiency of object-oriented languages. Finally, providing a common interface to multiple databases has also been an issue for research. Each of these areas is discussed in more detail below.

1.7.1 Expert Systems and Databases

Much research has been done in the area of combining expert systems and databases, both to increase the representational capabilities of databases, and

to improve the storage and retrieval efficiencies of expert systems. Researchers have concentrated on the following approaches:

- **Extended DBMS:** One possibility is to start with a DBMS and add deductive capabilities, typically by adding inferencing to a database([STO83], [WON84], [STO84]). Problems arise because it becomes necessary to relax many of the constraints on the way data is stored in order to gain representational power. Yet it is because of these very constraints that DBMSs can be so efficient. Hence, this efficiency is lost.
- **Extended Expert System Shell:** The problem can also be approached from the other direction, by improving the efficiency of an existing expert system shell, typically PROLOG([BER85], [WAL83]). The major drawback here is that in order to gain efficiency, the representation of knowledge must be so constrained that it is often difficult to express extensional information.
- **KBMS:** Others are striving to come up with a general, integrated representation which can store both knowledge and data efficiently([WEI84], [SU85]). Again, explicit choices must be made between representational power and efficiency. In situations which are not constrained by existing databases or expert systems, this is the most promising strategy.
- **Translator:** From this perspective, expert systems and database systems are viewed as separate existing entities, and the research emphasis is on building a translator between them ([KEL82], [LAF83]). A typical translating system might store rules using an expert system methodology which operate on data stored in the database, providing efficient storage of data and a good representation for knowledge. The major disadvantage of this approach is the high cost of overhead involved in translating

between the two systems. Also, there is a redundancy of functionality, as the inference engine and the DBMS often perform similar operations, like searches.

1.7.2 Object-Oriented Database Systems

The major goal of these is to add persistence and sharability to objects by using DBMS techniques. A good example of such a system is the ODBS being developed by MCC [BAN86].

In an Object Database approach, an object system is given, and the goal is to permanently store those objects in a DBMS-like manner. This approach is most appropriate when there is an existing set of objects, but no existing database, as it requires creating a database to represent the objects. Also, an Object Database stores all the information about each object in the database, typically requiring an extension of the underlying data model.

This report, on the other hand, starts with a database and creates objects to represent it. Some information is stored in the database while other information (typically procedural or heuristic) is stored in the objects.

2 KOREL: a Knowledge Object REpresentation Language

2.1 An Overview of Objects

Objects are a natural way to think about the world. An object is a way to define a class of things by specifying its values and procedures (ie: what it is and what you can do with it). Checking accounts, for example, have a balance. Appropriate things to do with a checking account include writing checks, depositing cash, and balance inquiries.

There are two types of objects: *classes* and *instances*. *Classes* represent those features and procedures that a set of objects share. For example, all checking-accounts have a balance and procedures to deposit, withdraw, and check the balance. *Instances* are specific examples of a class, with values. For example, SAM-CHECKING-ACCOUNT has a BALANCE of \$14.01.

Figure 2 shows an example checking account class and instance. Each object is identified by a unique name. Those pictured are named *CHECKING-ACCOUNT* and *SAM-CHECKING-ACCOUNT*. Each object has a set of values, known as *slots* or *attributes*. The attributes of SAM-CHECKING-ACCOUNT, for example, are BALANCE and INSTANCE-OF. Finally, the actual value is described by information contained in the *facets* of a slot. These facets contain information or procedures concerning the slot. For example, the VALUE-TYPE facet of the BALANCE attribute signifies that only REAL values are acceptable fillers of this slot.

Some slots contain information about an object's capabilities or relations between objects. For example, the *INSTANCE* slot of a class object contains the names of those objects which instantiate that class.

```
(checking-account
  (BALANCE:
    (VALUE-TYPE REAL))
  (MESSAGES:
    (VALUE write-checks deposit-cash balance-inquiry))
  (SUPERIORS:
    (VALUE bank-account))
  (INSTANCES:
    (VALUE sam-checking-account)))
```

```
(sam-checking-account
  (BALANCE:
    (VALUE 14.01))
  (INSTANCE-OF:
    (VALUE checking-account)))
```

Figure 2: Example Checking Account Objects

Data abstraction is a defining premise of the object-oriented idea. Data abstraction requires that calling programs not make any assumptions about the implementation or the internal representations of the data types that they use. In object-oriented systems, message passing is used to hide the internals of an object. The user of an object can only use the messages that are defined for an object. Hence, unless explicitly provided, there is no way to see how an object is implemented. For example, we can not determine whether our checking account is stored as paper in a file cabinet or electronically on a computer.

This idea becomes more powerful when messages are defined together to provide a complete interface to a set of objects. Such a set of related messages is known as a *protocol*. Protocols are typically general enough to allow for alternative implementations [STE86].

When different classes of objects share a common protocol, then a program can treat different objects in exactly the same way. This is known as *polymorphism*. Polymorphism allows an abstraction away from the types of objects that a program manipulates, as long as each potential object supports the expected protocol.

For a more complete discussion of object-oriented languages and polymorphism, see [STE86].

2.2 Why are Objects a Good Choice?

Since our desired result is an interface to multiple databases, there are several good reasons to use an object-oriented language to model both the objects in the databases and the DBMSs themselves:

- Using the idea of polymorphism described above, we define the ADBMS protocol. Once we have extended our object language so that our objects

can use the messages of this protocol, then our objects can access data from any DBMS which responds to the the actual-DBMS messages.

- All that is required to add a new DBMS to the system is that an object be created which can interpret the messages of the actual DBMS protocol.
- Objects exist independently of the databases. It is thus easy to create an object model which provides an independent, higher-level interface to a set of databases for use by applications or users.
- Objects can be used to store intensional information. For example, the object formalism makes it simple to use knowledge rules, active value procedures, default values, constraint information, etc. Hence, the representational power of objects complements the efficient storage capabilities of database systems.
- Typically, a database relation represents a class of objects, while entries in a table correspond to instances of a class. For example, if our database contained a single table with a million entries, just one object would be created to represent the structure of the table. An instance object would *NOT* be created for each entry. Instead, queries would be defined to return information about any entry; ie: a virtual instance object can be created upon request.
- Because objects are arranged as a hierarchy, attaching queries to objects can be thought of as a way of structuring the set of possible database queries.

2.3 KOREL—A Knowledge Object REpresentation Language

There are numerous object-oriented languages, notably LOOPS and Smalltalk. KOREL was written in Gold Hill Common LISP on the IBM PC/AT. KOREL is distinctive from typical object languages in the following areas: inference, representation, inheritance, and interface, each discussed in more detail below.

Since KOREL allows procedures to operate directly on objects (as opposed to constraining object access to message-passing), KOREL is considered a hybrid frame/object language instead of a "pure" object-oriented language. The distinction is unimportant for the purposes of this report.

KOREL's specification is provided in Appendix A.

2.3.1 Inference

KOREL provides a means of representing knowledge as *if-then rules*, also known as *situation-action* or *production rules*. The definition and usage of these rules is discussed in [WIN84].

Figure 3 shows the general format of a KOREL rule and a specific example. KOREL supports both *forward* and *backward chaining* of a set of rules.

Forward chaining means that the system starts with a set of facts, and repeatedly applies its rules until no more rules fire. For example, if we had the facts: (BEAR teeth-shape pointed), (BEAR claws yes) and (BEAR eyes-point forward) in our facts base, and we forward chained given rule IDENTIFY6, then (BEAR type carnivore) would be added to our facts base.

Backward chaining, on the other hand, starts with a top level goal. Rules which confirm that goal are tried. If a selected rule has unconfirmed antecedents,

Rule Format:

```

(RULE <rule-name>
  (IF <fact1> ... <facts>)
  (THEN <fact1> ... <factm>))

```

Fact Format:

```

(<object-or-pattern> <attribute-or-pattern> <value-or-pattern>)
where:

```

object is the name of an object

attribute is an attribute of that object

value is an acceptable value for that attribute

pattern is one of the following:

(> var-name) binds a variable for the rule

(< var-name) uses a previously-bound variable

Example:

```

(RULE IDENTIFY6
  (IF ((> animal) teeth-shape pointed)
      ((< animal) claws yes)
      ((< animal) eyes-point forward)
  (THEN (< animal) type carnivore)))

```

Figure 3: Format of a KOREL Rule

then the system will begin to backward chain on those. The result of backward chaining is that the top level goal is either confirmed or denied. Backchaining on the fact (BEAR type carnivore) will cause the system to see if (BEAR type carnivore) is in the facts base. If it is not, then all rules which could conclude this fact are returned. IDENTIFY6 in the figure above is one such rule. KOREL would then try to confirm each of the antecedents of IDENTIFY6, backchaining on them if they are unknown. If each of its antecedents can be confirmed, then (BEAR type carnivore) is confirmed. This strategy is appropriate when there is a large quantity of data, but only a small number of goals to be confirmed or denied.

Rules in KOREL can be attached to the *RULE-SET* attribute, or to the *RULES* facet of a slot.

For a more in-depth discussion of what rules are and how forward and backward chaining work, see [WIN84].

2.3.2 Representation

Much of KOREL's power comes from its abilities to represent information about an object.

Figure 4 shows an example of the facets that a KOREL object can specify:

- The value of an attribute can either be true in all cases or only a "good guess" at a value. Values of the first type are stored in the *VALUE* facet of a slot, whereas the latter type are stored as *DEFAULTs*.
- Often, there are times when we would like to use a procedure in association with an action on a slot. These are known as *active values* or *demons*. *IF-NEEDED* procedures are used if no *VALUE* or *DEFAULT* is present. One


```

(parent
  (NAME:
    (VALUE-TYPE STRING))
  (AGE:
    (IF-NEEDED 'ASK)
    (SELF-CONSTRAINTS greater-than-0 less-than-150))
  (GENDER:
    (CHOICES male female))
  (TITLE:
    (RULES identify-dad identify-mom))
  (CHILDREN:
    (MULTIPLE-VALUE-F t)
    (IF-ADDED 'congratulate))
  (TEENAGERS-AT-HOME:
    (MULTIPLE-VALUE-F t)
    (IF-REMOVED 'congratulate))
  (CAR-TYPE:
    (DEFAULT station-wagon))
  (INSTANCES:
    (QUERY (ORACLE PEOPLE (NAME) (= OCCUPATION PARENT))))))

(RULE identify-dad
  (IF ((> parent) GENDER MALE)
    (THEN ((< parent) TITLE DAD)))

(RULE identify-mom
  (IF ((> parent) GENDER FEMALE)
    (THEN ((< parent) TITLE MOM)))

```

Figure 4: An Example Showing KOREL Facets

common IF-NEEDED demon is ASK, which simply asks the user for the value.

KOREL also has *IF-ADDED* and *IF-REMOVED* demons, which are executed whenever a value is placed into or removed from a slot, respectively. The CONGRATULATE demon in the above example prints out a congratulatory message whenever a value is added to the CHILDREN slot or removed from the TEENAGERS-AT-HOME slot.

- Other facets restrict the type of value which may be stored in a slot.

VALUE-TYPE contains a type such as INTEGER, STRING, etc. If the user tries to fill a slot with a value which is not of the indicated type, then that value is rejected.

Similarly, *SELF-CONSTRAINTS* contains a set of predicates. If placing a value in the slot would violate one of these predicates, then the value is rejected. For example, the constraints associated with the AGE slot is that its value be greater-than-zero and less-than-150. If we tried to indicate that a PARENT's age was -30, then that value would be rejected.

The *CHOICES* facet contains a set of allowable values for a slot (assuming that they can be enumerated). Only values which are elements of this set are acceptable.

Finally, *MULTIPLE-VALUE-F* is not a constraint on the value of a slot, but on the number of values of a slot. Hence, if the MULTIPLE-VALUE-Flag for a slot is true, then the slot stores and returns multiple items, for example, a PARENT can have multiple CHILDREN. Other times, only one value is appropriate, like the value for the GENDER attribute.

- *RULES* contains a set of rules which may be applicable to the contents of a slot. For example, the parent object can determine whether to call a

parent "mom" or "dad" by applying the rules associated with the TITLE slot.

- The contents of the *QUERY* facet is a multi-query. This is one way to allow an object to get information from a database. This facet and its use are discussed in more detail in a later chapter.

Appendix B contains the specification of KOREL's facets.

2.3.3 Inheritance

KOREL provides for two different types of inheritance: superior/subordinate and instance.

Superior/Subordinate inheritance arises among class objects when one class is a specialization of another class. For example, a checking account class is a subordinate of the more general bank-account class.

All objects have superiors except for one, the CLASS class. All objects thus eventually have CLASS as a superior. Inversely, all classes may be reached by following a subordinate chain from CLASS.

A class object may have multiple superiors. For example, CIRCUS-ELEPHANTS is a subordinate class of both ELEPHANTS and CIRCUS-ANIMALS. This means that it inherits values and procedures from each of its superiors.

To instantiate a class, a message is sent to it, telling it to create an instance of itself. Specific values are then filled in.

2.3.4 Interface

One of KOREL's design principles is usability. This is demonstrated by the user interface. KOREL has a menu-driven interface, which means that you

```

#####
: KOREL: Knowledge Object REpresentation Language :
L#####
: Display objects  Type KOREL commands  Define queries  Help :
#####

#####
: KOREL top-Level Help Window :
L#####
:
: right/left arrows : move forward/back to next option :
: F1 key           : choose option :
: ESC key          : leave KOREL :
:
: Select display to see what an object looks like, as well as :
: its superiors, subordinates, and instances are. :
:
: Select modify to type in korel commands. :
:
: This is the help screen. :
:
: Press any key when ready. :
:
#####

```

Figure 5: KOREL top level menu

can do almost everything by selecting choices from a menu. You can also type commands directly to KOREL, by-passing the menu system. Finally, ample help is available by selecting *help* options.

Figure 5 shows the KOREL top level menu. As you can see, there are several choices available to the user, viewing the objects, modifying or creating objects, defining queries to connect the objects to a database, and getting help.

Figure 6 shows the KOREL display-object screen, which prints the contents of a KOREL object as well as those objects near it in the inheritance hierarchy, if any. The user can select an object's superior, subordinate, or instance, or enter the name of any other object to be displayed next.

```

#####
: KOREL: Knowledge Object REpresentation Language :
#####
: Display objects Type KOREL commands Define queries Help :
#####

)#####:#####:
: CHECKING-ACCOUNT :: Superiors, Subordinate :
#####~#####~#####
: :: Superiors: :
: :: BANK-ACCOUNT :
: CHECKING-ACCOUNT: :: :
: BALANCE: :: Subordinates: :
: (VALUE-TYPE REAL) :: No SUBORDINATES :
: :: :
: MESSAGES: :: Instances: :
: (MULTIPLE-VALUE-F T) :: SAM-CHECKING-ACCOUNT :
: (VALUE WRITE-CHECKS DEPOSIT-CASH: :: :
: BALANCE-INQUIRY) :: HELP :
: #####:
: :
: :
: :
#####

```

Figure 6: KOREL Display-Object Screen

```

#####:
: KOREL: Knowledge Object Representation Language:
#####:
: Display objects Type KOREL commands Define queries Help:
#####:

#####:
: Enter KOREL commands: type (korel-help) for list:
#####:
:
: KOREL: Type QUIT when finished.:
: * (display-classes):
: CLASS:
: SUBS of CLASS: (PERSON THING):
: SUBS of PERSON: (STUDENT EMPLOYEE):
: SUBS of THING: (UNIVERSITY):
: SUBS of STUDENT: (TA):
: SUBS of EMPLOYEE: (STAFF FACULTY):
: SUBS of FACULTY: (TA):
:
: NIL:
: *:
:
:
:
#####:

```

Figure 7: KOREL Enter-Commands Screen

Figure 7 shows the screen which can be used to enter KOREL commands directly. Those commands which can be entered are described in Appendix A. Typing QUIT in this screen returns the user to the top-level menu.

3 ADBMS - An Abstract Database Management System

The ADBMS is a higher-level conceptual DBMS. It is higher-level because it can send queries to multiple DBMSs and combine their results.

Figure 8 shows the ADBMS protocol. This protocol defines two types of queries: multi-queries and single-queries. *Multi-queries* are made up of single queries, combined by either concatenation or relational joining. (*Single-queries*, messages which are sent to the actual DBMS modules, are discussed in the next chapter.) The results of a multi-query can be grouped together to provide other information. Grouping operations include: finding the MINIMUM or MAXIMUM value of those returned, finding how many values were returned (CARDINALITY), and SUMming the values returned. These operations are specified in Appendix C.

The actual ADBMS module executes multi-queries. Single-queries are sent to the actual DBMS modules. The results of these queries are combined using procedures in this module.

For example, let's suppose there is a list of all the workers of a company stored in the EMP table of an ORACLE database, and a list of departments and locations stored in the DEPT table of an IDMS database, and we wish to perform a relational join to find the city where each employee works. Figure 9 shows the form and results of a multi-query to obtain the desired results.

BNF notation for multi-query

<multi-query>: (JOIN <column-name> <multi-query> <multi-query>) |
 (CONCAT <multi-query> <multi-query>) |
 (<group-relation> <multi-query>) |
 <single-query>

<group-relation>: NO-NULLS | AVERAGE | SUM |
 MINIMUM | MAXIMUM | CARDINALITY

<single-query>: <dbms> <object> <result-form> <conditions> |
 <dbms> <object> <result-form>

<dbms>: the name of a DBMS.

<object>: the name of a database table.

<result-form>: <column-name>

<column-name>: the name of a column in a DBMS table

<conditions>: || <pattern>

<pattern>: <relation> <column-name> <col-or-val> |
 (AND <pattern> <pattern>) |
 (OR <pattern> <pattern>) |
 (NOT <pattern>)

<relation>: = | > | < | >= | <= | <>

<col-or-val>: <column-name> | literal-value

Format of Returned Values:

((column-name1 ... column-namen)
 (val11 ... val1n)
 (valm1 ... valmn))

Figure 8: ADBMS Protocol


```
=> (send-multi-query  
    '(JOIN DEPT-NO  
          (ORACLE EMP (EMP-NAME DEPT-NO))  
          (IDMS DEPT. (DEPT-NO CITY))))
```

Joining on the DEPT-NO key:

```
((EMP-NAME DEPT-NO)  
 (ALLEN 30)  
 (SMITH 20)  
 (ADAMS 20))
```

```
((DEPT-NO CITY)  
 (20 DALLAS)  
 (30 CHICAGO))
```

This is the result:

```
((EMP-NAME CITY)  
 (ALLEN CHICAGO)  
 (SMITH DALLAS)  
 (ADAMS DALLAS))
```

Figure 9: ADBMS Relational Join Example

4 Actual DBMS modules

4.1 Required Messages and Use

Figure 10 shows the protocol which must be supported by a DBMS in order to interface with KOREL objects. This protocol is itself implemented by an object for each DBMS. Hence, each DBMS has an object associated with it that can process these four messages:

- SELF-INFO prints out a description of the DBMS.
- GET-TABLES returns a list of all the tables which the user (or application) can access in the database.
- GET-COLUMNS returns all the columns for a specified table.

SELF-INFO

DESCRIPTION: Prints out a short description of the DBMS.

GET-TABLES

DESCRIPTION: Connects to the database, returns a list of all the tables accessible to the user in the database.

GET-COLUMNS <table-name>

DESCRIPTION: Connects to the database, returns a list of all the columns accessible to the user from the specified table.

GET-DATA <single-query>

DESCRIPTION: Creates a database query from the single-query pattern, connects to the database, executes that query, transforms the results into the ADBMS return-values format, and returns it.

Figure 10: Actual DBMS Module Protocol

- **GET-DATA** is the key message of the interface. This message takes a single-query which is to be translated and passed to the DBMS. These queries have four parts. The first part of a single-query specifies which DBMS to send it to. (This has been stripped off by the time the single-query message reaches the actual DBMS object.) The second section names the table which contains the desired information. The third part describes the information (columns) to be returned. The optional final section lists conditions which must be satisfied for an entry to be selected. After a single query has been executed, its results are translated to a common format and returned.

The only actual DBMS module which has been implemented to date is for the ORACLE relational DBMS on the IBM PC. The difficulty of adding others is discussed in the following section.

4.2 Adding a New Actual DBMS

It is relatively easy to add a new DBMS to the existing system. To accomplish this, a module representing the interface must be constructed. This module

must create an object which represents the DBMS and define procedures for handling the messages described in the previous section.

Typically, adding a new DBMS will not result in a change to any existing applications. To make use of objects in the new DBMS, however, queries will have to be changed to examine the data in the new database. For example, suppose our application currently sends a multi-query that gets the names of employees from an ORACLE database. To make our application also retrieve employees from an IDMS database, the multi-query that it sends must be extended. Hence, the only thing about our object that we have to change is the multi-query that it sends.

Constructing an actual DBMS module is a problem of widely varying difficulty. If the database to be added uses the relational model, then the availability of the data dictionary and the similarity of the relational query to the single-query message required by the system implies a relatively simple module construction. Otherwise, additional entities can be created in the database to store information about the tables and columns. In theory, with complex enough modules, even DBMSs which use hierarchical or network data models can be interfaced with KOREL objects. This remains an issue for future research.

Finally, the actual DBMS module contains the physical interface between the KOREL implementation language and the DBMS. In some cases, the DBMS can be called as a procedure upon a properly formed query. In other cases, the use of an intermediary file might be necessary to store a query. Then, the DBMS is activated, reads its input from a file, and outputs to another file. Finally, the KOREL implementation language is again activated, which allows the actual DBMS module to read the results from a file and format them accordingly. The actual DBMS for ORACLE uses this latter approach.

5 BUILD - a Tool for Interactively Defining Queries

BUILD is a tool used to define new queries and attach them to a KOREL object, for later use. This process might seem very complex, as it separates defining a query from its use. However, it is appropriate because the same query might be used numerous times.

BUILD interacts with a developer to define new multi-queries and attach them to the current object model. Because BUILD also interacts with the data dictionary of the current database, it can provide lists of choices to the developer, lessening the amount that they have to remember about the structure of that database. BUILD also hides the details of connecting to the database and reformatting the query results.

BUILD is called by executing the (BUILD) command. Its initial screen is shown in Figure 11.

The first step in the BUILD process is to identify the current data model. The user is presented with a list of all the object models that the system knows about. The user can select one of these or choose to create a new one.

Next, BUILD presents the user with a list of all objects which are in the object model. If the user wishes to create a new object, the *OTHER* choice may be selected.

BUILD then asks whether the query is to be attached at the object or slot level. If there are previous queries, BUILD can display them as a reference for the construction of the new query.

BUILD now begins the process of defining a multi-query by displaying a menu of ways to combine multi-queries, shown in Figure 12.

```

BBBB      U   U   IIIII   L           DDD
B   B    U   U   I       L           D  D
BBBB      U   U   I       L           D  DD
B   B    U   U   I       L           D  D
BBBB      UUU   IIIII   LLLLL   DDD

```

Copyright (C) 1986, 1987 by Samuel P. Levine

BUILD allows you to define or modify the abstract database queries that are associated with KOREL objects.

Before using BUILD, a KOREL data model and its objects must already exist. If not, exit BUILD at this point, and use KOREL to build them.

Press any key to begin, or <BREAK> to exit.

Figure 11: BUILD Initial Screen

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
 2. JOIN multi-queries
 3. CONCAT multi-queries
 4. NO NULLS multi-query
 5. AVERAGE multi-query
 6. SUM multi-query
 7. MINIMUM multi-query
 8. MAXIMUM multi-query
 9. CARDINALITY multi-query
- Choose the type of multi-query
Your choice: 6

Figure 12: BUILD Defining a Multi-Query Menu

- *Define single-query* causes BUILD to start building a single-query.
- *JOIN* will first prompt for a column name to act as a key for the relational join. It will then bring up the *Defining a multi-query menu* twice more to define the multi-queries which produce the tables to join on.
- *CONCAT* simply allows the user to define two more multi-queries. The results are put together and returned as a single result.

Note: Because BUILD brings up the multi-query menu each time, it is possible to recursively define JOINS and CONCATs. For example, here is a case in which the second multi-query in a JOIN is a CONCAT multi-query:

(JOIN key (multi-query-1) (CONCAT (multi-query-2) (multi-query-3)))

- The remaining operations are grouping operations. The user selects an operation like *SUM* and then BUILD allows the user to define another multi-query. The operation will then be applied to the result of that multi-query. For example, if the multi-query returned a list of numbers, SUM would return the sum of those numbers. These grouping operations are described in more detail in Appendix C.

The only way to exit the multi-query definition loop is to choose to define a single-query.

BUILD begins a single-query by asking which DBMS is used to store the information. Note: if the information is stored in more than one DBMS, then a multi-query should be defined.

Next, BUILD prompts the user with a list of database tables to find out which table the data is stored in. The user selects one of those which are available.

Defining the select conditions for a query.

If there are multiple select conditions, they can be logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition, or no conditions at all.

Choose 0 for no action.

1. AND
 2. OR
 3. NOT
 4. Define a Pattern
 5. No Search conditions
- Choose how to combine multiple patterns
Your choice: 5

Figure 13: BUILD Combining Select Conditions Menu

Note: if the information is stored in more than one table, then a multi-query should be defined.

BUILD will then ask which columns should be returned. It prompts the user to select from among the columns of the table selected in the previous step. Multiple columns may be selected.

BUILD then pops up a menu to help the user combine the select clauses of a single query, shown in Figure 13.

- The user can select a logical combination, like *AND* or *OR*. These combine two select clauses. Selecting *NOT* will cause BUILD to have the user define one select clause. These logical operators have the expected meaning.
- To escape the logical combination loop, a user can choose to define a comparison clause. This is discussed in more detail below.
- Alternately, the user can stop choosing logical combinations by choosing *NONE*. This has the effect of generating a query which has no select

clauses.

Finally, BUILD pops up a menu to help the user define the comparison clauses of the single-query. First, the user chooses a comparison operators, like $\geq$. Next, BUILD pops up a menu of possible columns so that the user can select a column to examine. Finally, the user selects a column or enters a literal value as the value to compare to. For example, if we only want to select people whose salary is greater than \$30,000, we would first choose $>$, then select the *SALARY* column, and finally enter the literal value *30000*.

Finally, BUILD stores the defined multi-query in the appropriate place in the object. If it is to be used at the object level, the user is first prompted for a sentence which describes the query or the question that the query is meant to answer.

The advantage of having a module like BUILD, aside from decreasing the amount of knowledge that the developer must remember about the database, is that it ensures that all multi-queries are at least syntactically correct.

How to actually use this query is discussed in more detail in the following section.

6 Phase 1: Creating an Object Model

6.1 Overview

Figure 14 shows the modules which interact to build a data model and interface it with databases.

An object model is a model of those things which exist in the domain of concern. Some information about these objects might be contained in a database. These objects are typically arranged in a class hierarchy. Hence, an object representing the entire object model is at the root of the tree (ie: all objects in the object model are subordinate to it).

A similar procedure is followed to modify an existing data model.

- A object model is an instance of the general DATA-MODEL class of objects. An instance of this class should be created with an appropriate name (using the KOREL create-instance command).
- A developer who is familiar with the particular domain of interest uses KOREL to create significant domain objects. (using the KOREL make-object command). In a banking domain, the developer might create CLIENT, TELLER, and CHECKING-ACCOUNT objects. The definition of these objects may be guided by (but is not restricted to) the data found in databases. The developer may want to model information not contained in the database, or there might be information in the database which is not essential for the domain at hand. A procedure to calculate interest on a checking account is an example of an object containing knowledge not in the database, while the database might contain information about Individual Retirement Accounts which is irrelevant to the domain of interest.

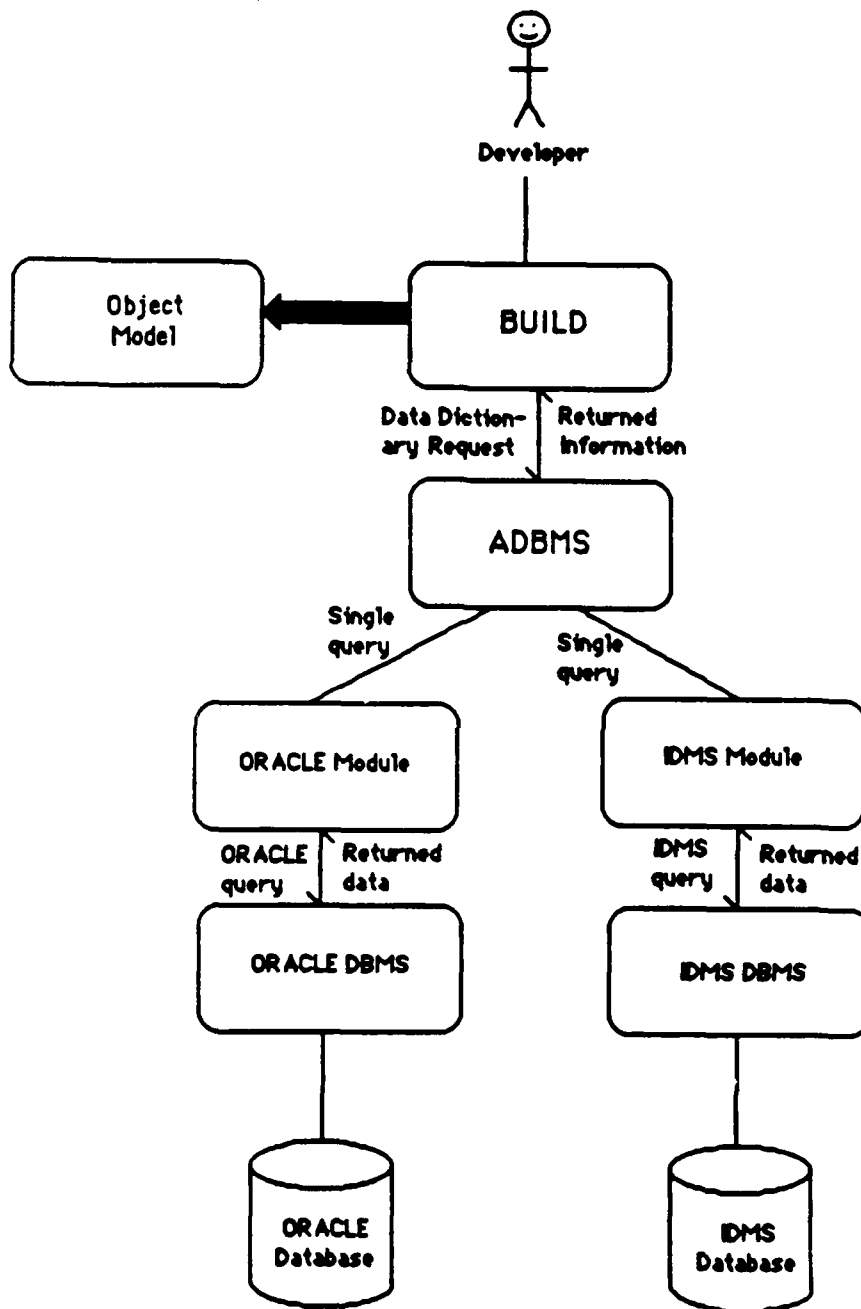


Figure 14: Phase 1: Creating an Object Model

- Next, the developer must determine how the objects will interact with the database, either at the object level or at the slot level. Object level queries exist independently of any slots and are executed explicitly. They are intended to be used for conscious human interaction. For example, we might want to define an object level query labeled "What are all the overdrawn accounts?" to return all the overdrawn account numbers for the CHECKING-ACCOUNT class. Slot level queries, on the other hand, are associated with a slot and are used if no value or default is present. These are more typically used implicitly by human users or by applications programs, during the execution of a KOREL command. For example, we might put a slot in our CHECKING-ACCOUNT object called *OVERDRAWN-ACCOUNTS* which contains a slot level query to return the overdrawn accounts from the databases.
- Once the developer knows what queries are required and where to put them, BUILD is called by typing (BUILD). BUILD will interact with the user and the databases as described in the previous chapter to formulate a multi-query and attach it to the existing object model. Specifically, during the definition of a single-query, BUILD sends GET-TABLES and GET-COLUMNS messages to the appropriate DBMSs to lessen the amount of knowledge that the developer must have about what is in the database and how to find out about it. The complexities of the queries which can be defined are discussed in the following section. Once a query is defined, it is stored so that it can be used numerous times.

The result of this phase is an object model which stores some information locally, but which can also return and operate upon information from multiple databases.

6.2 Connecting Objects to a Database

There are several types of queries which can be defined. Examples of each can be found in the University example in Chapter 8.

- *Simple Retrieval:* A simple retrieval is a query which operates on one database, to return one or multiple columns directly, possibly with some select conditions. In addition, it can be a grouping operation, like SUM or MINIMUM, applied to the result of a simple query. This is well within the capabilities of existing DBMSs.
- *Complex Retrieval:* A complex retrieval in some way combines multiple queries, either by performing a CONCAT or a relational JOIN. In the case where the queries operate on the same database, these queries too are within the typical capabilities of current DBMSs. Those situations where the information must be retrieved from multiple databases, or on multiple machines has also been solved on a case by case basis using translation techniques.
- *Simple Mapping:* Simple mapping is needed in situations where the data to be returned is in different units. Mapping is currently handled by creating a view, in which arithmetic or renaming transformations can be performed.
- *Procedural Capabilities:* This is needed when a sequence of information must be returned. By embedding queries into a program, it is possible to include this capability into a database system, although it is arguably outside the capabilities of the underlying data model. This capability allows the solution of *least fixed point* problems, which cannot be solve by relational calculus or algebra.

- *Heuristic Capabilities:* Often, it is desirable to support heuristic capabilities. This is beyond the capabilities of typical current DBMSs, but can be done by procedurally embedding the knowledge into a higher level language. Again, this capability is outside the functionality of current Database Management Systems.

A typical use of heuristic capabilities would be to provide ways to resolve contradictions of data from multiple sources, or to fill in incomplete information. For more information, see [POC87].

- *Meta-level Capabilities:* Often, for purposes of explanation or justification, it is desirable to explicitly represent knowledge. For example, when presented with some data, one might wonder where that data came from: was it retrieved from the database, derived through some simple procedure, or the result of inferencing some heuristic rules? This process of justifying data requires the system to be able to reference the mechanisms which it uses to return an answer. This ability is outside the database domain, but is a fairly standard AI knowledge representation methodology.

Each of the last three capabilities is an efficiency improvement over the previous one (ie: given procedural capabilities, it is possible to implement heuristics, and certain heuristics can be used to provide meta-level capabilities). It would be possible to only use procedures. However, it would be impractical to implement a knowledge-based system like a forward chaining expert systems using procedures, as the knowledge changes frequently and interacts unpredictably. That same expert system becomes very straight-forward when given the abstraction of heuristics. A similar argument can be used for meta-level capabilities: an expert system which represents its heuristics implicitly cannot reason about their contents. However, the explicit representation of heuristics leads itself to

applications like explanation.

7 Phase 2: Using an Object Model

Figure 15 shows the modules which interact to return information from a KOREL object model defined in the definition phase described in the previous chapter. The way in which KOREL responds depends upon whether the multi-query is called implicitly or explicitly. This in turn depends upon the type of the user. Human users will either explicitly or implicitly call multi-queries, whereas application programs or expert systems will typically only call them implicitly. The differences between these two ways of calling is described below.

- **Implicitly calling multi-queries**

If the value of a slot is requested, KOREL will first look for a value in the VALUE, DEFAULT, and IF-NEEDED facets. If none of these successfully return a value, KOREL will then examine the QUERY facet.

Queries stored in the QUERY facet are assumed to return a value or list of values for the object. These are active values, which means that the query is executed at the time of the request for a value. This minimizes the possibility of using old data.

It is possible to store the retrieved data in a slot, allowing a trade-off between the cost of re-executing a query, and the cost of possibly using dated information. This issue is examined in more depth in Chapter 9.

For example, if we have a class object which represents our EMPLOYEE table, the QUERY facet might contain a multi-query which returns the names of all employees from the database. A user might request the contents of this slot, but the VALUE, DEFAULT, and IF-NEEDED facets are empty. The query would then be executed, and the results returned to the user.

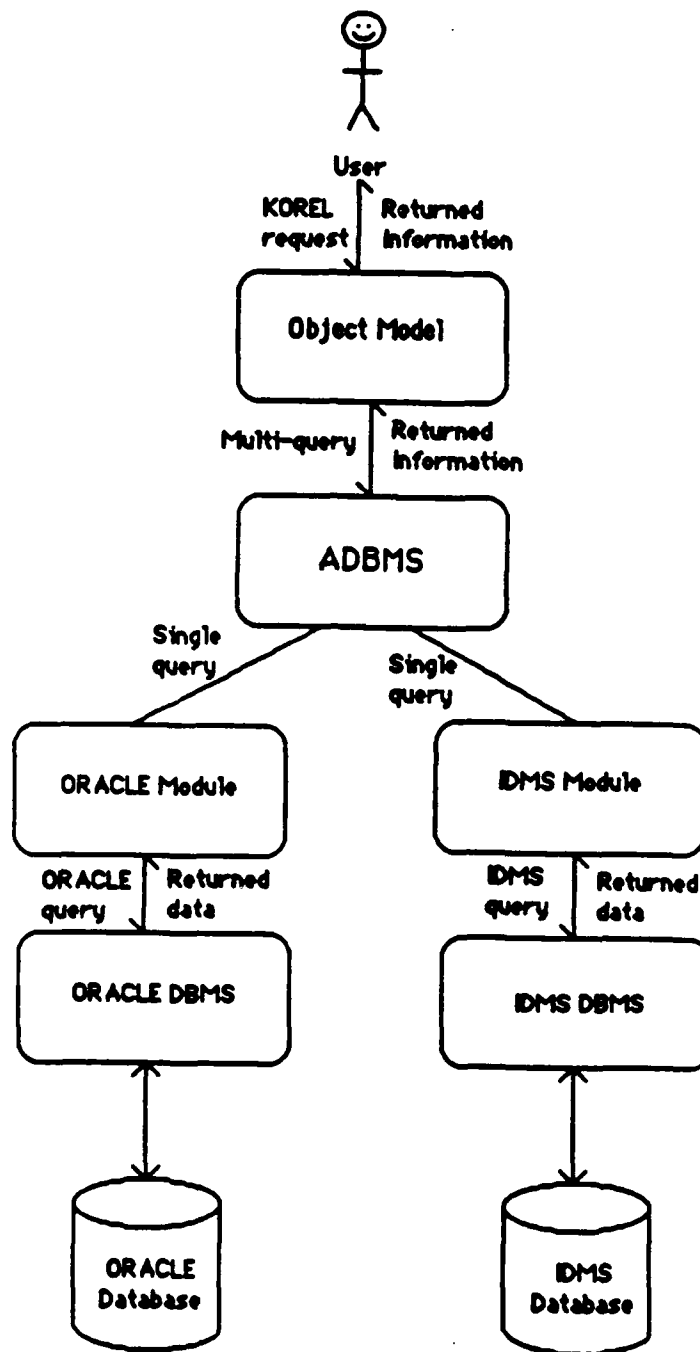


Figure 15: Phase 2: Using an Object Model

- Explicitly calling multi-queries

Alternately, each object contains a slot called *QUERIES*. Each entry in this slot is actually a pair. The first element is a text string which describes what the query does, or the question which the query answers. The second element is the actual multi-query.

These queries are chosen by using the (*SELECT-QUERY object*) function. This function presents the list of queries associated with an object and allows the user to select one of them to execute.

For example, the *EMPLOYEE* object defined above might contain the following descriptions of multi-queries in its *QUERIES* slot:

What is the total weekly payroll?

Which of our employees didn't earn a bonus last year?

Are we an equal opportunity employer?

Whether called explicitly or implicitly, KOREL passes the multi-query to the ADBMS module. The ADBMS responds by breaking the multi-query up into its component single-queries and sending them to the appropriate actual DBMS modules. The DBMS modules respond to a single-query message by generating a corresponding actual query which is executed on the database. The table returned is reformatted by the actual DBMS module and passed back to the ADBMS. The ADBMS combines the results of its single-queries appropriately, by JOINing, CONCATenating, or applying one of the grouping operations. Finally, this result is passed back to the KOREL user.

8 UNIVERSITY: an Extended Example

The use of these tools is demonstrated with an example, the set of databases which might be used and maintained by the administration of a university. I begin by describing the existing database tables, go through the process of designing and creating a useful object model, and define queries to link that object model to the existing database.

I give examples of potentially desirable classes of capabilities which were discussed in Section 6.2. For each, I will describe a typical question which might be asked of the UNIVERSITY model, outline how that question could be answered in a database-only system, and show how it can be answered using objects and multi-queries. Hence, each example has three distinct steps: the question to be answered along with a strategy for answering it, the development process of building queries and attaching them to objects, and finally how these questions would actually be answered.

The examples are annotated for clarity. Horizontal lines separate what the user would see as individual screens. An italicized explanation of what will occur precedes each screen.

8.1 The University Database

There are currently five tables which are used by the example university.

Figure 16 shows the EMPLOYEES table. This table contains a description of all the people the university employs. It is used, among other things, for payroll and tax purposes. Note that employees can be professors, tas (teaching assistants), or staff. The salary figure for each of these classes of employees is figured differently: professors are paid twelve times per year (monthly), tas are

EMPLOYEES			
Name (key)	Position	Salary	Manager
Stu	Professor	50000	Bob
Rich	Professor	50000	Bob
Sam	TA	900	Stu
Tony	TA	900	Rich
Butch	Staff	2000	Waldo
Bob	Staff	3000	Jennifer
Jennifer	Staff	3200	Paul
Waldo	Staff	3000	Paul

Figure 16: University EMPLOYEES Table

paid nine times per year (monthly during the academic year), and staff members are paid bi-weekly (26 times per year).

Figure 17 shows the STUDENTS table, which describes all the students who attend the university. It is used by the registrar's office to determine the registration status of the students, and to help prepare grade reports.

More specific information about a student's schedule is contained in the CURRENT.CLASSES table, shown in figure 18. This table contains those classes a student is taking during the current term.

The previous academic performance of the students is kept by department in the CLASS_HISTORY tables. Each department keeps a record of all the grades that have been received by students majoring in that department. The CLASS_HISTORY for two departments, management and computer-science are shown in figure 19. These tables are used mainly to generate transcripts and calculate GPA's.

STUDENTS			
Name (key)	Major	Advisor	Entered-date
Sam	computer-science	Stu	fall-82
Bill	computer-science	Rich	fall-83
Dane	management	Stu	fall-82
Jeff	management	Rich	fall-83

Figure 17: University STUDENTS Table

CURRENT_CLASSES	
Name (key)	Class (key)
sam	writing
sam	economics
bill	art
bill	chemistry
dane	macroeconomics
dane	french
jeff	manufacturing

Figure 18: University CURRENT_CLASSES Table

COMPUTER_SCIENCE_CLASS_HISTORY		
Name (key)	Class (key)	Grade
sam	physics1	3.0
sam	calculus1	4.0
sam	physics2	4.0
sam	calculus2	3.0
sam	music	3.0
sam	writing	2.0
sam	software	3.0
sam	hardware	3.0
sam	hacking	4.0
sam	thesis	4.0
bill	physics1	3.0
bill	calculus1	4.0
bill	physics2	4.0
bill	calculus2	4.0
bill	art	3.0
bill	writing	4.0
bill	software	3.0
bill	hardware	3.0
bill	hacking	4.0
bill	psychology	3.0

MANAGEMENT_CLASS_HISTORY		
Name (key)	Class (key)	Grade
dane	physics1	2.0
dane	calculus1	3.0
dane	physics2	4.0
dane	calculus2	3.0
dane	art	4.0
dane	writing	3.0
dane	accounting	3.0
dane	finance	4.0
dane	entrepreneurship	4.0
dane	psychology	3.0
jeff	physics1	3.0
jeff	calculus1	4.0
jeff	physics2	2.0
jeff	calculus2	4.0
jeff	music	4.0
jeff	writing	2.0
jeff	accounting	3.0
jeff	finance	3.0
jeff	entrepreneurship	4.0
jeff	psychology	4.0

Figure 19: University CLASS_HISTORY Tables

8.2 Creating a University Object Model

The first step in creating a more user-friendly view of the data in the database is to build an object model.

Initially, we begin at a very rough grain of classification (much like the game which begins with the question "animal, vegetable, or mineral?"). Things which naturally appear at a university are either PEOPLE or THINGS.

Next, there are four kinds of people who may be found at a university, namely PROFESSORS, TAS, STAFF, and STUDENTS. But since the first three of these share many common characteristics (salary, W-4 information, date-hired, etc.) it is more efficient to create a new classification, namely EMPLOYEES. Similarly, since TAS share characteristics of both PROFESSORS and STUDENTS, the TAS class has each of these as a superior. For example, TAS have a salary and a class they're teaching, like professors, as well as having classes they're taking, like students.

A similar process is used to create objects to model university THINGS. The result is the UNIVERSITY object model shown in figure 20.

8.3 Simple Retrieval Example

If the annual course catalog included a list of faculty members, a fairly common question might be "Who are all the Professors?"

In Oracle, this would be answered by executing a query like the following:

```
select NAME
from EMPLOYEES
where POSITION = 'PROFESSOR';
```

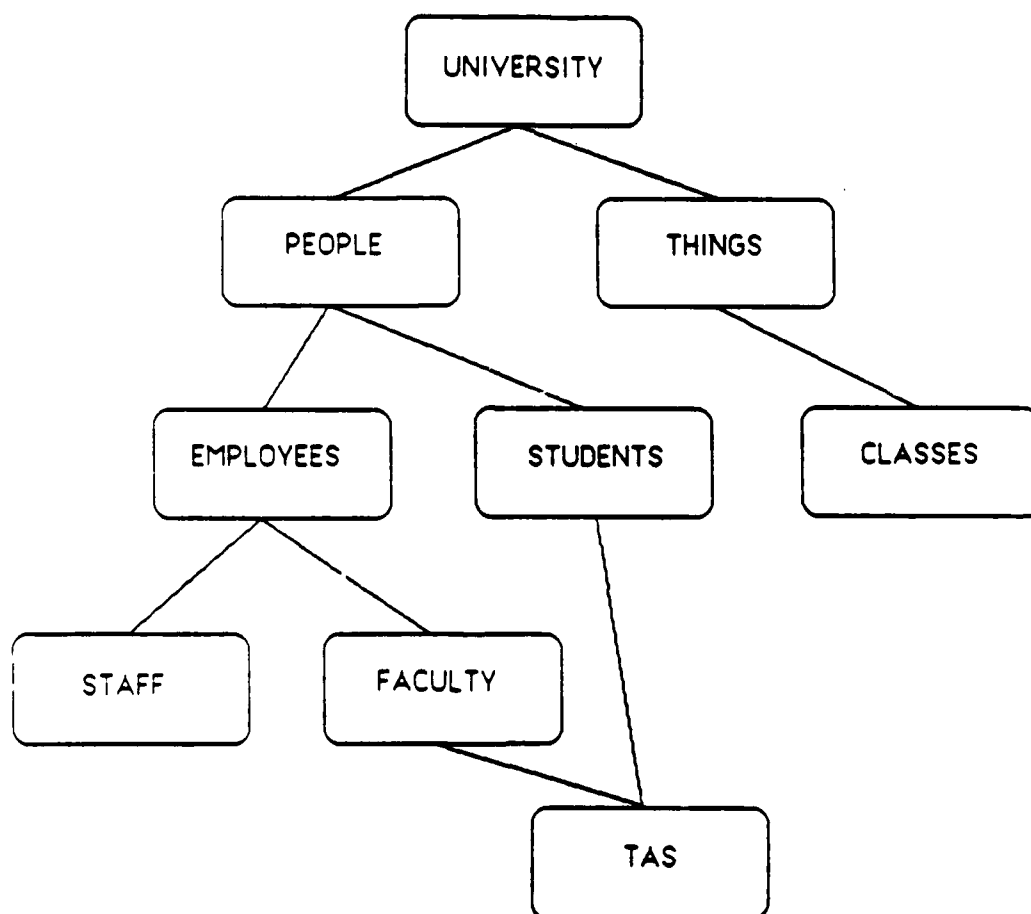


Figure 20: The UNIVERSITY Object Model

In KOREL, we would get this information by typing the command:

```
(get-object 'professors 'instances)
```

This would then cause the following query, in the QUERY facet of the INSTANCES slot to be executed:

```
(ORACLE EMPLOYEES (NAME) (= POSITION 'PROFESSOR'))
```

Which, after the connection to ORACLE had been established, would generate the desired query. This process and its result is shown below.

The developer begins by calling BUILD.

```
* (build)
```

This is the BUILD introductory screen.

```
BBBB  U  U  IIIII  L      DDD
B  B  U  U    I    L      D  D
BBBB  U  U    I    L      D  DD
B  B  U  U    I    L      D  D
BBBB      UU      IIIII  LLLLL  DDD
```

Copyright (C) 1986, 1987 by Samuel P. Levine

BUILD allows you to define or modify the abstract database queries that are associated with KOREL objects.

Before using BUILD, a KOREL object model and its objects must already exist. If not, exit BUILD at this point, and use KOREL to build them.

Press any key to begin, or <BREAK> to exit.

BUILD presents the developer with a list of all the existing object models. The developer selects the one to extend.

Here is a list of KOREL object models that currently exist.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. UNIVERSITY

Choose an existing Object model.
Selected so far: NIL

Your choice: 1

Next, BUILD presents a list of all the components of that object model. In this case, since our question is about professors, it should be attached to the PROFESSORS object.

Here is a list of the objects in the UNIVERSITY object model.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. PEOPLE
2. THINGS
3. EMPLOYEES
4. STUDENTS
5. PROFESSORS
6. STAFF
7. TAS
8. CLASSES

Choose an object to examine.
Selected so far: NIL

Your choice: 5

The query is to be attached to a slot so it will be automatically executed whenever the value of that slot is needed.

Attach queries at the (O)bject or (S)lot level? S

The user creates a new attribute of the PROFESSORS object: NAMES.

Here is a list of PROFESSORS's attributes.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. CREATE-NEW-SLOT
2. TITLE
3. DEPARTMENT

Choose the slot to attach the query to:

Selected so far: NIL

Your choice: 1

Enter the name of a new slot to store it in: names

Now BUILD interacts with the developer to define a multi-query to return the value of NAMES for PROFESSORS. We only need a single-query.

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 1

Next, the system presents a list of all the DBMSs it knows about. This system only knows how to access ORACLE, so that is the developer's only choice.

Defining a single-query

Here is a list of DBMSs that it is possible to communicate with.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. ORACLE

Select the DBMS to send the single-query to.

Selected so far: NIL

Your choice: 1

Now, BUILD accesses the ORACLE data dictionary to retrieve all the tables that the developer can access. First, the ORACLE query which is automatically

generated is shown, then the results from the database, and finally, the results translated into the KOREL format

Getting all tables from the database
to select one which contains data about the object.

If there are multiple, then define a multi-query.
I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select TNAME from CATALOG;
```

```
spool off
```

Here is the table that was returned:

```
UFI> select TNAME from CATALOG;
```

```
TNAME
```

```
-----
STUDENTS
CURRENT_CLASSES
MANAGEMENT_CLASS_HISTORY
COMPUTER_SCIENCE_CLASS_HISTORY
EMPLOYEES
```

```
UFI> spool off
```

This is transformed into the KOREL format:
((TNAME) (STUDENTS) (CURRENT_CLASSES) (MANAGEMENT_CLASS_HISTORY)
(COMPUTER_SCIENCE_CLASS_HISTORY) (EMPLOYEES))

The developer can now select the table which contains the desired information.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. STUDENTS
2. CURRENT_CLASSES
3. MANAGEMENT_CLASS_HISTORY
4. COMPUTER_SCIENCE_CLASS_HISTORY
5. EMPLOYEES

Select the table which contains object data
Selected so far: NIL

Your choice: 5

Next, BUILD retrieves a list of all the columns of that table, so the developer can select those to be returned.

Getting all columns from the current table

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select CNAME from COL where TNAME = 'EMPLOYEES';

spool off
```

Here is the table that was returned:

```
UFI> select CNAME from COL where TNAME = 'EMPLOYEES';
```

CNAME

NAME

POSITION

SALARY

MANAGER

UFI> spool off

This is transformed into the KOREL format:
((CNAME) (NAME) (POSITION) (SALARY) (MANAGER))

The developer goes through the list, selecting those columns to be returned. In this case, the developer specifies that only the NAME column should be retrieved.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return
Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return
Selected so far: (NAME)

Your choice: 0

Next, the developer specifies the conditions for retrieving rows from the database. Here, only a single condition is needed.

Defining the select conditions for a query.

If there are multiple select conditions, they can be logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition, or no conditions at all.

Choose 0 for no action.

1. AND
2. OR
3. NOT
4. Define a Pattern
5. No Search conditions

Choose how to combine multiple patterns

Your choice: 4

Only PROFESSORS should be returned. The user starts by selecting = (prefix notation is used throughout KOREL).

This is to define a single search condition.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. =
2. >
3. <
4. >=
5. <=
6. <>

Choose a relation for a search condition

Selected so far: NIL

Your choice: 1

Here we specify what to look at (POSITION).

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Choose a column
Selected so far: NIL

Your choice: 2

Finally, the value to compare to is chosen. In this case, we only want "PRO-FESSORS", so we enter that as a literal.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. SPECIFY-A-LITERAL
2. NAME
3. POSITION
4. SALARY
5. MANAGER

Specify a Column or Value
Selected so far: NIL

Your choice. 1

Enter a literal value to compare to: PROFESSOR

Here is the finished query in its printed and internal representation.

DBMS: ORACLE

TABLE: EMPLOYEES

COLS: (NAME)

WHERE: (= POSITION "PROFESSOR")

(ORACLE EMPLOYEES (NAME) (= POSITION "PROFESSOR"))

This is an example of how that query might be used. It starts by using a KOREL command to get the value of NAMES in the PROFESSORS object. This causes a multi-query to be sent to the ADBMS, which processes it, finds it is a single-query, then sends it to the ORACLE actual-DBMS module. An ORACLE query is generated, executed, and its results are translated into the KOREL format and returned to the user

```
* (get-object 'professors 'names)
I passed the following string to ORACLE
```

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select NAME from EMPLOYEES where POSITION = 'PROFESSOR';
spool off
```

Here is the table that was returned:

```
UPI> select NAME from EMPLOYEES where POSITION = 'PROFESSOR';
```

```
NAME
-----
STU
RICH
```

```
UPI> spool off
```

This is transformed into the KOREL format:

```
((NAME) (STU) (RICH))
```

```
((NAME) (STU) (RICH))
```

```
*
```

8.4 Complex Retrieval Example

Another question we might want to answer is "What classes are Stu's advisees taking?"

It is more difficult to answer this question because it requires first accessing the STUDENTS table to find out who Stu's advisees are, and then accessing the CURRENT_CLASSES table to see what classes those people are taking. Although in this specific situation, the two tables are in the same database, the procedure followed is exactly the same as it would be if STUDENTS were an IDMS database on an IBM mainframe computer, and CURRENT_CLASSES were an ORACLE database on a DEC minicomputer.

The process required for KOREL to answer this question is shown below.

This is the BUILD introductory screen.

* (build)

```

BBBB      U  U   IIIII  L        DDD
B  B      U  U     I     L        D  D
BBBB      U  U     I     L        D DD
B  B      U  U     I     L        D  D
BBBB      UUU   IIIII  LLLLL  DDD

```

Copyright (C) 1986, 1987 by Samuel P. Levine

BUILD allows you to define or modify the abstract database queries that are associated with KOREL objects.

Before using BUILD, a KOREL object model and its objects must already exist. If not, exit BUILD at this point, and use

KOREL to build them.

Press any key to begin, or <BREAK> to exit.

The developer is interested in the UNIVERSITY object model.

Here is a list of KOREL object models that currently exist.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. UNIVERSITY

Choose an existing Object model.

Selected so far: NIL

Your choice: 1

We're interested in information about a specific professor.

Here is a list of the objects in the UNIVERSITY object model.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. PEOPLE
2. THINGS
3. EMPLOYEES
4. STUDENTS
5. PROFESSORS
6. STAFF
7. TAS
8. CLASSES

Choose an object to examine.

Selected so far: NIL

Your choice: 5

This time we have a specific question to answer, so it is more appropriate to attach the query at the object level.

Attach queries at the (O)bject or (S)lot level? 0

No queries are currently defined at the object level for PROFESSORS.

Here is a list of the queries currently attached to the PROFESSORS object. If one of them has close to the desired functionality, choose and modify it.

Otherwise, choose NONE-ARE-CLOSE to define a new query.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NONE-ARE-CLOSE

Is one of the following close to what you want?

Selected so far: NIL

Your choice: 1

To answer this question requires a relational-join: we must find the students who have STU as an advisor and then join that list with the CURRENT-CLASSES table.

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 2

JOIN on which column (both must have the same name)?NAME

We begin by defining the first multi-query of the join.

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 1

NO 8155-833

OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE
SEMANTICS VOLUME 4(U) MASSACHUSETTS INST OF TECH
CAMBRIDGE A GUPTA ET AL DEC 87 MIT-K011SE-4

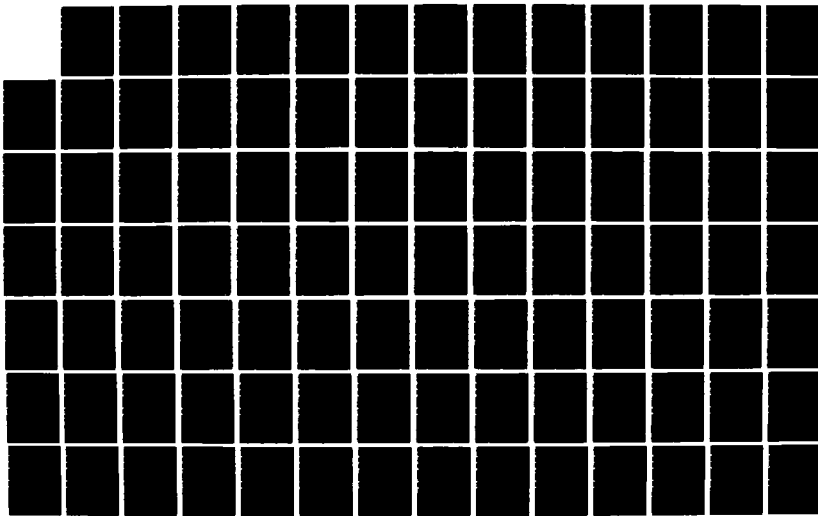
274

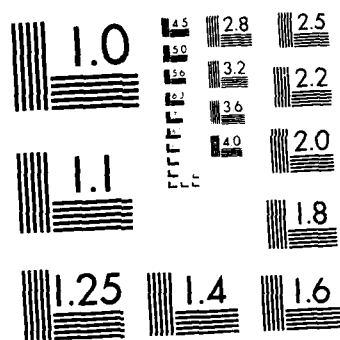
UNCLASSIFIED

DTR557-85-C-00003

F/G 12/7

NL





The information's stored in ORACLE.

Defining a single-query

Here is a list of DBMSs that it is possible to communicate with.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. ORACLE

Select the DBMS to send the single-query to.
Selected so far: NIL

Your choice: 1

Getting all tables from the database
to select one which contains data about the object.

If there are multiple, then define a multi-query.
I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select TNAME from CATALOG;
```

```
spool off
```

Here is the table that was returned:

```
UFI> select TNAME from CATALOG;
```

TNAME

STUDENTS
CURRENT_CLASSES
MANAGEMENT_CLASS_HISTORY
COMPUTER_SCIENCE_CLASS_HISTORY
EMPLOYEES

UFI> spool off

This is transformed into the KOREL format:
((TNAME) (STUDENTS) (CURRENT_CLASSES) (MANAGEMENT_CLASS_HISTORY)
(COMPUTER_SCIENCE_CLASS_HISTORY) (EMPLOYEES))

We need data from the STUDENTS table.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. STUDENTS
2. CURRENT_CLASSES
3. MANAGEMENT_CLASS_HISTORY
4. COMPUTER_SCIENCE_CLASS_HISTORY
5. EMPLOYEES

Select the table which contains object data
Selected so far: NIL

Your choice: 1

Getting all columns from the current table

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select CNAME from COL where TNAME = 'STUDENTS';
```

```
spool off
```

Here is the table that was returned:

```
UFI> select CNAME from COL where TNAME = 'STUDENTS';
```

```
CNAME
```

```
-----
```

```
NAME
```

```
MAJOR
```

```
ADVISOR
```

```
ENTERED_DATE
```

```
UFI> spool off
```

This is transformed into the KOREL format:

```
((CNAME) (NAME) (MAJOR) (ADVISOR) (ENTERED_DATE))
```

We only need the NAMES of the students.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. MAJOR
3. ADVISOR
4. ENTERED_DATE

Select those columns to return

Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. MAJOR
3. ADVISOR
4. ENTERED_DATE

Select those columns to return
Selected so far: (NAME)

Your choice: 0

We only need those students who have STU as an advisor.

Defining the select conditions for a query.

If there are multiple select conditions, they can be
logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition,
or no conditions at all.

Choose 0 for no action.

1. AND
 2. OR
 3. NOT
 4. Define a Pattern
 5. No Search conditions
- Choose how to combine multiple patterns

Your choice: 4

This is to define a single search condition.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. =
2. >
3. <
4. >=
5. <=
6. <>

Choose a relation for a search condition

Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. MAJOR
3. ADVISOR
4. ENTERED_DATE

Choose a column

Selected so far: NIL

Your choice: 3

Choosing a selected item de-selects it.

Choose 0 to exit.

1. SPECIFY-A-LITERAL
2. NAME
3. MAJOR
4. ADVISOR
5. ENTERED_DATE

Specify a Column or Value
Selected so far: NIL

Your choice: 1

Enter a literal value to compare to: STU

Now we begin defining the second multi-query for the join.

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 1

Defining a single-query

Here is a list of DBMSs that it is possible to communicate with.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. ORACLE

Select the DBMS to send the single-query to.
Selected so far: NIL

Your choice: 1

Getting all tables from the database
to select one which contains data about the object.

If there are multiple, then define a multi-query.
I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select TNAME from CATALOG;

spool off
```

Here is the table that was returned:

```
UFI> select TNAME from CATALOG;
```

```
TNAME
-----
STUDENTS
CURRENT_CLASSES
MANAGEMENT_CLASS_HISTORY
```

COMPUTER_SCIENCE_CLASS_HISTORY
EMPLOYEES

UFI> spool off

This is transformed into the KOREL format:

((TNAME) (STUDENTS) (CURRENT_CLASSES) (MANAGEMENT_CLASS_HISTORY)
(COMPUTER_SCIENCE_CLASS_HISTORY) (EMPLOYEES))

Now we need information from the *CURRENT_CLASSES* table.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. STUDENTS
2. CURRENT_CLASSES
3. MANAGEMENT_CLASS_HISTORY
4. COMPUTER_SCIENCE_CLASS_HISTORY
5. EMPLOYEES

Select the table which contains object data
Selected so far: NIL

Your choice: 2

Getting all columns from the current table

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select CNAME from COL where TNAME = 'CURRENT_CLASSES'
```


spool off

Here is the table that was returned:

UFI> select CNAME from COL where TNAME = 'CURRENT_CLASSES';

CNAME

NAME

CLASS

UFI> spool off

This is transformed into the KOREL format:

((CNAME) (NAME) (CLASS))

This time we need to return two columns: NAME and CLASS.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. CLASS

Select those columns to return

Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. CLASS

Select those columns to return

Selected so far: (NAME)

Your choice: 2

No select conditions this time.

Defining the select conditions for a query.

If there are multiple select conditions, they can be logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition, or no conditions at all.

Choose 0 for no action.

1. AND
2. OR
3. NOT
4. Define a Pattern
5. No Search conditions

Choose how to combine multiple patterns

Your choice: 5

Now the query is complete, so we associate the question that the query answers with it and store it in the PROFESSORS object.

Query constructed.

What does this query do? What classes are Stus advisees taking this term?

The query is selected by using the select_query command, which displays a list of all the object-level queries associated with an object and asks the user to select one to execute. The selected query is sent to the ADBMS which breaks it down and sends its components to the ORACLE DBMS object. The ADBMS then performs the relational join and returns the results to the user.

```
(" What classes are Stu's advisees taking this term?"
(QUOTE (JOIN "NAME" (ORACLE STUDENTS (NAME) (= ADVISOR "STU")))
      (ORACLE CURRENT_CLASSES (NAME CLASS) NIL))))
```

```
* (select-query 'professors)
```

Choosing a selected item de-selects it.

Choose 0 to exit.

1. What classes are Stu's advisees taking this term?

Choose a query to execute

Selected so far: NIL

Your choice: 1

JOIN

NAME

DBMS: ORACLE

TABLE: STUDENTS

COLS: (NAME)

WHERE: (= ADVISOR "STU")

DBMS: ORACLE

TABLE: CURRENT_CLASSES

COLS: (NAME CLASS)

WHERE: NIL

I passed the following string to ORACLE

```
connect dbbess/san
```

```
set pause off
```

```
spool c:\gclisp2\monti\from-ora\from-ora.lsp
```

```
select NAME from STUDENTS where ADVISOR = 'STU';
```

```
spool off
```

Here is the table that was returned:

```
UFI> select NAME from STUDENTS where ADVISOR = 'STU';
```

```
NAME
```

```
-----
```

```
SAM
```

```
DANE
```

```
UFI> spool off
```

This is transformed into the KOREL format:

```
((NAME) (SAM) (DANE))
```

I passed the following string to ORACLE

```
connect dbbess/sam
```

```
set pause off
```

```
spool c:\gclisp2\monti\from-ora\from-ora.lsp
```

```
select NAME , CLASS from CURRENT_CLASSES;
```

```
spool off
```

Here is the table that was returned:

```
UFI> select NAME, CLASS from CURRENT_CLASSES;
```

```
NAME
```

```
CLASS
```

```
-----
```

```
SAM
```

```
WRITING
```

```
SAM
```

```
ECONOMICS
```

```
BILL
```

```
ART
```

```
BILL
```

```
CHEMISTRY
```

```
DANE
```

```
MACROECONOMICS
```

```
DANE
```

```
FRENCH
```

```
JEFF
```

```
MANUFACTURING
```

7 records selected.

```
UFI> spool off
```

This is transformed into the KOREL format:

**((NAME CLASS) (SAM WRITING) (SAM ECONOMICS) (BILL ART)
(BILL CHEMISTRY) (DANE MACROECONOMICS) (DANE FRENCH)
(JEFF MANUFACTURING))**

And here is the answer:

**((NAME CLASS) (SAM WRITING) (SAM ECONOMICS) (DANE MACROECONOMICS)
(DANE FRENCH))**

8.5 Simple Mapping Example

What's the total annual expenditure for salaries?

Here, we must operate on the data to get the information we require. The annual salary for a professor is twelve times their associated salary, the annual salary for a ta is nine times their salary, and the annual salary for staff is 26 times their salary.

This is done by building a query for each salary. Thus, there is a slot in the EMPLOYEES object which can retrieve the total salary for professors, another which can get the total salary for tas, and a third which can access the database for the total salary for staff members. Finally, a slot called TOTAL-ANNUAL-SALARY is defined which will add the results of the other three attributes and add together their results. Using similar mechanisms, arbitrarily complex mappings can be provided to the user of KOREL objects.

We begin by defining TOTAL-PROFESSOR-SALARY.

* (build)

This is the BUILD introductory screen.

```

BBBB  U  U  IIIII  L      DDD
B  B  U  U      I  L      D  D
BBBB  U  U      I  L      D  DD
B  B  U  U      I  L      D  D
BBBB      UU  IIIII  LLLLL  DDD

```

Copyright (C) 1986, 1987 by Samuel P. Levine

BUILD allows you to define or modify the abstract database

queries that are associated with KOREL objects.

Before using BUILD, a KOREL object model and its objects must already exist. If not, exit BUILD at this point, and use KOREL to build them.

Press any key to begin, or <BREAK> to exit.

Here is a list of KOREL object models that currently exist.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. UNIVERSITY

Choose an existing Object model.
Selected so far: NIL

Your choice: 1

Here is a list of the objects in the UNIVERSITY object model.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. PEOPLE
2. THINGS
3. EMPLOYEES
4. STUDENTS
5. PROFESSORS
6. STAFF
7. TAS
8. CLASSES

Choose an object to examine.

Selected so far: NIL

Your choice: 3

Attach queries at the (O)bject or (S)lot level? S

Here is a list of EMPLOYEES's attributes.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. CREATE-NEW-SLOT

2. SALARY

Choose the slot to attach the query to:

Selected so far: NIL

Your choice: 1

Enter the name of a new slot to store it in: total-professor-salary

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query

2. JOIN multi-queries

3. CONCAT multi-queries

4. NO_NULLS multi-query
 5. AVERAGE multi-query
 6. SUM multi-query
 7. MINIMUM multi-query
 8. MAXIMUM multi-query
 9. CARDINALITY multi-query
- Choose the type of multi-query
Your choice: 6
-

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
 2. JOIN multi-queries
 3. CONCAT multi-queries
 4. NO_NULLS multi-query
 5. AVERAGE multi-query
 6. SUM multi-query
 7. MINIMUM multi-query
 8. MAXIMUM multi-query
 9. CARDINALITY multi-query
- Choose the type of multi-query
Your choice: 1
-

Defining a single-query

Here is a list of DBMSs that it is possible
to communicate with.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. ORACLE

Select the DBMS to send the single-query to.

Selected so far: NIL

Your choice: 1

Getting all tables from the database
to select one which contains data about the object.

If there are multiple, then define a multi-query.
I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select TNAME from CATALOG;
```

```
spool off
```

Here is the table that was returned:

```
UFI> select TNAME from CATALOG;
```

```
TNAME
```

```
-----
STUDENTS
CURRENT_CLASSES
MANAGEMENT_CLASS_HISTORY
COMPUTER_SCIENCE_CLASS_HISTORY
EMPLOYEES
```

```
UFI> spool off
```

This is transformed into the KOREL format:

```
((TNAME) (STUDENTS) (CURRENT_CLASSES) (MANAGEMENT_CLASS_HISTORY)
  (COMPUTER_SCIENCE_CLASS_HISTORY) (EMPLOYEES))
```

Choosing a selected item de-selects it.
Choose 0 to exit.

1. STUDENTS
2. CURRENT_CLASSES
3. MANAGEMENT_CLASS_HISTORY
4. COMPUTER_SCIENCE_CLASS_HISTORY
5. EMPLOYEES

Select the table which contains object data
Selected so far: NIL

Your choice: 5

Getting all columns from the current table

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select CNAME from COL where TNAME = 'EMPLOYEES';

spool off
```

Here is the table that was returned:

```
UPI> select CNAME from COL where TNAME = 'EMPLOYEES';
```

CNAME

NAME

POSITION

SALARY

MANAGER

UFI> spool off

This is transformed into the KOREL format:
((CNAME) (NAME) (POSITION) (SALARY) (MANAGER))

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return
Selected so far: NIL

Your choice: 3

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return
Selected so far: (SALARY)

Your choice: 0

Defining the select conditions for a query.

If there are multiple select conditions, they can be logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition, or no conditions at all.

Choose 0 for no action.

1. AND
2. OR
3. NOT
4. Define a Pattern
5. No Search conditions

Choose how to combine multiple patterns

Your choice: 4

This is to define a single search condition.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. =
2. >
3. <
4. >=
5. <=
6. <>

Choose a relation for a search condition

Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Choose a column
Selected so far: NIL

Your choice: 2

Choosing a selected item de-selects it.
Choose 0 to exit.

1. SPECIFY-A-LITERAL
2. NAME
3. POSITION
4. SALARY
5. MANAGER

Specify a Column or Value
Selected so far: NIL

Your choice: 1

Enter a literal value to compare to: PROFESSOR

SUM
DBMS: ORACLE
TABLE: EMPLOYEES
COLS: (SALARY)
WHERE: (= POSITION "PROFESSOR")

(SUM (ORACLE EMPLOYEES (SALARY) (= POSITION "PROFESSOR")))

Next, we run BUILD again to define the total-ta-salary.

* (build)

BBBB U U IIIII L DDD

B	B	U	U	I	L	D	D
BBBB		U	U	I	L	D	DD
B	B	U	U	I	L	D	D
BBBB		UUU		IIIII	LLLLL	DDD	

Copyright (C) 1986, 1987 by Samuel P. Levine

BUILD allows you to define or modify the abstract database queries that are associated with KOREL objects.

Before using BUILD, a KOREL object model and its objects must already exist. If not, exit BUILD at this point, and use KOREL to build them.

Press any key to begin, or <BREAK> to exit.

Here is a list of KOREL object models that currently exist.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. UNIVERSITY

Choose an existing Object model.
Selected so far: NIL

Your choice: 1

Here is a list of the objects in the UNIVERSITY object model.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. PEOPLE
2. THINGS
3. EMPLOYEES
4. STUDENTS
5. PROFESSORS
6. STAFF
7. TAS
8. CLASSES

Choose an object to examine.

Selected so far: NIL

Your choice: 3

Attach queries at the (O)bject or (S)lot level? S

Here is a list of EMPLOYEES's attributes.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. CREATE-NEW-SLOT
2. SALARY
3. TOTAL-PROFESSOR-SALARY

Choose the slot to attach the query to:

Selected so far: NIL

Your choice: 1

Enter the name of a new slot to store it in: total-ta-salary

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 6

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 1

Defining a single-query

Here is a list of DBMSs that it is possible to communicate with.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. ORACLE

Select the DBMS to send the single-query to.
Selected so far: NIL

Your choice: 1

Getting all tables from the database
to select one which contains data about the object.

If there are multiple, then define a multi-query.
I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select TNAME from CATALOG;
```

```
spool off
```

Here is the table that was returned:

```
UFI> select TNAME from CATALOG;
```

```
TNAME
```

```
-----  
STUDENTS
```

```
CURRENT_CLASSES
```

MANAGEMENT_CLASS_HISTORY
COMPUTER_SCIENCE_CLASS_HISTORY
EMPLOYEES

UFI> spool off

This is transformed into the KOREL format:

((TNAME) (STUDENTS) (CURRENT_CLASSES) (MANAGEMENT_CLASS_HISTORY)
(COMPUTER_SCIENCE_CLASS_HISTORY) (EMPLOYEES))

Choosing a selected item de-selects it.
Choose 0 to exit.

1. STUDENTS
2. CURRENT_CLASSES
3. MANAGEMENT_CLASS_HISTORY
4. COMPUTER_SCIENCE_CLASS_HISTORY
5. EMPLOYEES

Select the table which contains object data
Selected so far: NIL

Your choice: 5

Getting all columns from the current table

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select CNAME from COL where TNAME = 'EMPLOYEES';
```

spool off

Here is the table that was returned:

UFI> select CNAME from COL where TNAME = 'EMPLOYEES';

CNAME

NAME

POSITION

SALARY

MANAGER

UFI> spool off

This is transformed into the KOREL format:

((CNAME) (NAME) (POSITION) (SALARY) (MANAGER))

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return

Selected so far: NIL

Your choice: 3

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return

Selected so far: (SALARY)

Your choice: 0

Defining the select conditions for a query.

If there are multiple select conditions, they can be logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition, or no conditions at all.

Choose 0 for no action.

1. AND
2. OR
3. NOT
4. Define a Pattern
5. No Search conditions

Choose how to combine multiple patterns

Your choice: 4

This is to define a single search condition.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. =
2. >
3. <
4. >=
5. <=
6. <>

Choose a relation for a search condition

Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Choose a column
Selected so far: NIL

Your choice: 2

Choosing a selected item de-selects it.
Choose 0 to exit.

1. SPECIFY-A-LITERAL
2. NAME
3. POSITION
4. SALARY
5. MANAGER

Specify a Column or Value
Selected so far: NIL

Your choice: 1

Enter a literal value to compare to: TA

And here is the query to sum the tas' salaries:

```
SUM
DBMS:  ORACLE
TABLE:  EMPLOYEES
COLS: (SALARY)
WHERE: (= POSITION "TA")
```

(SUM (ORACLE EMPLOYEES (SALARY) (= POSITION "TA")))

Next, we define a query to return the total staffs' salary.

* (build)

BBBB	U	U	IIIII	L	DDD
B B	U	U	I	L	D D
BBBB	U	U	I	L	D DD
B B	U	U	I	L	D D
BBBB	UUU	IIIII	LLLLL	DDD	

Copyright (C) 1986, 1987 by Samuel P. Levine

BUILD allows you to define or modify the abstract database queries that are associated with KOREL objects.

Before using BUILD, a KOREL object model and its objects must already exist. If not, exit BUILD at this point, and use KOREL to build them.

Press any key to begin, or <BREAK> to exit.

Here is a list of KOREL object models that currently exist.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. UNIVERSITY

Choose an existing Object model.
Selected so far: NIL

Your choice: 1

Here is a list of the objects in the UNIVERSITY object model.

Choosing a selected item de-selects it.
Choose 0 to exit.

1. PEOPLE
2. THINGS
3. EMPLOYEES
4. STUDENTS
5. PROFESSORS
6. STAFF
7. TAS
8. CLASSES

Choose an object to examine.
Selected so far: NIL

Your choice: 3

Attach queries at the (O)bject or (S)lot level? S

Here is a list of EMPLOYEES's attributes.
Choosing a selected item de-selects it.
Choose 0 to exit.

1. CREATE-NEW-SLOT
2. SALARY
3. TOTAL-PROFESSOR-SALARY
4. TOTAL-TA-SALARY

Choose the slot to attach the query to:
Selected so far: NIL

Your choice: 1

Enter the name of a new slot to store it in: total-staff-salary

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 6

Defining a multi-query

Choose one of the following options:

Choose 0 for no action.

1. Define single-query
2. JOIN multi-queries
3. CONCAT multi-queries
4. NO_NULLS multi-query
5. AVERAGE multi-query
6. SUM multi-query
7. MINIMUM multi-query
8. MAXIMUM multi-query
9. CARDINALITY multi-query

Choose the type of multi-query

Your choice: 1

Defining a single-query

Here is a list of DBMSs that it is possible
to communicate with.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. ORACLE

Select the DBMS to send the single-query to.

Selected so far: NIL

Your choice: 1

Getting all tables from the database

to select one which contains data about the object.

If there are multiple, then define a multi-query.
I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select TNAME from CATALOG;

spool off
```

Here is the table that was returned:

```
UFI> select TNAME from CATALOG;
```

```
TNAME
-----
STUDENTS
CURRENT_CLASSES
MANAGEMENT_CLASS_HISTORY
COMPUTER_SCIENCE_CLASS_HISTORY
EMPLOYEES
```

```
UFI> spool off
```

This is transformed into the KOREL format:

```
((TNAME) (STUDENTS) (CURRENT_CLASSES) (MANAGEMENT_CLASS_HISTORY)
 (COMPUTER_SCIENCE_CLASS_HISTORY) (EMPLOYEES))
```

Choosing a selected item de-selects it.
Choose 0 to exit.

1. STUDENTS
2. CURRENT_CLASSES
3. MANAGEMENT_CLASS_HISTORY
4. COMPUTER_SCIENCE_CLASS_HISTORY
5. EMPLOYEES

Select the table which contains object data
Selected so far: NIL

Your choice: 5

Getting all columns from the current table

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select CNAME from COL where TNAME = 'EMPLOYEES';

spool off
```

Here is the table that was returned:

```
UFI> select CNAME from COL where TNAME = 'EMPLOYEES';
```

```
CNAME
```

```
-----
```

```
NAME
POSITION
SALARY
MANAGER
```

```
UFI> spool off
```

This is transformed into the KOREL format:
((CNAME) (NAME) (POSITION) (SALARY) (MANAGER))

Choosing a selected item de-selects it.
Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return

Selected so far: NIL

Your choice: 3

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Select those columns to return

Selected so far: (SALARY)

Your choice: 0

Defining the select conditions for a query.

If there are multiple select conditions, they can be logically joined using AND, OR, or NOT.

Otherwise, you can choose to specify only a single condition, or no conditions at all.

Choose 0 for no action.

1. AND
 2. OR
 3. NOT
 4. Define a Pattern
 5. No Search conditions
- Choose how to combine multiple patterns

Your choice: 4

This is to define a single search condition.

Choosing a selected item de-selects it.

Choose 0 to exit.

1. =
2. >
3. <
4. >=
5. <=
6. <>

Choose a relation for a search condition

Selected so far: NIL

Your choice: 1

Choosing a selected item de-selects it.

Choose 0 to exit.

1. NAME
2. POSITION
3. SALARY
4. MANAGER

Choose a column

Selected so far: NIL

Your choice: 2

Choosing a selected item de-selects it.

Choose 0 to exit.

1. SPECIFY-A-LITERAL
2. NAME
3. POSITION
4. SALARY
5. MANAGER

Specify a Column or Value

Selected so far: NIL

Your choice: 1

Enter a literal value to compare to: STAFF

And here is the query to get the total-staff-salary.

SUM

DBMS: ORACLE

TABLE: EMPLOYEES

COLS: (SALARY)

WHERE: (= POSITION "STAFF")

(SUM (ORACLE EMPLOYEES (SALARY) (= POSITION "STAFF")))

Finally, we define a LISP function to add the total salaries (gotten by accessing the slots we just defined) and attach that procedure to the IF-NEEDED facet of the TOTAL-ANNUAL-SALARY slot.

```
* (defun total-annual-salary (obj attr)
  (+
    (* 12 (get-object 'employees 'total-professor-salary))
    (* 9 (get-object 'employees 'total-ta-salary))
    (* 26 (get-object 'employees 'total-staff-salary))))
```

TOTAL-ANNUAL-SALARY

```
* (put-object 'employees 'total-annual-salary 'total-annual-salary
  'if-needed)
```

TOTAL-ANNUAL-SALARY

Here we show how this would be used. The user asks for the contents of the TOTAL-ANNUAL-SALARY slot. This activates the total-annual-salary procedure defined above, which scales and sums the results of other multi-queries, called by accessing the slots they are attached to.

```
* (get-object 'employees 'total-annual-salary)
I passed the following string to ORACLE
```

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select SALARY from EMPLOYEES where POSITION = 'PROFESSOR';
spool off
```

Here is the table that was returned:

```
UFI> select SALARY from EMPLOYEES where POSITION = 'PROFESSOR';
```

```
SALARY
-----
```


50000

50000

UFI> spool off

This is transformed into the KOREL format:

((SALARY) (50000) (50000))

100000

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select SALARY from EMPLOYEES where POSITION = 'TA';
spool off
```

Here is the table that was returned:

UFI> select SALARY from EMPLOYEES where POSITION = 'TA';

SALARY

900

900

UFI> spool off

This is transformed into the KOREL format:

((SALARY) (900) (900))

1800

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select SALARY from EMPLOYEES where POSITION = 'STAFF';
spool off
```

Here is the table that was returned:

```
UFI> select SALARY from EMPLOYEES where POSITION = 'STAFF';
```

SALARY

3000

3200

3000

2000

```
UFI> spool off
```

This is transformed into the KOREL format:

```
((SALARY) (3000) (3200) (3000) (2000))
```

11200

And here is the final answer, returned to the user:

1507400

8.6 Procedural Capabilities Example

Here, I show an example of the least fixed point problem, discussed earlier. By showing that this can be solved using KOREL, I can demonstrate the advantages of this system over the pure relational data model. Just as procedures can be defined to solve this problem, complex LISP procedures can be attached to KOREL objects to aid in the solution of other problems which require procedures in their solution.

Figure 21 shows the management tree at the university. To settle disputes among employees, we might want to ask a question like "Who is the lowest common manager for two employees?" This is clearly a fixed point problem, as the management tree must be searched to an arbitrary depth. While KOREL can solve this problem by using procedural attachment, it would be impossible to solve using solely relational algebra or calculus.

To find the lowest manager supervising two employees, follow the chain of management of each employee up to the top, and then compare the lists, stopping at the first name they have in common. That is what the LISP procedure `lowest-common-manager` does, which accesses the database to find out who a person's manager is.

This is one example where KOREL demonstrates some advantages over other approaches. Here, objects are not created for each employee. Instead, KOREL gets all the required information from the database. Also, only the information which is directly needed is retrieved. For example, instead of getting all the employees and their managers, the management chain is only retrieved for the employees of interest.

Sam and Butch are having an argument, and we want to see who the lowest manager is who can settle their dispute. We find this out by calling the procedure

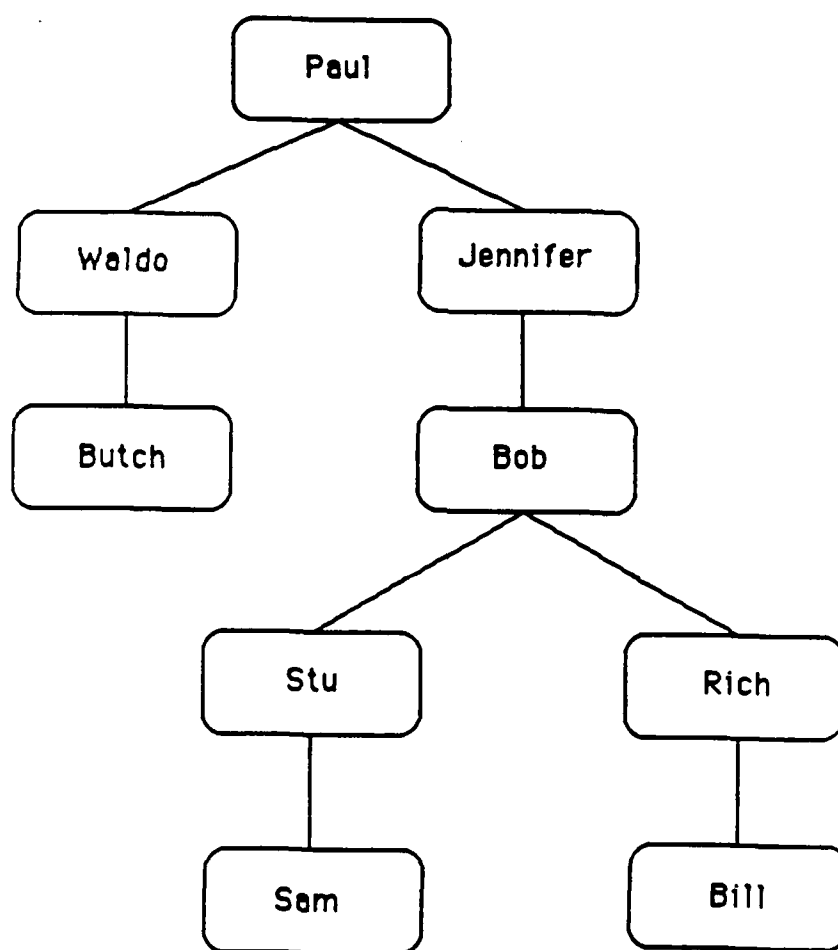


Figure 21: Management Tree at the University

lowest-common-manager.

Top-Level

* (lowest-common-manager)

This finds the lowest common manager for a pair of employees.

Enter the name of the first employee: sam

Enter the name of the second employee: butch

Begins by finding Sam's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'SAM';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'SAM';
```

```
MANAGER
```

```
-----
```

```
STU
```

```
UFI> spool off
```

This is transformed into the KOREL format:

```
((MANAGER) (STU))
```

Then, it finds Stu's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'STU';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'STU';
```

MANAGER

BOB

```
UFI> spool off
```

This is transformed into the KOREL format:
((MANAGER) (BOB))

Then it finds Bob's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'BOB';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'BOB';
```

MANAGER

JENNIFER

UFI> spool off

This is transformed into the KOREL format:

((MANAGER) (JENNIFER))

Then it finds Jennifer's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'JENNIFER';
spool off
```

Here is the table that was returned:

UFI> select manager from employees where name = 'JENNIFER';

MANAGER

PAUL

UFI> spool off

This is transformed into the KOREL format:

((MANAGER) (PAUL))

Then it tries to find Paul's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
```

```
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'PAUL';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'PAUL';
```

```
UFI> spool off
```

```
ERROR--Multi-Query Response in bad form:
NIL
```

Now it starts searching the next person. It begins by finding Butch's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'BUTCH';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'BUTCH';
```

```
MANAGER
```

```
-----
```

```
WALDO
```

```
UFI> spool off
```

This is transformed into the KOREL format:
((MANAGER) (WALDO))

Then, it tries to find Waldo's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'WALDO';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'WALDO';
```

MANAGER

PAUL

```
UFI> spool off
```

This is transformed into the KOREL format:
((MANAGER) (PAUL))

Then it tries unsuccessfully to find Paul's manager.

I passed the following string to ORACLE

```
connect dbbess/sam
set pause off
spool c:\gclisp2\monti\from-ora\from-ora.lsp
select MANAGER from EMPLOYEES where NAME = 'PAUL';
spool off
```

Here is the table that was returned:

```
UFI> select MANAGER from EMPLOYEES where NAME = 'PAUL';
```

```
UFI> spool off
```

ERROR--Multi-Query Response in bad form:

NIL

Finally, it compares the two lists to find that the lowest common manager is Paul.

(STU BOB JENNIFER PAUL) (WALDO PAUL)

The lowest common manager is: PAUL

*

8.7 Heuristic Capabilities Example

The next example I shall discuss answers the question of which students can graduate. While apparently straight-forward, this question is of such complexity that it is inappropriate for the normal procedural approach. The requirements a person must satisfy to graduate change each year, and also depend on a person's major. Thus, a procedural program would get entangled in conditional branching. However, heuristics are a clear way to embody the knowledge independently and modularly. Below are shown the rules that a person must currently satisfy in order to graduate:

Students must complete general institute requirements as well as the requirement of their department.

```
(RULE graduation-1
(IF      ((> student) has-completed institute-requirements)
          ((< student) has-completed course-requirements))
(THEN    ((< student) can-graduate yes)))
```

Institute requirements include physics, calculus, and humanities.

```
(RULE graduation-2
(IF      ((> student) entered-date fall-82)
          ((< student) has-completed physics-requirements)
          ((< student) has-completed calculus-requirements)
          ((< student) has-completed humanities-requirements))
(THEN    ((< student) has-completed institute-requirements)))
```

Recently, the institute added a writing requirement. This must be completed by all students who entered fall-83.

```
(RULE graduation-3
(IF      ((> student) entered-date fall-83)
          ((< student) has-completed writing-requirements)
          ((< student) has-completed physics-requirements)
          ((< student) has-completed calculus-requirements))
```

```
((< student) has-completed humanities-requirements))
(THEN ((< student) has-completed institute-requirements)))
```

The physics requirement is 2 terms of physics.

```
(RULE graduation-4
(IF      ((> student) has-taken physics1)
          ((< student) has-taken physics2))
(THEN   ((< student) has-completed physics-requirements)))
```

The calculus requirement is 2 terms of calculus.

```
(RULE graduation-5
(IF      ((> student) has-taken calculus1)
          ((< student) has-taken calculus2))
(THEN   ((< student) has-completed calculus-requirements)))
```

Music satisfies the humanities requirement.

```
(RULE graduation-6
(IF      ((> student) has-taken music))
(THEN   ((< student) has-completed humanities-requirements)))
```

So does art.

```
(RULE graduation-7
(IF      ((> student) has-taken art))
(THEN   ((< student) has-completed humanities-requirements)))
```

So does psychology.

```
(RULE graduation-8
(IF      ((> student) has-taken psychology))
(THEN   ((< student) has-completed humanities-requirements)))
```

To pass the writing requirement, the student must get an A or B in writing.

```
(RULE graduation-9
(IF      ((> student) aced writing))
(THEN   ((< student) has-completed writing-requirements)))
```

```
(RULE graduation-10
(IF      ((> student) beed writing))
(THEN   ((< student) has-completed writing-requirements)))
```

Managers must take accounting, finance, and entrepreneurship.

```
(RULE graduation-11
(IF      ((> student) major management)
        ((< student) has-completed accounting)
        ((< student) has-completed finance)
        ((< student) has-completed entrepreneurship))
(THEN   ((< student) has-completed course-requirements)))
```

Computer Scientists must take hardware, software, hacking, and do a thesis.

```
(RULE graduation-12
(IF      ((> student) major computers-science)
        ((< student) has-completed software)
        ((< student) has-completed hardware)
        ((< student) has-completed hacking)
        ((< student) has-completed thesis))
(THEN   ((< student) has-completed course-requirements)))
```

Before applying these rules, the user could first specify a subset of the database to examine. For example, if the user was only trying to find out if SAM could graduate, then they might first retrieve all the information about SAM's classes, and then forward chain on that.

Objects are not created for each student before forward chaining begins. During the course of answering questions, however, certain facts about students are asserted. At that time, KOREL objects are created to store those facts.

The rules are run until no more succeed. After the forward chaining is complete, those people who can graduate have an attribute CAN-GRADUATE and a value YES, whereas those people who cannot have the value nil.

We begin by running the rules associated with an object model to see which students can graduate.

We begin forward chaining. The first pass through tries to see if people have taken some basic requirements (like physics and calculus), as well as to ensure that they have completed the basic requirements for their departments.

* (forward-chain)

Trying rule GRADUATION-1...

Trying rule GRADUATION-2...

Trying rule GRADUATION-3...

Trying rule GRADUATION-4...

Rule GRADUATION-4 asserts:

(SAM HAS-COMPLETED PHYSICS-REQUIREMENTS)

because: (SAM HAS-TAKEN PHYSICS2)

(SAM HAS-TAKEN PHYSICS1)

Rule GRADUATION-4 asserts:

(BILL HAS-COMPLETED PHYSICS-REQUIREMENTS)

because: (BILL HAS-TAKEN PHYSICS2)

(BILL HAS-TAKEN PHYSICS1)

Rule GRADUATION-4 asserts:

(DANE HAS-COMPLETED PHYSICS-REQUIREMENTS)

because: (DANE HAS-TAKEN PHYSICS2)

(DANE HAS-TAKEN PHYSICS1)

Rule GRADUATION-4 asserts:

(JEFF HAS-COMPLETED PHYSICS-REQUIREMENTS)

because: (JEFF HAS-TAKEN PHYSICS2)

(JEFF HAS-TAKEN PHYSICS1)

Trying rule GRADUATION-5...

Rule GRADUATION-5 asserts:

(SAM HAS-COMPLETED CALCULUS-REQUIREMENTS)

because: (SAM HAS-TAKEN CALCULUS2)

(SAM HAS-TAKEN CALCULUS1)

Rule GRADUATION-5 asserts:

(BILL HAS-COMPLETED CALCULUS-REQUIREMENTS)

because: (BILL HAS-TAKEN CALCULUS2)

(BILL HAS-TAKEN CALCULUS1)

Rule GRADUATION-5 asserts:

(DANE HAS-COMPLETED CALCULUS-REQUIREMENTS)

```
        because: (DANE HAS-TAKEN CALCULUS2)
                  (DANE HAS-TAKEN CALCULUS1)
Rule GRADUATION-5 asserts:
(JEFF HAS-COMPLETED CALCULUS-REQUIREMENTS)
    because: (JEFF HAS-TAKEN CALCULUS2)
              (JEFF HAS-TAKEN CALCULUS1)
Trying rule GRADUATION-6...
Rule GRADUATION-6 asserts:
(SAM HAS-COMPLETED HUMANITIES-REQUIREMENTS)
    because: (SAM HAS-TAKEN MUSIC)
Rule GRADUATION-6 asserts:
(JEFF HAS-COMPLETED HUMANITIES-REQUIREMENTS)
    because: (JEFF HAS-TAKEN MUSIC)
Trying rule GRADUATION-7...
Rule GRADUATION-7 asserts:
(BILL HAS-COMPLETED HUMANITIES-REQUIREMENTS)
    because: (BILL HAS-TAKEN ART)
Rule GRADUATION-7 asserts:
(DANE HAS-COMPLETED HUMANITIES-REQUIREMENTS)
    because: (DANE HAS-TAKEN ART)
Trying rule GRADUATION-8...
Trying rule GRADUATION-9...
Rule GRADUATION-9 asserts:
(BILL HAS-COMPLETED WRITING-REQUIREMENTS)
    because: (BILL ACED WRITING)
Trying rule GRADUATION-10...
Rule GRADUATION-10 asserts:
(DANE HAS-COMPLETED WRITING-REQUIREMENTS)
    because: (DANE BEED WRITING)
Trying rule GRADUATION-11...
Rule GRADUATION-11 asserts:
(DANE HAS-COMPLETED COURSE-REQUIREMENTS)
    because: (DANE HAS-TAKEN ENTREPRENEURSHIP)
              (DANE HAS-TAKEN FINANCE)
              (DANE HAS-TAKEN ACCOUNTING)
              (DANE MAJOR MANAGEMENT)
Rule GRADUATION-11 asserts:
(JEFF HAS-COMPLETED COURSE-REQUIREMENTS)
    because: (JEFF HAS-TAKEN ENTREPRENEURSHIP)
              (JEFF HAS-TAKEN FINANCE)
              (JEFF HAS-TAKEN ACCOUNTING)
```

(JEFF MAJOR MANAGEMENT)

Trying rule GRADUATION-12...

Rule GRADUATION-12 asserts:

(SAM HAS-COMPLETED COURSE-REQUIREMENTS)

because: (SAM HAS-TAKEN THESIS)

(SAM HAS-TAKEN HACKING)

(SAM HAS-TAKEN HARDWARE)

(SAM HAS-TAKEN SOFTWARE)

(SAM MAJOR COMPUTER-SCIENCE)

I've tried all the rules; should I try again? (Y or N) Yes

The next pass checks to see if people have completed institute requirements or not

Trying rule GRADUATION-1...

Trying rule GRADUATION-2...

Rule GRADUATION-2 asserts:

(SAM HAS-COMPLETED INSTITUTE-REQUIREMENTS)

because: (SAM HAS-COMPLETED HUMANITIES-REQUIREMENTS)

(SAM HAS-COMPLETED CALCULUS-REQUIREMENTS)

(SAM HAS-COMPLETED PHYSICS-REQUIREMENTS)

(SAM ENTERED-DATE FALL-82)

Rule GRADUATION-2 asserts:

(DANE HAS-COMPLETED INSTITUTE-REQUIREMENTS)

because: (DANE HAS-COMPLETED HUMANITIES-REQUIREMENTS)

(DANE HAS-COMPLETED CALCULUS-REQUIREMENTS)

(DANE HAS-COMPLETED PHYSICS-REQUIREMENTS)

(DANE ENTERED-DATE FALL-82)

Trying rule GRADUATION-3...

Rule GRADUATION-3 asserts:

(BILL HAS-COMPLETED INSTITUTE-REQUIREMENTS)

because: (BILL HAS-COMPLETED HUMANITIES-REQUIREMENTS)

(BILL HAS-COMPLETED CALCULUS-REQUIREMENTS)

(BILL HAS-COMPLETED PHYSICS-REQUIREMENTS)

(BILL HAS-COMPLETED WRITING-REQUIREMENTS)

(BILL ENTERED-DATE FALL-83)

Trying rule GRADUATION-4...

Trying rule GRADUATION-5...

Trying rule GRADUATION-6...

Trying rule GRADUATION-7...

Trying rule GRADUATION-8...

Trying rule GRADUATION-9...

Trying rule GRADUATION-10...

Trying rule GRADUATION-11...

Trying rule GRADUATION-12...

I've tried all the rules; should I try again? (Y or N) Yes

The next pass tries to assert who can graduate.

Trying rule GRADUATION-1...

Rule GRADUATION-1 asserts:

(SAM CAN-GRADUATE YES)

because: (SAM HAS-COMPLETED COURSE-REQUIREMENTS)

(SAM HAS-COMPLETED INSTITUTE-REQUIREMENTS)

Rule GRADUATION-1 asserts:

(DANE CAN-GRADUATE YES)

because: (DANE HAS-COMPLETED COURSE-REQUIREMENTS)

(DANE HAS-COMPLETED INSTITUTE-REQUIREMENTS)

Trying rule GRADUATION-2...

Trying rule GRADUATION-3...

Trying rule GRADUATION-4...

Trying rule GRADUATION-5...

Trying rule GRADUATION-6...

Trying rule GRADUATION-7...

Trying rule GRADUATION-8...

Trying rule GRADUATION-9...

Trying rule GRADUATION-10...

Trying rule GRADUATION-11...

Trying rule GRADUATION-12...

I've tried all the rules; should I try again? (Y or N) Yes

finally, all the rules can be tried without any firing. We're finished.

```
Trying rule GRADUATION-1...
Trying rule GRADUATION-2...
Trying rule GRADUATION-3...
Trying rule GRADUATION-4...
Trying rule GRADUATION-5...
Trying rule GRADUATION-6...
Trying rule GRADUATION-7...
Trying rule GRADUATION-8...
Trying rule GRADUATION-9...
Trying rule GRADUATION-10...
Trying rule GRADUATION-11...
Trying rule GRADUATION-12...
I've tried all the rules; should I try again? (Y or N) No
Ok, I quit.
NIL
*
```

Now we can see which students can graduate or not.

```
* (get-object 'sam 'can-graduate)
YES
* (get-object 'bill 'can-graduate)
NIL
* (get-object 'dane 'can-graduate)
YES
* (get-object 'jeff 'can-graduate)
NIL
*
```

8.8 Meta-Level Capabilities Example

The final example I shall discuss is a continuation of the previous example. Once it is known which students can and which cannot graduate, we might want to delve deeper to find out why KOREL gave that answer.

This ability differs from pure procedural attachment in that the KOREL objects must be able to refer to the processes which were used to come up with a value. For this example, an answer can be given based on the knowledge that the KOREL objects have about the requirements for graduation. This sort of feature would be very difficult to implement in a purely procedural language.

The strategy for answering the above question involves displaying the rules that were used to arrive at an answer. The example below uses the KOREL *why* capability.

The value of a KOREL fact may have been created by inferencing, retrieved from the database, or stored in the object by the developer. To find out which, the user types WHY. KOREL then will say how the value of the fact was arrived at. If the fact is the result of inferencing, then the rules that were used to assert that fact would be displayed.

The user wants to know why Sam can graduate. The process involves calling the KOREL function WHY with the fact that the user is interested in.

**(WHY '(SAM CAN-GRADUATE YES))*

This first fact was derived by the inferencing chain which was shown in the previous example.

Fact (SAM CAN-GRADUATE YES) is the result of an inferencing chain.

It was asserted by the rule GRADUATION-1 because:
(SAM HAS-COMPLETED COURSE-REQUIREMENTS)
(SAM HAS-COMPLETED INSTITUTE-REQUIREMENTS)

Each of the facts which allowed rule GRADUATION-1 to fire must also be explained.

Fact (SAM HAS-COMPLETED COURSE-REQUIREMENTS) is the result of an inferencing chain.

It was asserted by the rule GRADUATION-12 because:
(SAM HAS-TAKEN THESIS)
(SAM HAS-TAKEN HACKING)
(SAM HAS-TAKEN HARDWARE)
(SAM HAS-TAKEN SOFTWARE)
(SAM MAJOR COMPUTER-SCIENCE)

Fact (SAM HAS-COMPLETED INSTITUTE-REQUIREMENTS) is the result of an inferencing chain.

It was asserted by the rule GRADUATION-2 because:
(SAM HAS-COMPLETED HUMANITIES-REQUIREMENTS)
(SAM HAS-COMPLETED CALCULUS-REQUIREMENTS)
(SAM HAS-COMPLETED PHYSICS-REQUIREMENTS)
(SAM ENTERED-DATE FALL-82)

The explanation process can stop whenever the explanation for a fact is that it was retrieved from a database or came from an object.

Fact (SAM HAS-TAKEN THESIS) retrieved from a database.

Fact (SAM HAS-TAKEN HACKING) retrieved from a database.

Fact (SAM HAS-TAKEN HARDWARE) retrieved from a database.

Fact (SAM HAS-TAKEN SOFTWARE) retrieved from a database.

Fact (SAM MAJOR COMPUTER-SCIENCE) retrieved from a database.

Fact (SAM HAS-COMPLETED HUMANITIES-REQUIREMENTS) is the result of an inferencing chain.

It was asserted by rule GRADUATION-6 because:
(SAM HAS-TAKEN MUSIC)

Fact (SAM HAS-COMPLETED CALCULUS-REQUIREMENTS) is the result of an inferencing chain.

It was asserted by rule GRADUATION-5 because:
(SAM HAS-TAKEN CALCULUS2)
(SAM HAS-TAKEN CALCULUS1)

Fact (SAM HAS-COMPLETED PHYSICS-REQUIREMENTS) is the result of an inferencing chain.

It was asserted by rule GRADUATION-4 because:
(SAM HAS-TAKEN PHYSICS2)
(SAM HAS-TAKEN PHYSICS1)

Fact (SAM ENTERED-DATE FALL-82) retrieved from a database.

Fact (SAM HAS-TAKEN MUSIC) retrieved from a database.

Fact (SAM HAS-TAKEN CALCULUS2) retrieved from a database.

Fact (SAM HAS-TAKEN CALCULUS1) retrieved from a database.

Fact (SAM HAS-TAKEN PHYSICS2) retrieved from a database.

Fact (SAM HAS-TAKEN PHYSICS1) retrieved from a database.

9 Conclusions

9.1 Summary of the Main Points

- I began by proposing an object model as a more useful and natural model of the real world than, for example, a relational data model. Then, working with this as a premise, there are four general ways that objects and databases can be combined: extending objects, extending databases, KBMS, and translating between the two. In cases where there are large, shared, existing databases which contain information to be used, then the translating approach is superior, as it allows the creation of object systems which use the existing databases without requiring them to be modified.
- Once having selected the translating approach, I made several other design decisions. One which is only appropriate if the database changes frequently or when there is a high penalty for using non-current data is to discard the data retrieved from the database instead of storing it in the objects. This helps minimize the problem of skew.
- To perform the translation, I designed an abstract interface language to a DBMS (called ADBMS). Once a translating program has been written between this language and a DBMS, then an abstract query can be automatically translated into an actual query. This allows naive users to formulate queries without knowing the specific DBMS query language. Queries can also return information from the database and information about the database (from the data dictionary). ADBMS also allows queries to multiple DBMSs. It can combine the results from multiple databases by relational joining or concatenation, two useful ways of combining information which might be stored in using different DBMSs or on different

machines.

- I designed and implemented an object-oriented representation language that runs in Gold Hill Common Lisp on an IBM PC/AT, as well as a translator for the ORACLE DBMS on the IBM PC.
- Finally, I demonstrated the implementation and feasibility of this approach by building and using an object model of a university's databases.

9.2 Tradeoffs

- The translating approach allows the use of existing multi-user databases, instead of changing the database schemata or creating a new database. This isn't necessary or appropriate in cases where there is no current database, or the database is only used for this one application. In these situations, it might be more efficient to take use an ODBS or a KBMS and translate the existing data, if any.
- This approach facilitates the use of data by interactively defining queries using the data dictionary, instead of expecting the user to know its contents.
- The ADBMS allows the use of data from multiple DBMSs.
- Once the translating object is written for a DBMS, it can be used by other applications needing data in that database, instead of specifically hard-wiring the interface for an application. This means that there is more mechanism required for single applications. For example, if I just wanted to perform one retrieval numerous times, then it would be quicker to just write a procedure which just connects to the database and performs that query. The ADBMS approach requires a lot more effort before use, in creating an actual-DBMS module to translate ADBMS queries. This generality may not be needed in all cases, but in cases where the use of a database is not greatly constrained, the ADBMS approach simplifies the definition of queries.
- The developer doesn't have to learn each DBMS and its connecting procedure, only the ADBMS, because of the translator for each DBMS. Of course, the developer must learn the ADBMS. Additionally, the developer

must still have some knowledge about what the database contains: eg: what the columns mean, and which ones should be returned to answer a question.

- This approach minimizes the chance of using old data, because it doesn't store the data in the objects as well as in the database. In situations where the data doesn't change much and it is costly to connect to a database, or where there is only a small penalty for using old data, then this approach is less efficient. One could instead load the entire database into objects at the beginning of a session, and use them as needed.
- This approach only gets the data that is needed instead of getting an entire table or database. When the cost of executing a query is high, then it might be more efficient to bring in and store more information than is requested in the hope that it will be requested later.
- The object model is easy for users to learn to use as opposed to the data model. However, many people who are currently familiar with databases would have to learn a new view of data.
- The system runs on an IBM PC/AT. This might increase the difficulty of connecting to other computers.
- This approach doesn't fully use DBMS capabilities. Can't store or modify data using current protocol, which might be needed by application.

9.3 Further Research

- Currently, the multi-query can only return information. Oftentimes, it is useful for changes to objects to be reflected to the underlying database. This capability would require extending multi-queries to allow the deletion and modification of database data. An additional benefit of this extension would be the ability to add permanence to objects.
- The KOREL/database interface could be made much more efficient if the breakdown of a multi-query into single queries used more of the capabilities of the DBMSs involved. For example, if a relational join were to be performed upon two tables (or upon the same table) in the same database, then it is much more efficient to allow that DBMS to perform the join than for KOREL to do it.
- Another research issue involves providing arguments to query messages to make them more specific. Hence, *general-queries* which allow variables could be defined. For example, here is a general-query which takes an instance name, and will return data about that instance.

(general-query (instance-name)

(ORACLE (employees) (ALL) (= NAME instance-name)))

This capability is currently provided in KOREL, but it should be made available for other applications which might be built upon the ADBMS.

- Many efficiency benefits could be obtained by adding a way to represent the cost of re-executing a query vs. some measure of the permanence of the returned values and the cost of using old data. Thus, if the cost of

executing a query were great (because a lot of values are returned) and the data is stable (ie: it doesn't change much), then it might be more efficient to store data locally in the object instead of re-executing the query the next time the data is needed.

9.4 Vision for the Future

There is little disagreement that there is currently a vast amount of information which is stored in databases. In addition, due to inertia, people will tend to keep using the tried and true tools of the past, so that databases will keep on growing for some time to come. It all comes down to a question of which is the best way to get that information out so that people can use it.

Ignoring the problem by concentrating on the tools of the future won't make it go away. That, I think, is the fatal flaw in systems which can't easily access current databases, like the research which is being conducted in object-oriented databases.

Similarly, it is too narrow to constrain the solution to only one DBMS. Information has many sources. You can't answer complex questions by only providing the information in dribs and drabs. The solution is to have a system that's smart enough to provide facilities for combining data from various sources. Such a system could truly provide insight into a difficult problem, instead of only providing the scrambled pieces of a jigsaw puzzle.

Also misguided are the approaches which concentrate on solving the problems as they arise. General tools like this one can ignore the exponential explosion caused by trying to anticipate each question which might be asked of a system and encoding the way to solve it in a procedure. While a common interface like the one described here might not be needed for the specific task at hand, it does simplify the solution of future tasks which use that same DBMS. Once the work to build a translator is done, it is general enough to be re-used.

The database language which is defined here is minimal. But because it is so small, it simplifies the construction of the translator, without sacrificing the power of the DBMS. While using only a subset of a data language causes some

efficiency to be lost, it is still an unanswered question as to how much actually is sacrificed.

In addition, it is clear that as new database products arrive, it will be important to be able to access them from existing systems. The system which I describe here can act as a bridge between currently popular systems like IDBS, state-of-the-art databases like ORACLE, and the databases of the future.

The problem attacked here is insidious and growing. As the amount of data which can be usefully combined grows, the pressure to come up with a good framework to combine it builds exponentially. This report suggests one approach which can allow the integration of existing and future databases. Among the myriad benefits of this approach are better decision making and better control capabilities, vital for today's diversified business environments.

10 References

- [AHO79] Aho, A.V., Ullman, J.D., "Universality of Data Retrieval Languages", Sixth Annual ACM Symposium on Principles of Programming Languages, January 1979, pp 110-120.
- [BAN86] Banerjee, J., Kim, W., Kim, H., "Framework for Schema Evolution in an Object-Oriented Database", MCC technical report DB-250-86, 1986.
- [BER85] Berghel, H.L., "Simplified Integration of PROLOG with RDBMS", Data Base, Spring 1985, pp 3-12.
- [BRA84] Brachman, R.J., Levesque, H.J., "What Makes a Knowledge Base Knowledgeable? A View of Databases from the Knowledge Level", Proceedings of the First International Workshop on Expert Database Systems, 1984.
- [JAR83] Jarke, M., Vassiliou, Y., "Coupling Expert Systems with Database Management Systems", Artificial Intelligence Applications for Business, 1983, pp 65-85.
- [KEL82] Kellogg, C., "Knowledge Management: A Practical Amalgam of Knowledge and Data Base Technology", Proceedings of the AAAI-82, 1982, pp 306-9.
- [LAF83] Lafue, G.M.E., "Basic Decisions about Linking an Expert System with a DBMS: a Case Study", IEEE Database Engineering, December 1983.
- [MER83] Meersman, R., Van Assche, F., "Modeling and Manipulating Production Data Bases in Terms of Semantic Nets", IJCAI-83, 1983, pp 325-9.
- [NGU86] Nguyen, G.T., "Object Prototypes and Database Samples for Expert Database Systems", First Workshop on Expert Database Systems, 1986, pp 3-14.
- [NIJ83] Nijessen, G.M., "From Databases Towards Knowledge Bases", DBMS-A Technical Comparison, 1983, pp 113-131.
- [PIR79] Pirotte, A., "Fundamental and Secondary Issues in the Design of Non-Procedural Relational Languages", IEEE, 1979, pp 239-250.
- [STE86] Stefik, M., Bobrow, D.G., "Object-Oriented Programming: Themes and Variations", AI Magazine, pp 40-62, Winter 1986.
- [SU85] Su, S.Y.W., Raschid, L., "Incorporating Knowledge Rules in a Semantic Data Model: an Approach to Integrated Knowledge Management", IEEE AI Applications Conference, 1985.
- [VAS83] Vassiliou, Y., et al, "How does an Expert System Get Its Data?", Very Large Data Bases Proceedings-83, 1983, pp 70-72.

[WAL83] Walker, A., "Databases, Expert Systems, and PROLOG", *Artificial Intelligence Applications for Business*, 1983, pp 87-109.

[WIE84a] Wiederhold, G., "Knowledge and Database Management", *IEEE Software*, January 1984, pp 63-73.

[WIE84b] Wiederhold, G., "Knowledge Bases", *Fifth Generation and Super Computers*, 1984, pp 223-235.

[WIN84] Winston, P.H., Horn, K.P.H., *LISP*, second edition, 1984.

A KOREL Specification

The following notational conventions are used throughout this Appendix:

square brackets [] enclose optional arguments.

braces { } enclose arguments which may be repeated ad infinitum.

parenthesis () signify choosing one of the enclosed possibilities.

PUT-OBJECT object slot value [facet]

RETURNS: value

DESCRIPTION: puts the value into facet of the slot. The VALUE facet is the default.

GET-OBJECT object slot [facet, ALL]

RETURNS: value

DESCRIPTION: If facet, returns the contents of that facet. If unspecified, returns the contents of the VALUE, DEFAULT, IF-NEEDED, and QUERY facets. If still no value, repeats this value finding procedure with each of the object's superiors. If ALL, then returns the value in the slot, and follows the inheritance appropriately up or down to return each found value.

For example, (get-object object 'superiors 'all) would return each of the object's superiors, and its superiors' superiors, etc. (get-object object 'instances 'all) would return each of the object's instances, and each of its sub-ordinates' instances etc.

REMOVE-OBJECT object slot value [facet]

RETURNS: (t, nil)

DESCRIPTION: Removes the specified value from the specified fact. VALUE is the default facet. Returns t if value was successfully removed.

MAKE-OBJECT object {slot value [facet]}

RETURNS: t

DESCRIPTION: Creates a new object. This command is equivalent to doing a (put-object slot value [facet]) for each slot-value-facet triple. If facet is not specified, VALUE is used as the default.

CREATE-INSTANCE class-name instance-name

RETURNS: (instance, nil)

DESCRIPTION: Creates a new instance of the class. If the object in the class-name is an instance, then an error is signalled, and nothing is created (nil is returned). Otherwise, the created instance is returned.

SEND-MESSAGE object message [arguments]

DESCRIPTION: Send the message and arguments to the object. If the message is not handled there, the message and arguments propagate up the inheritance hierarchy. The results of handling the message are returned, or nil if the message is not handled.

PRINT-OBJECT object

DESCRIPTION: Prints all the non-internal attributes and facets. (ie: SUPERIORS, SUBORDINATES, and INSTANCES slots not printed).

DISPLAY-CLASSES

DESCRIPTION: prints the inheritance hierarchy, beginning at the CLASS class. Doesn't show instances.

REMOVE-CLASSES [class]

RETURNS: (t, nil)

DESCRIPTION: Deletes the specified class (the CLASS class is the default) and all those classes and instances which are subordinate to it in the inheritance hierarchy.

SELECT-QUERY object

DESCRIPTION: Prints each of the queries of an object and prompts the user to select one. When one is chosen, it is sent to the ADBMS. The results of the multi-query are returned, or nil if none are selected.

USE-RULES object [slot]

DESCRIPTION: Forward chains the rules in the RULES slot of the object, or if a slot is specified, then those rules in that slot's RULES facet. The last asserted fact is returned, or nil if no rules fire.

Note: since facts are of the form (object attribute value), then they are automatically stored into KOREL objects as they are asserted.

B Specification of KOREL Facets

The following notational conventions are used throughout this Appendix:

braces { } describe the acceptable contents of each facet.

The following are the only allowable KOREL facets. Attempting to access a facet other than one of those listed below will result in an error, and the requested operation will not be completed.

- **VALUE {values}**: This specifies a value for the slot. Returns single or multiple, depending on **MULTIPLE-VALUE-F**.
- **DEFAULT {values}**: This specifies a value as a default, to be used only if there is no associated **VALUE**. Returns single or multiple, depending on **MULTIPLE-VALUE-F**.
- **IF-NEEDED {procedure-names}**: A list of procedures to try if there is no value or default values for this slot.
- **IF-ADDED {procedure-names}**: A list of demons which are executed whenever a value is placed into the value facet of this slot.
- **IF-REMOVED {procedure-names}**: A list of demons which are executed whenever a value is removed from the value facet of this slot.
- **VALUE-TYPE {integer, string, fraction, real}**: Whenever a value is placed into value, default, or choices facets, it is first checked to ensure that it is of the appropriate value type.

A value type of *nil* means that there are no type restrictions on the value of this slot.

- **CHOICES {values}**: An enumeration of the acceptable values for this slot. Checked whenever a value is placed into value or default facets.
- **SELF-CONSTRAINTS {constraints}**: Whenever a value is placed into the value or default facets, this set of constraints is checked to ensure that none are violated.

Whenever a new constraint is added to this list, the choices, value, and default facets are checked to ensure consistency. Inconsistent defaults and choices are eliminated, while the user is asked how to resolve inconsistent values.

Constraints are lisp procedures which take three arguments: frame, slot, and value.

- **MULTIPLE-VALUE-F {t, nil}**: If t, signifies that the slot accepts multiple values and defaults. Default is nil (ie: single-valued).
- **QUERY {multi-query}**: If there is no value or default value for this slot, and the if-needed demons don't return a value either, then this multi-query is executed in an attempt to get a value for this slot from a database.
- **RULES {rules}**: Contains multiple rules.

C Specification of ADBMS Grouping Operations

Each of these is applied to a *list of elements*.

- **NO-NULLS**: returns list with null elements filtered out.
- **AVERAGE**: returns $SUM(list)$ divided by $CARDINALITY(list)$. List must be numeric.
- **SUM**: returns the summation of the elements. List must be numeric.
- **MINIMUM**: returns the numerically smallest value in list. List must be numeric.
- **MAXIMUM**: returns the numerically largest value in list. List must be numeric.
- **CARDINALITY**: returns the number of elements in list.

A KNOWLEDGE-BASED SYSTEM FOR RESOLVING SEMANTIC CONFLICTS: A PROBLEM OF INTEGRATING HETEROGENEOUS DATABASE MANAGEMENT SYSTEMS

MARINA POCATERRA SILVA

Full integration of heterogeneous database management systems involves three distinct objectives. There are:

- (a) Ability to access heterogeneous database management systems;
- (b) Ability to recognize and to resolve underlying semantic conflicts; and
- (c) Ability to present a reconciled and sensible, integrated version of all the pieces of data.

Objective (a) is relatively simple to achieve, as compared to the other two objectives. The latter two objectives can be potentially achieved by using Knowledge-Based Systems (KBS) methodologies.

Semantic issues deal with the problems of meaning that appear in the context of heterogeneous database management systems. Semantic mismatches exist in any context where different people have been involved in the definition of data models. These arise from the fact that each person or group makes a particular set of assumptions, interests, and needs which differs from the set of assumptions made by other persons and groups. This leads to the fact that whenever one tries to combine information from different database management systems, a wide range of semantic conflicts invariably arise. The various types of semantic conflicts are as follows:

- (a) Objective Conflicts: This group of conflicts involves differences in concepts. Examples are:
 - (i) Syntax -- differences in wording; and
 - (ii) Ambiguity -- differences in meaning.
- (b) Subjective Conflicts: This group of conflicts involves differences in interpretation of data. Examples are:
 - (i) Contradiction; and
 - (ii) Incompleteness.

The above set of conflicts are solved by using a combination of several different problem-solving strategies. These include:

- (a) Comparison of evidence;
- (b) Analysis of credibility of sources;
- (c) Search for complementary data; and
- (d) Inferencing from similar data available elsewhere.

A prototype system has been implemented for a particular scenario. The intelligence of this is located in the places where non-trivial decisions have to occur. By distributing knowledge and reasoning capabilities across various parts of the system, such as system manager, application manager, and user interface, it has been possible to design a powerful system that supports full integration of information within the context of the experimental scenario.

1. INTRODUCTION AND OVERVIEW.

1.1. Background.

Advances in networking are producing increasing interest among users for accessing data from distributed heterogeneous DBMS. For instance, a user might want to search for products' features through several electronic shopping guides, check several airlines for the appropriate flight, or search for a hotel through different on-line tourist guides. Certainly, large organizations need to access information from their different departments. The problem is that even when the heterogeneous data is semantically related, the underlying differences in data models and access methods are highly significant.

Databases are different because they are independently created to accommodate particular needs. Even though databases share the same data models, they usually present mutual semantic conflicts -incompatibilities with respect to the meaning of 'similar' data. These conflicts result from different perceptions of the same reality by different people. Should a user be expected to learn about each database he wants to access? In this study we address these issues, and offer a new perspective to resolve semantic conflicts in heterogeneous database management systems.

Many approaches have been proposed for integrating heterogeneous DBMS. They usually differ in the degree of integration pursued. We define full integration of

heterogeneous DBMSs as one that provides a common view and common access to the underlying databases. Common view, in this case, should go as far as to provide the user with a reconciled version of the underlying differences of opinion between the various databases. A fully integrated system must be able, in addition, to provide information from the combination of different sources of data.

This methodology for integration, its problems and solutions are the main topic of this thesis

Naturally, if databases are homogeneous both in content and structure, no problem exists. The constraint imposed is that the solution must preserve the autonomy of the local DBMSs. If this premise is not respected the scope of system would be very narrow. We want to maintain an open environment where many different systems can access the same databases without interfering with each other's view of them.

1.2 Semantic Conflicts.

Integration of heterogeneous databases will inevitably raise several semantic conflicts. A system that expects to accomplish this integration will have to address this problem effectively.

Dealing with semantics immediately suggests ambiguity and uncertainty, characteristics that are not easily dealt with by traditional computer programming techniques, which require very well-specified, unambiguous problems. This thesis proposes a different framework to approach this kind of integration. We apply

Artificial Intelligence technology, in particular Knowledge-Based Systems methodologies to design integrative systems. The inference engines of KBSs provide the necessary capabilities of implementing the sets of heuristics needed to solve semantic conflicts. The expected final product is a system capable of drawing sensible conclusions and even capable of making inferences about information that is not explicitly present in any database. Our experiments with a case study dealing with the integration of three sources of information about hotels in Paris show that our approach offers a suitable and powerful way of solving the semantic conflicts.

The organization of this thesis is as follows. The second Chapter introduces the problem of integrating heterogeneous DBMSs. In the third Chapter, a detailed explanation of the problem of Semantic Conflicts in heterogeneous DBMSs is presented. Chapter 3 also addresses alternative problem-solving strategies for solving Semantic Conflicts. As a result, four general strategies are presented as ways of solving subjective conflicts. Next, Chapter 4 describes the proposed framework, which is based on the application of Knowledge-Based Systems methodologies. Chapter 5 presents a prototype implementation using the framework, to show the features, requirements and technologies that would be necessary in a real-world problem. The object-oriented approach offers especial advantages from a design perspective, because of its transparent structure and organization. Finally, the last chapter presents some conclusions, including a summary of the features of the proposed framework for hosting integrative systems, and directions for further work.

2. INTEGRATING DISTRIBUTED HETEROGENEOUS DBMSs.

Advances in electronic technology have resulted in a proliferation of computer systems that gives to a great number of people the possibility of having some kind of computer equipment to their disposal.

Advances in communication technology have made it possible to interconnect several different computer systems through the same network.

The value of communicating with many different computers is even greater when we consider the quantity of users and hence the variety of data and applications that can be shared among them [Kimb:81].

Databases are important examples of pieces of information that might generate interest among different parties. For instance, there are various companies that record information about energy issues, like oil prices, oil production per country, etc.. As expected, any individual whose work relates to these issues will find it strategically important to access this type of databases. In the context of large organizations, it will be important, in terms of managerial control, to have access to the various database of the different divisions or departments. These databases, in many cases, are independently created and maintained by each division, and they do not necessarily use the same computer systems.

Furthermore, as communication and integrative technology progresses, more and more common sources of information will go on-line. Examples of these are newspapers, books, market analyses, all kind of guides -tourist, technical, medical.

Given that the physical possibility -network connection- exists to access databases at different locations, it will be interesting to analyze which are other parameters involved that may in turn difficult the real exploitation of such facility. The objective is to see how feasible is to actually make use of this power.

To access databases through a network will probably imply accessing different kinds of DBMSs. As there are several distinct DBMSs in the market, it is very likely that different organizations, using all kind of computer environments, will be using different DBMSs.

2.1. Inconveniences in Accessing Different Databases.

The fact is that differences among databases are so many that, access to any particular one demands from the user special knowledge about the database. The differences can be attributed, first, to the computer and DBMS environment in which the database has been implemented (See [Glig:86]). Second, and perhaps more subtle, to the particular data model used in defining the associated data.

The former involves learning the actual method of access, both to the computer system and to the specific DBMS. The second involves identifying and understanding the set of assumptions made in creating the particular data model.

These issues have to be carefully considered as they may act as deterrents for people wanting to exploit the power of accessing different sources of information. A detailed explanation of these issues follows.

2.1.1. Different Database Languages

Database languages (DBL) are provided by the DBMSs as the mechanisms for the user to interact with the data stored in the databases. They can be divided in two parts: Data Definition Language and Data Manipulation Language [Date, 1986]. The former is used to define the data itself -data structures, attributes and relationships between them. The latter is used to manipulate the data. It includes functions for adding, updating and retrieving data from the database.

These languages are completely dependent of the Database Management System used. For example IMS, DBase3, Oracle will each have a particular DBL of their own. They are usually very different from each other¹. The degree of difficulty in utilizing them vary from DBMS to DBMS. In any case, this implies that anyone wanting to access a database has to start by learning the associated DBL. The important point is that even when the task involved is not very difficult -many times it is,- there is certainly a period of time involved in training

2.1.2. Different Data-Models

Data models constitute the conceptual or abstract representation of the information contained in a database. Data models reflect the way in which users perceive the set of information stored in a database.

The data models are defined depending on the particular needs, interests and assumptions of a group of people [Borg:86]. The process of generating a database model reflects a specific mental model. This is particularly important when we are to compared the data models generated for different databases. The underlying

¹ "Some kind" of similarity, however, can be found within databases that use the same data model, namely relational, network, and hierarchical.

assumptions made by two human beings -even when they belong to the same organization but i.e. to different departments,- are likely to vary. The least they will vary is in the name of the fields used. For example, somebody may name a field as "Part.#" while another person in another division may refer to the same field as "Component.#".

Bigger differences will occur when issues of meaning are involved. This occurs when the various people mean different things even when referring to the same terms. For example, two different persons or groups may have a database field named "price" or "salary", however, the actual meaning for each one may involve completely different concepts. If we combine the differences that can be found among data models, any integration of data models seem like a non trivial task.

This happens because each person or group has its own way of doing things, of solving problems, of representing objects, its own "jargon". If this type of differences occur even within the same organization or within a similar group of people. Can we anticipate the magnitude of the differences that will occur between database models developed by different organizations or by unrelated groups of people?

These conflicts of meaning are present at all levels in almost any context. They could be said to be inherent to the human being, its mental model and reasoning process.

This type of problem shows that, even if DBLs were the same, the conflicts produced by differences in semantics between the DBMSs are tremendous.

Without knowing the real significance of the data, it is very difficult to place questions and thus, get useful answers from the heterogeneous system.

A person that wants to overcome these problems, and thus be able to compare data provided by different sources effectively, has to take the time to understand the corresponding criteria of each local database. Specifically, he will have to determine whether a figure is expressed in dollars or francs; whether the prices include tax or not; whether the # of barrels per day include only heavy or all type of oil crude. This is a time consuming and many times painful task that may easily act as deterrent, preventing casual users from using external DBMSs as complementary sources of information.

2.2. Possible Solutions

Certainly, many people have expressed the need for systems that facilitate access to different DBMSs ([Kimb:81], [Glig:83], [Smit:86]). The idea is to devise a system that could mask from the user the underlying differences between the DBMSs. Different approaches have been proposed which vary in the degree of integration assumed. Some support solutions to mask, from the user, the differences in accessing methods to heterogeneous DBMSs. This involves, in general, defining a global schema together with a global DBL that can be used by a user to communicate with any of the available DBMS. This does not imply integration among different data from the various DBMSs.

Others propose more ambitious solutions that could support some combination - at least some composition- of fields between heterogeneous DBMSs which hold similar data models. Each of these is explained in detail in the following sections.

2.2.1. Uniform Method of accessing different DBMSs.

The main objective within this approach is to design a common way of access between the user and the several DBMSs. As mentioned before, there is no attempt to mix data from the various DBMSs.

Some advocates of this solution focus on the design of a single DBL that will in turn translate a request into the corresponding DBLs of each participant DBMS [Jark:83]. This is a very popular approach. However, rather complicated given that it involves, in most cases, designing a very general translator -- parser -- able to perform mappings into many different local DBLs. In any case, it would facilitate the user's task who will only have to learn one DBL in order to access the different DBMSs².

Other group has focused in the design of natural language interfaces [Spar:84]. Unfortunately, the technology is not ready yet as to provide reliable interfaces. Even if it were ready, this approach has a problem. Allowing the user to input natural languages commands gives too much freedom to the user to ask unbounded questions. This demands a very intelligent system able to handle a great variety of requests.

Another approach is to use a menu-driven model in which the system establishes a dialog with the user by providing different options according to previous answers. The advantage of this method with respect to the user is that it is straight forward to use. In terms of the system, it has the advantage of keeping permanent control of the possible set of requests. The translation of information between the system and the local DBMSs is simpler than in the previous cases. As the language is not formal the need for a complicated parser is eliminated.

² This is assuming that this is the only domain in which we will be interested. Any other domain will again require him to learn about the associated DBLs.

2.2.2. Combination of Data Between Different DBMSs

The objective of this solution is to allow the user to mix data from different DBMSs ([Kimb:81], [Nava:86]). This solution is a super-set of the previous one as it involves common access method thus, global schema, plus some especial capabilities to interpret heterogeneous data. This solution would involve reconciling more subtle differences like Semantics between different DBMSs. The problem is that these differences are by no means easy to solve.

Depending on the degree of manipulation expected from the system -- flexibility for combining the different data -- this could be both a very powerful and difficult solution. Extensive explanation of the problems involved in reconciliation of data between different DBMS, both homogeneous and heterogeneous is given in Chapter 3.

2.3. Alternative Approaches for Integrating Different DBMSs.

The advantages of these unifying solutions are many. Just by assuming a common access method to the various DBMSs, the time saving considerations become tremendous. A person would have to engage in learning only one DBL -if any. Even more benefits could be gained from integrating the data from different sources of information in an easy way. Two possible alternatives are discussed in the following sub-sections.

2.3.1 Standardization

The easiest approach to achieve these goals would be to propose some kind of standardization methods. This will eliminate, at least, the problem of dealing with several DBLs. It is possible that industries at some point accept some kind of

standardization processes - as in the case of SQL []. However, the proliferation of new products is a strong forcing function for maintaining heterogeneity. This means that any attempt of standardization will be permanently threatened by the appearance of new and better products.

Moreover, an attempt for standardization will imply that existing DBMSs will have to either be redesigned or discontinued. This is an enterprise that most people will not even consider.

In any case, this standardization would only apply to the first solution, namely, defining common access methods. No standardization is possible with respect to the semantics of each database. Each person or group has its own way of regarding data needs and these will be extremely difficult to reconcile. For these reasons it is unfeasible to expect that standardization will solve the problem of accessing different DBMSs.

2.3.2. A System that Handles Heterogeneity

A much more feasible and plausible alternative would be to acknowledge the natural existence of heterogeneity and rather work towards designing a system that deals effectively with this fact of life.

The advantages of an alternative like this is that it opens a real possibility of easy access to several pre-existing DBMSs with little or none disruption on the existing environment [Daya:84].

The disadvantage is the complexity of the task to solve and the need of an appropriate technology [Kimb:81]. One of the most important obstacles that will

unavoidably arise in integrating heterogeneous and even homogeneous DBMSs is the problem of Semantic Conflicts.

This thesis will study this problem carefully. The following chapter will develop this issue extensively.

3. SEMANTIC ISSUES.

Semantic issues deal with the problems of meaning that appear in the context of heterogeneous DBMS's. Semantic mismatches exist in any context where different people have been involved in the definition of data models. These arise from the fact that each person or group has a particular set of assumptions, interests, and needs about certain issues that lead them to define data models in a likely unique way. This also applies to the same person or group defining models at different points in time. This happens because interests and assumption evolve. Each time a person redesigns a model certain changes will be made. For these reasons, we expect that whenever one tries to combine information from different DBMSs, a wide range of semantic conflicts will arise.

To use different databases, which have been developed at different times and using different criteria, will demand a clear understanding of the assumptions associated with the database we are interested in using.

3.1 Different Semantic Problems.

Certainly, any system that attempts to do a comprehensive integration of heterogeneous DBMSs will encounter many difficulties in terms of semantics. These difficulties are usually referred to as Semantic Problems. They can be divided into two broad groups. One deals with semantic integrity of the heterogeneous DBMSs, and has to do with assuring accurate data in the systems at all times. This implies assuring semantically correct updates to the DBMSs. This is a complex problem even in the context of homogeneous DBMSs. An example of this occurs when the constraint

policies of the local DBMSs differ with respect to each other. For instance one database does not accept prices greater than \$10, another database does not accept prices greater than \$5. What happens with an item of price \$8?. In such case, the update attempt will be accepted by one database while rejected by the other, producing inconsistency in the system as a whole.

The other type of semantic issue in heterogeneous DBMS environments appears in the context of information retrieval. The problem here results from inconsistencies between the data of the various local DBMSs. For example, two databases give values to a field; however, they contradict each other. This type of semantic conflict together with the definition of appropriate problem-solving strategies are the main focus of this thesis. From now on we will refer to them as Semantic Conflicts.

3.2. A Taxonomy of Semantic Conflicts.

The inconsistencies of information between different DBMSs occur at various levels. Assuming an attempt to integrate different DBMSs³, the objective of this section is to classify the conflicts in different groups in order to facilitate further analysis.

The taxonomy is aimed towards alerting and helping the system designer to:

- 1) Identify the various types of semantic conflicts that could be expected to raise.

³ This will apply to heterogeneous DBMSs as well as to homogeneous DBMSs.

2) Identify mechanisms or problem-solving strategies to approach the various types of semantic conflicts.

3) Identify special characteristics of the information contained in the DBMSs that could aid in solving some of the semantic conflicts.

The taxonomy consists of two main groups comprising two sub-groups each, as follows:

OBJECTIVE CONFLICTS: This group involves differences with respect to the way in which the different databases refer to their concepts. Also, to the definition or meaning of the values assigned to those concepts.

Syntax: *This category includes differences in wording.* Depending on the environment, different names can be used to refer to the same concept. For example, one DBMS may use the word "salary" and another DBMS can use the word "compensation". However, both refer to the exact same concept. Same thing happens between "rates" and "prices", "components and "parts". This could be as common as there are synonyms to a word.

Ambiguity: *This category includes differences in meaning.* The scope of a word is very large. Sometimes, the same word can be used, by different persons, to refer to different concepts, i.e. "diet" can refer to a person's usual habits for food and drinking. It may as well refer to a regimen imposed by a doctor. A wrong combination of data from two databases in the same context , yet each assuming different meanings for the same word, could lead to believe that a doctor has prescribed a person: to drink 7 Cokes per day.

More subtle however crucial differences may appear among concepts that involve numerical values. For example, "salary" may or may not include benefits. "Price" may or may not include taxes. Also, concepts that involve scale units, i.e. Distances may be expressed in miles or km. Careless checking of word meaning can lead to terrible conclusions.

SUBJECTIVE CONFLICTS: This group involves interpreting the information provided by the data. Objective conflicts should be solved at this point.

Contradiction: This category comprises the cases in which one database gives a value for a particular concept and another database contradicts it. It might take place in the form of a straight contradiction between database with respect to a fact, i.e. "the house has pool" vs. "the house has no pool". Or simply by providing different values to a concept, i.e. one database assigns "red" to the attribute color of certain item while the other database assigns "blue" to the same item.

Incompleteness: This category refers to lack of information for some particular items. How can we compare the salaries of some employees when some of them do not present any data ?. This cases require careful attention because lack of data can many times result from assumptions made by the database designer. For example, no-shows in the "salary" field can mean: "*annual salary below \$ 10,000*" in one source. However, in other sources lack of data with respect to a person's salary could just mean that there is no information about it.

Analysis of the type of problem involved in each of the 4 groups allow us to anticipate the kind of decisions that will be involved in trying to solve the different semantic conflicts.

AD-A195 853 OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE

34

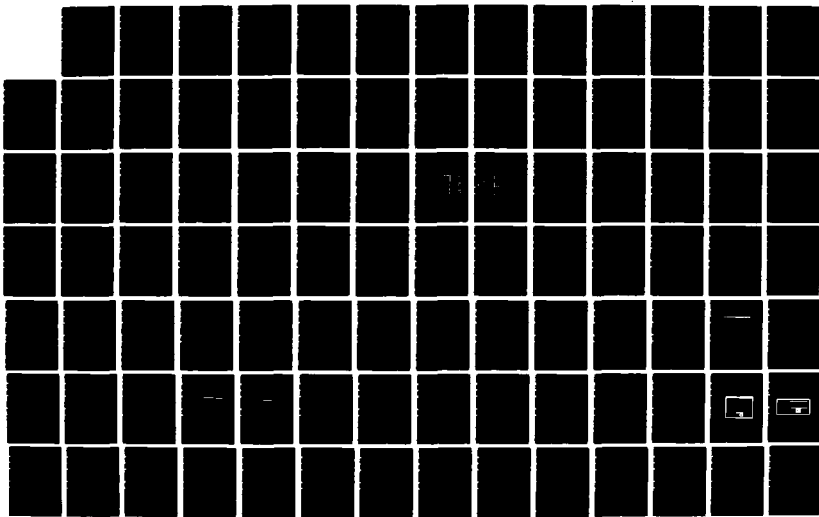
SEMANTICS VOLUME 4(U) MASSACHUSETTS INST OF TECH
CAMBRIDGE 8 GUPTA ET AL DEC 87 MIT-KRISE-4

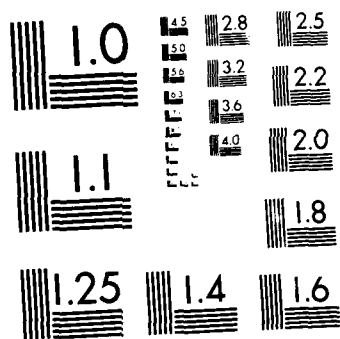
DTSS57-85-C-00083

UNCLASSIFIED

F/G 12/7

NL





Actually, the first two groups: Syntax and Ambiguity, in their simplest cases, mainly refer to discrepancies of concept names and meaning of similar concepts. The solution of the problems might be said to involve some kind of mapping or data conversion. Be this in terms of concept names -i.e. price vs. rates- or manipulation of the associated values -i.e. scale conversions: lbs --> kgs.

There are other cases where the meaning of the concepts vary. These are less obvious and involve more specific information about the meaning of a value. For example, it is important to know that prices in one database include taxes and service charge while not in the other one. Information about those quantities -- tax and service -- will be needed to sensibly compare and manipulate the data.

We will further refer to them as Conflicts that involve Objective Decisions. We say objective because assuming clear definition of the data, mapping between them requires following a specific process that will usually involve no uncertainties about it.

On the other hand, we have the third and fourth groups of semantic conflicts, namely, Contradiction and Incompleteness. These groups involve more complex problems that will in turn require much more sophisticated mechanisms to solve them. A quick analysis of the conflicts suggests that they will require some kind of subjective judgement to solve them. In the case of Contradictions, it will be necessary to decide whether one assertion is stronger than the other or if the conflict cannot simply be resolved. In these cases, it is important that all the concepts definition are clear.

The second case requires to decide how to handle the lack of data. Whether it should be considered as a negative or a a positive assertion. Whether it may be overcome by calculating estimates using other relevant data.

For these reasons, we would refer to the third and fourth groups as Conflicts involving Subjective Decisions.

We will present further analysis based on this taxonomy in subsequent sections.

3.3 Are The Semantic Conflicts Solvable?

Our objective is to be able to integrate information from different DBMSs. The task now is to find the ways of solving the semantic conflicts between them.

After the discussion about the link between individual mental models and generation of data models, the differences between local DBMSs seem inevitable. This assumption leads to initiate a search for new mechanisms that, recognizing the differences between local DBMSs, help us to deal with them in order to integrate different data effectively.

The first thing I will do is to analyze the process involved in human communication. This means to determine which are the necessary conditions to establish communication between different persons. Because we are very used to it, this process seems to be straight forward, however, it is not that simple⁴.

I claim that: Effective communication is only possible when all the parties agree on the "meaning" of the concepts used. Clarification of the meaning is

⁴ Even when we refer to common physical objects, like tables or chairs, there are a great deal of discrepancies between each person's own mental model of the object. These are influenced by taste, interests, needs.

accomplished by an explicit definition of the set of underlying assumptions associated with the concepts used.

Many times we find ourselves trying to get information from different sources. Soon, we start discovering several conflicts, in terms of ambiguity, incompatibility and inconsistency, among the information provided by each source. In such case, we have to undertake a process of re-definition that leads us to understand the differences in meaning from one source to the other. This process consists of a search for the underlying assumptions or criteria used by each source to define its data. This helps us to understand for instance, what is that source A means by concept X, and so on. Then, we can map concepts between the various sources.

The objective of this process is to normalize or express the individual concepts in terms of a common set of parameters⁵ that can be used by the different people to understand each other. The outcome of this process is a homogenized version of the heterogeneous information contained in the individual sources.

If we extrapolate the human analogy to the case of heterogeneous DBMSs, we have that in order to achieve effective communication each local DBMS has to clearly state the set of underlying assumptions or criteria used.

We assume that each criteria contains at least the definition of its data model - data, data structures, etc.. Specifically, definition of each field (concept) including range of possible values and clear explanation of its meaning. This first step opens the

⁵ These common set of parameters is obviously influenced by the person's own mental model of the concepts.

possibility for mapping between the various data models. The system now knows for example, how to map "prices" of one database with "rates" in another, or how to manipulate values expressed in "kilograms" and values expressed in "pounds".

Going back to the Taxonomy of Semantic Conflicts (defined in the previous section), we could say that a clear definition of the data and its properties -data model- should be enough to allow us to solve most conflicts of the first and second categories. These categories refer to Syntax and Ambiguity conflicts which we identified as conflicts involving Objective Decisions.

This type of information however, will not be enough to solve conflicts of Contradiction or Lack of information. For example, one database asserts something is "heavy", while other two assert it is "light" (contradiction) and another one does not say anything (lack of data).

Can we really expect to solve this dilemma ?

As it was mentioned above, these types of conflicts involve Subjective Decisions. Whether we can solve these conflicts or not depends on the particular context.

There are some contexts or applications where questions demand a very specific and accurate answer. For example, *What is the balance of my bank account ? What is the size of component X ?*. In these cases, answers which are built on the basis of subjective judgements will not be appropriate.

In contrast, there are other contexts where questions involve handling 'soft data' in which the corresponding answers are not expected to be accurate, but rather to

present an idea that can support the user in decision making issues. In these cases an answer based on "subjective judgement" may be absolutely appropriate. For example, *What may be the cost of producing certain machines ?, or what are the kind of profits in the paper industry ? Does country X has oil refining capabilities ?*

As mentioned before, these kind of questions are very frequent in Decision Support environments. Many times, there is not an easy way of getting exact information about a subject. In those cases, the best one can do is to consult the existing sources of information. The purpose would be to try to get some sensible conclusions from the assessment of different opinions. However, there have to be effective mechanisms to get valuable information out of the combination of these sources. A system that is intelligent enough as to be able to solve the semantic conflicts will prove very useful.

From now on in this thesis, when we talk about solving Semantic Conflicts we will be referring to the latter case of applications, specifically, providing answers for Decision Support purposes.

In the following section, we introduce some elements that are key in helping to solve Semantic Conflicts.

3.3.1. Problem-Solving Strategies.

The problem is now to try to identify some means that could help us solve Conflicts involving Subjective Judgement. First of all, we will assume, that at this point, all Objective Conflicts have already been solved. Again, drawing from the

human mechanisms, we find a group of strategies that can be used to solve semantic conflicts effectively.

In this sense we have 4 Problem-Solving Strategies: 1) Comparison of the different assertions; 2) Credibility of the sources; 3) Complementary information; and 4) Inferencing from similar data.

3.3.1.1. Comparison of the Different Assertions

The first thing we can do is to compare the assertions that, about a subject, are made by various sources of information. In the best case, this might disclose an unbalance among them that can favor a specific kind of assertion. For example, one database says that the range of cost for some material is \$ 3-7 and another says it is \$ 20- 30. In this case we may want to consult other sources of similar information. There are many possible scenarios. One of them is that we might find that other two sources also support the first range of prices, namely, \$ 3-7. This could help us conclude that the \$ 3-7 is the most likely range of cost for that material. There are still other ways to approach the problem, and this leads us to the next subsection.

3.3.1.2. Credibility of the Sources:

There are other scenarios where the comparison of data might prove useless. In these cases, we need to rely on other elements that could help us solve the conflicts. For instance, let us say we have the same example presented before. This time however, two sources support one fact and the other two sources support the other. How could we solve such confrontation?. There still are other mechanisms that could be used before giving up the possibility of solving the conflict. One of them is the "credibility" of each source. As we mention in chapter 3, the data model of each

database depends very much on the needs and interests of the person or group involved in its definition. It is rather common that some people focus or put more interest in specific concepts. Hence, the research done by a person interested in a particular subject will be much more exhaustive than that of someone less interested in it. Then, it is very easy to find that for example in the context of hotel guides, some guides are very good defining the style of a hotel while they are spotty in terms of rates of such hotels. In the same way, one finds guides that are very careful defining rates while weak in other areas.

The point here is that, assuming we can have knowledge about the weaknesses and strengths of the different sources, we can make use of the credibility element to adjust the scale in favor of one side or the other.

3.3.1.3. Complementary data: Recollection of unrelated, however relevant data.

There is still another scenario where we might have problems solving the conflicts of the sources. This time, let us say that two of the sources with the same credibility contradict each other and the other two sources do not have data about it⁶. In this case, looking into some complementary data might prove useful. An example will clarify this idea. Two different hotel guides might contradict each other in terms of certain facility that the hotel could have. For instance, one guide might assert that the hotel has "parking available" while another one denies it -both have the same credibility. Also, in cases like this we can make use of extra resources. For example,

⁶ This is the same case where all the sources with similar credibility contradict each other.

asking the sources about the location of the hotel might give us some insight on the chances that the hotel has in having "parking available". If we find out that the hotel is located in a city downtown, we may be inclined to doubt about those chances. By contrast, if we find that the location of the hotel is in a *suburb* or it is a *resort* then the possibilities of the hotel having parking might increase considerably.

In summary, there are usually some data that seems unrelated however, may prove relevant in solving certain kind of conflicts.

3.3.1.4. Inferencing from Similar Data.

Another scenario is the one in which we cannot find evidences to support an assertion. This happens when the sources lack information about some issues. In this case the strategy could be to see whether we can infer or fabricate an estimate from information provided by similar entities in our database. For example, somebody wants to know how far is certain hotel from the airport. That information is missing for that particular hotel. However, if we check "location" that might tell us for instance: "downtown". Looking for another hotel in the databases that is located in downtown, we might be able to cover for the lack of data in the databases.

Another example could be to find information of the cost of a room in a specific hospital where there is no explicit information about it. In this case, we could try to determine the type of hospital. Is it public/private, in which neighborhood is it located, is it associated with any organization or university?. Assuming we obtain this information, then we can go to the database sources, locate the group of hospitals that belong to the same category and then take an estimate of their room costs. It is likely that hospitals rooms in the same category do not vary very much.

In some cases, a combination of strategies will be needed. In other cases, the conflicts will simply not be solvable with the available mechanisms. In these cases, it will be good to present the user with the explicit cases. In this way the user is allowed to apply any personal strategy of its own. He might have his own set of assumptions about the credibility of the sources and he might want to get one of the individual assertions as the concluding value.

3.3.2. What Do We Need to Support the Problem-Solving Strategies?

We have just identified plausible strategies for solving Semantic Conflicts between heterogeneous DBMSs. The next step would be to determine what kind of tools or elements we have to provide in order to support the problem-solving strategies. A careful analysis of the previous description of the problem-solving strategies shows that we need a special kind of information. This time it will not be enough to have detailed information about the data contained in the various DBMSs. To be able to use these problem-solving strategies, we will need to have detailed somewhat subjective information about each of the DBMSs involved. This means that we need to know the strengths and weaknesses of each database in order to assign different credibility factors to the dbs. The information has to be very detailed, as each database will have to have a credibility factor associated with each of its fields.

Furthermore, we need information that gives us insight on which complementary data could be relevant for solving different Semantic Conflicts. Also, information on how and when to apply the various problem-solving strategies.

In the following section, we categorize the different kinds of information needed to support the enterprise of solving Semantic Conflicts between heterogeneous DBMS.

3.3.2.1. Knowledge About the Domain.

A system attempting to solve Semantic Conflicts on a particular domain is required to know about such domain. These information is needed first because the system need to interact with the user in order to determine the type of information requested as well as to be able to express the conclusions found in a sensible and useful manner. Second, the system needs to understand the kind of data involved in answering a user's request for information. Third, the system must have general information about the domain as to be able to make scale conversions and mappings between the different data. Fourth, the system needs to set some parameters relative to which to compare the assertions made by the various DBMSs.

This knowledge involves: Information about the relevant actors in the domain as well as the inter-relations between them. Information about the range of possible values for different elements in the system. Specification of general units and scales.

3.3.2.2. Knowledge about the individual assumptions

or criteria of each database.

These type of knowledge is particularly relevant for solving Semantic Conflicts involving Objective Decisions.

The system needs to have detailed information about each database. This includes criteria for defining the data -i.e. prices are expressed in yens- and data

structures and explanation of the meaning of the data -i.e. prices include taxes. This is very important in order to manipulate the data in a sensible way.

Very importantly, this category of knowledge demands explicit definition of the assumptions made. This is to state clearly all the default assumptions made by each DBMS. This can be accomplished by presenting for each field of the database its exact definition -i.e. rate = net price of room + tax, currency = dollars, unit = per person. This is a key factor in interpreting lack of data in a database. Default values can be very confusing for a person that is not very familiar with a particular DBMS. For instance, in the context of hospital information: A database can assume that all private hospitals have private rooms, thus this information is not explicitly stated in the database. An unfamiliar person might as well think that such hospital does not have private rooms.

3.3.2.3. Subjective Knowledge.

This group of information involves a higher level of knowledge. Specifically, about the characteristics of the DBMSs by themselves and by comparing them to other DBMSs.

In this sense, we will require specification of the strengths and weaknesses of each database with respect to the different data concepts or fields. Also, information about how the sources compare with each other with respect to the different issues.

The acquisition of this last group of knowledge can be obtained by consulting an expert that has had enough experience in using the various sources of information. In some cases it could be obtained by undertaking a careful study of the sources involved. In the latter case, the study should be as exhaustive as possible. The

expertise of the system in this category will greatly determine the intelligent capacity of the system to solve more complicated semantic conflicts effectively.

4. A SYSTEM FOR INTEGRATING HETEROGENEOUS DBMSs: **A KNOWLEDGE-BASED APPROACH**

Integration of heterogeneous DBMSs under a common user view may imply different things depending on the degree of integration pursued. In general, people refer to integration as the possibility to present the different local schemas in terms of a single global schema together with a common accessing method.

The most frequent approach is to provide point-to-point translation [Ceri:84]. For example, a user can ask for some data stored in DBMS "A" and as a consequence, the system will look for such data and then map it into the global schema, before presenting it to the user. Same thing happens if the user wants to access some data from DBMS "B". The system will take the data from DBMS "B" and translate it to the global schema. The functions of the systems are limited to extracting data and presenting it to the user in terms of the global model.

It is important to notice that, under this approach, no attempt is made to compare, combine or compose data from heterogeneous DBMSs.

This approach is already advantageous with respect to systems that do not provide any kind of integration. The user does not have to deal with the individual details of each DBMS in terms of access or data-models. The user always deals with the "same kind" of data. The system takes care of performing all the necessary changes to the local data so they all look similar -same format, same language- as though they were coming from a single DBMS.

Other approaches, by contrast, try to achieve a higher degree of integration. This consists of getting one step further than merely mapping the data of one DBMS into a global schema. The objective is to build systems able to generate information from the combination of heterogeneous DBMSs data.

In systems like this, users do not ask for retrieval of specific records of data from a DBMS. Rather, they ask for information. It is the system then, who has the task of deciding which physical records to access in order to gather relevant data. Once this is done, the system engages in the task of comparing the different assertions made by the different sources in order to build a sensible and useful answer for the user. In order to do this, the system has to confront the data from the various DBMSs against each other. This will inevitably raise several Semantic Conflicts that the system will have to resolve.

This is what we call a full integration of heterogeneous DBMSs. This kind of integration, its problems and possible solutions are the topic in which this thesis concentrates. The purpose is to provide a framework in which applications requiring full integration of DBMSs can be built. Specifically, applications which involve accessing different sources of information and thus reconciling the expected semantic problems that appear at different levels -objective and subjective (See chapter 3 for an explanation of these concepts).

Solving Semantic Conflicts is not a trivial task. Reasoning strategies will be especially needed to approach those conflicts that involve subjective decisions - evaluation of the different assertions made by the various sources of information. In this sense, we will require more sophisticated technology, than the conventional one,

to implement reasoning strategies efficiently and effectively. These reasons lead us to consider the use of Artificial Intelligence technology. Specifically, Knowledge-Base Systems (KBS) techniques seem to provide the appropriate environment in which to design and implement the desired capabilities of our integrative system. This topic will be further discussed in subsequent sections.

The solution we are going to propose consists of designing an intelligent system to act as an interface between the user and the various DBMSs (See Figure 4.1). The system is intended to work as a consultant for the user. The reason for using a variety of DBMSs is to have different sources available from which to gather different opinions. A good sensing of these opinions -DBMSs data- should allow for the building of sensible and useful information for the user.

4.1. Relevance of a Fully Integrated System.

The use of this approach is especially relevant for building Decision Support Systems whose comparative advantage are based on their capability to consult diverse opinions. From another perspective, the use of this approach is relevant for any application that requires resolving semantic conflicts that arise from the interaction of heterogeneous agents.

Examples of applications could be easily found in the strategic planning and managerial control areas, i.e. competitive corporate strategy. Systems like this are also useful when there is no easy way, or simply it is not feasible to verify the accuracy of

THE INTEGRATIVE SYSTEM

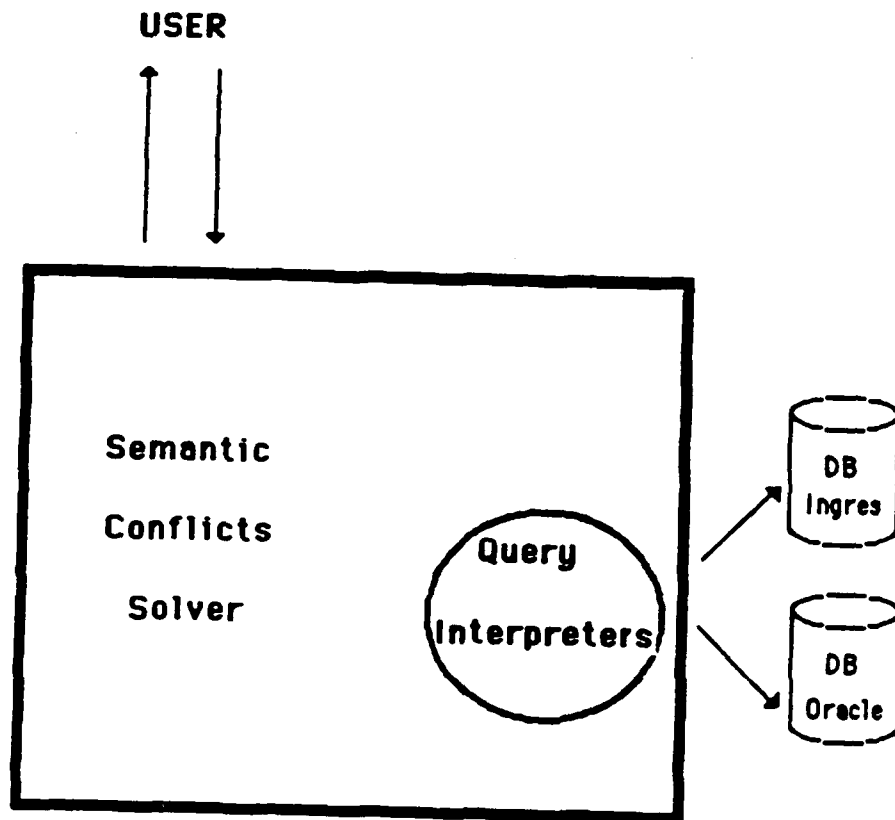


Figure 4.1

the data provided by one source of information. In these cases, the best we can do then, is to assess the data provided by different sources.

4.2. Goals Specifications.

The main goals we want to achieve are the following:

- 1) To design a fully integrated system of heterogeneous DBMSs by:
 - Providing the user with a common view of the underlying, different DBMSs. This implies presenting information to the user, rather than data. This involves solving Semantic Conflicts at all levels.
 - Facilitating the interaction between the user and the system.
- 2) To preserve the autonomy of the local DBMSs.

The first point has already been discussed extensively, let us explain now the second goal -or constraint- of the system.

An group of interconnected computers is a very "open environment" , in the sense developed by [Hewi:83]. There is a large number of people working concurrently on different projects, at different locations. In the context of DBMSs, there are many pre-existing DBMSs that are used by many people with different interests or purposes at the same time. This is a fundamental issue that should be given a great consideration: The local operations of the different elements of the system must be unaffected by the

addition of new elements to the system. Approaches that involve changes on the local DBMSs are already disturbing other uses that other people have given to them. If each person decides to modify the DBMSs for their own purposes, soon, there will be no database to be used by anybody.

In the next section we will state the main characteristics of the system we intend to design. These support our consideration about KBS being the appropriate environment to use.

4.3. Characteristics of a Fully Integrated System.

The main characteristics of a fully integrated system are complexity and non-determinism. It is complex because it involves putting together several different system which are very different in nature. It is non-deterministic because it involves solving problems that are ill-structured - i.e solving the Semantic Conflicts that arise between heterogeneous DBMSs. As known, conventional software technologies are suitable for very well-structured, deterministic problems. As important parts of our system do not present such properties, especial technology to approach them will be needed. Given an ill-structured problem, a KBS is capable of using its deductive capabilities to reason through the knowledge available generating possible options or solutions. KBSs seem to posses all the necessary attributes to provide an excellent environment for our application. They offer a highly advantageous approach for knowledge representation. In addition, their deductive capabilities appear to be ideal for solving Subjective Semantic Conflicts by means of heuristics.

4.4. Functions of the System

The overall functions of the proposed system are as follows:

- 1) Receive the user's request for information.
- 2) Organize a search on the different sources. This involves expressing the requested data in terms of the local schemas: Solve the Objective Semantic Conflicts.
- 3) Send the appropriate Queries to the external DBMSs.
- 4) Gather the various responses. This involves expressing the data collected in terms of the global schema: Solve the Objective Semantic Conflicts.
- 5) Compare the data collected: Solve the Objective and Subjective Semantic Conflicts.
- 7) Present the user with a coherent answer. In the event of many options show, for each option, the corresponding likelihood of occurrence.

4.5. The Basic Ingredient of the System: Knowledge

In order to carry out the previous functions effectively, we need to embody different types of knowledge.

In terms of accessing the DBMSs the system has to know about:

- the available resources or DBMSs.
- the mechanisms to communicate with each local DBMSs and their corresponding computer environments.

In terms of solving Semantic Conflicts⁷ involving Objective Decisions, the system has to know about :

- the definition of the global schema.
- the individual criteria or conceptual models used to define each DBMS (data and data structure definition).
- general concepts to be used for conversion purposes (for example, monetary exchange rates, conversion tables like km. -->miles, kg. --> lbs., country regulations on particular areas).

In terms of solving Semantic Conflicts⁸ involving Subjective Decisions, the system has to know about :

- the subject area or domain.
- the individual criteria or conceptual models used to define each DBMS (to

⁷ See chapter 3 for an explanation of this concept.

⁸ See chapter 3 for an explanation of this concept.

identify default values).

- the credibility of the different DBMSs with respect to different aspects (strengths and weaknesses of each information source).
- problem-solving strategies.

The next important question is how these different piece of knowledge should interact in order to generate the information required by the user.

It is clear that the needed knowledge is so diverse that it is crucial to find an effective way to organize it. By "effective" we mean easy access and manipulation.

Having described the general aspects of the integrative system including the goals we want to accomplish, it is now pertinent to expand our discussion about software technologies that could help us in such enterprise.

4.6. Knowledge-Base Systems

For many years, research in Databases and research in Artificial Intelligence seemed to go by non-intersecting paths as well as motivated by completely different interests.

Today however, many areas are being identified where possible intersections between the two fields seem to promise substantial potential benefits ([Alzo:86],[Vick:87]).

From the Database point of view, AI techniques may allow for the enhancement of data representation, data manipulation and interaction with the user [Wied:86, Kell:86]. In this sense, we find that more research is being done on the use of Object-oriented models as means to increase the semantic properties of the data [Neuh:85]. There is also important research in Natural Language processing as means to improve the interface between the user and the DBMSs [Spar:84]. Another interesting application of AI techniques in the Database field happens in query optimization [Miss:86].

Other research focus more on the informational side of the DBMSs. The objective is to add deductive capabilities to the DBMSs. In this sense, many approaches have been proposed. Some focus on building intelligent front-ends on top of the DBMSs using KBS techniques. Others propose to add Database Management capabilities into Expert Systems (ES); examples of these are Prolog-based applications that embed DBMS and ES capabilities in a single system ([Scio:86], [Jark:83]). Yet others try to embed ES capabilities inside the DBMSs.

The possibility of combining knowledge at various levels with reasoning capabilities opens new horizons in terms of the type of systems that can now be built. We think that a mixture of these two areas, AI and DBMS, could be exploited to achieve effective integration of heterogeneous DBMSs⁹. DBMSs are specialized systems able to efficiently access massive sources of detailed knowledge. Knowledge-Base Systems, on the other hand, are especially powerful for reasoning through the

⁹ Some authors had already anticipated the need for more sophisticated technology for developing systems of this sort [Kimb:81].

available knowledge. Their Inference Engines provide for efficient built-in deductive mechanisms. If we can provide the KBS with easy and efficient access to all the DBMS knowledge, we are already increasing the power of the KBS and thus the range of applications that can be built.

By providing inter-systems communication the KBS and the DBMSs can coexist as independent systems¹⁰. This allows for the DBMS to operate as a totally separate system with its own set of users.

4.7 Knowledge Representation

Complex problems require effective methods for representing the knowledge involved. In our case, the purpose is to find a method that allows to organize the knowledge to provide for easy access and manipulation of it. Object-oriented models appear particularly suitable to tackle this type of problems because they provide for: 1) a natural and intuitive way to deal with the problem; 2) easy organization and representation of the knowledge at high levels of abstraction. Object-oriented methodologies allow the designer to organize the knowledge according to its own human perception of the situation being model, rather than in response to the constraints imposed by the computer system used. Therefore, we can identify a shift from machine-oriented data models to user-oriented semantic models.

¹⁰ This could be very advantageous because the KBS would not have to keep all the data inside, but rather access it -through a DBMS- as needed.

Following these lines of reasoning, we have decided to structure our system using the Object-oriented model. In the following sections we describe the overall architecture together with the design principles that we propose for designing Fully Integrative Systems.

4.8. Framework Description.

We will start by explaining the particular design principles that we will use in organizing the objects themselves. The purpose is to decentralize the activities of the system as much as possible. The main motivation, besides those of modularity and flexibility, is to embed parallelism into the system as a means to achieve better performance, provided that concurrent environments are available, for example, multiuser, parallel architectures, and so forth.

First of all, we will decompose the problem in small pieces that represent the main actors of the system, as seen from a high level of abstraction. Knowledge will be encapsulated in the different actors or entities. Each actor has a set of attributes associated with it. These attributes describe the actor as well as the operations that define its relations with other actors in the system. The actors are very self-contained in the sense that they are limited to very particular knowledge. They just know about themselves and how to relate to certain external entities. This is actually, very useful because it allows the user to clearly identify the actors of the system, their tasks and their relationships. From a software engineering point of view, this model brings many

advantages in terms of organization, debugging and maintenance. All the communications between the actors take place in the form of message passing.

Although the overall model for organizing our system will be Object-oriented, we will also make use of Heuristics or Rule-models. The Inference Engine is the heart of a KBS as it provides the powerful deductive capabilities of such systems. Our system in particular needs to exploit these capabilities to support its decision making process. This capabilities are exploited by means of using heuristics or rules.

Rules will also be encapsulated as objects. Then, every time we want to trigger a particular set of rules, we send a message to the corresponding object. This is very convenient because it allows us to organize the huge number of rules in individual groups according to the functions they perform. This is a great advantage in terms of debugging the system because it allows to easily pin-point a smaller set of rules for a specific problem.

The overall methodology that we will use in developing the proposed system is as follows. All the factual knowledge, like definition of concepts and properties of them, will be represented as traditional objects. This is as sets of slots, attributes and values together with operations that the object or actor should perform in response to messages from other actors. All the Objective Decisions that the system has to make in order to solve the corresponding semantic conflicts between the *global schema* and the *local schemas* will be coded as operations of certain objects. This means in a procedural way. By contrast, knowledge about strategies for solving Subjective and Objective Semantic Conflicts *between* different DBMSs will be encapsulated in rules composed by sets of heuristics.

4.9. Design Specifications.

In this section we describe the actual elements in the proposed architecture for designing integrative systems. We describe the different types of actors involved and the interactions among them.

4.9.1. Actors in the System.

The actors in this framework can be divided in two groups. First, actors that will always be present regardless of the application and domain chosen. Second, other actors that will basically depend on the application, the domain and the designer's perception of the situation. Once the permanent actors are defined the designer can start creating the second group of actors. These will serve as support to the system and will usually include general knowledge about the domain as well as other relevant information.

It is important to notice that each actor in the system is very specialized. Each of them has a unique responsibility or role to play and there is almost no overlap between them. Their responsibilities might be similar in nature but completely different in terms of the actual job performed -different applications. The cooperation among them is what determines the overall performance of the system.

Further on in chapter 6, we present an example of an actual implementation using this framework. This should help the reader to better understand the proposed system.

As the first group or Permanent Actors constitute the building block of the whole system, we will proceed to describe them in great detail.

1) The Manager of the system: This actor is in charge of directing the activities of the system, this is to assign responsibilities to the other actors. It also constitutes the receptionist¹¹ for the system which means that it is the only actor that communicates with the outside world¹². This actor is in charge of helping the user to define its request; which it does by taking him through a number of menus. Also at the end, it is in charge of presenting the report of findings.

This actor's most important role is to solve the Semantic Conflicts -Objective and Subjective- that arise from comparing the individual reports of the local DBMSs against each other. For solving the Objective Semantic Conflicts, this actor relies in the System Criteria. This helps it to do the appropriate mappings between the local and global schemas.

To be able to solve the Subjective Conflicts effectively, the actor relies heavily in different groups of object composed by rules. These rules contain the necessary knowledge about problem solving strategies to tackle the different types of Subjective

¹¹ This concept has been borrowed from the "Actors model of Computation"[Agha:86].

¹² This is especially relevant in fully concurrent systems [Hewit et al:85] See Gul's.

Semantic Conflicts. This actor will be responsible for triggering the different rules according to the situation. It must also be supported by other actors containing information about the characteristics of the domain.

2) The Global Schema or System Criteria: This actor holds all the necessary information about the global schema of the system. This includes detailed information about the assumptions made by the system in trying to define a common model for reconciling the differences between local DBMS. The knowledge of this actor could be also captured in rules. However, it makes it much more difficult to modify. For this reason we recommend having a separate object to hold all the definitions of the system.

3) DBMS representative or DBMS Manager: There should be as many actors of this type as there are DBMSs available to the system. Each of them should be assigned to one particular DBMS. This actor's role is to be like a local manager for a particular DBMS. It could be regarded as the "logical" DBMS. First of all, it acts as the receptionist of this DBMS. This means that it will receive and process any request sent to the DBMS that it represents. It will also be in charge of reporting back any findings to the System Manager. It does not do any actual search for data, but rather delegates on other actors to do it. It has enough knowledge about its domain, namely its DBMS, as to: 1) know which kind of information is supported by its DBMS; 2) know how to delegate, who to ask in order to find the requested information; and 3) solve the Objective Semantic Conflicts that arise between the global schema of the system and the local schema of its DBMS and viceversa. This implies doing the mapping between the two. For accomplishing the latter, this actor relies heavily in the DBMS criteria - to be explained next.

4) DBMS criteria: This objects hold all the necessary information about the local schema of a particular DBMS. As for the previous actor, there should be as many actors of this type as there are DBMSs available to the system. Each of them should be assigned to one particular DBMS.

This object has exact information about its DBMS local schema. This includes detailed information about the assumptions made in building this local schema such as all the important default values. This actor reports about its attributes upon request from the other actors in the system.

5) DBMS Interpreter: This is the technical actor of the system. As for the previous actor, there should be as many actors of this type as there are DBMSs available to the system. Each of them will be assigned to one particular DBMS.

Each of these actors holds the necessary knowledge to communicate with its associated DBMS. These include: knowing about the query language and how to generate queries. Also, knowledge about the process to establish communication with the host computer. This actor is called every time that somebody wants data from the DBMS. These actors are completely dependent on their DBMS, in the sense that each of them knows only about its associated DBMS and nothing more. All of them could be working at the same time in their corresponding activities without disrupting any other.

4.9.2 Different Levels of Communication Among the Actors.

This section presents a detailed description of the different "levels of Communication" that occur in our model for Integrative System. The problems involved and the solutions proposed are presented for each case. It is important to repeat that the cooperation among actors is what determines the overall performance of the system. See Figure 4.2.

We identify 4 different levels of Communication:

- 1) User --> System,
- 2) System --> System,
- 3) System --> External (physical) DBMSs, and
- 4) System --> User.

Each of them, including the specific actors involved, are explained in the following paragraphs.

1) User --> System: This involves the interaction between the User and the System manager. Simply stated, the user interface. See Figure 4.3

In the fully integrated environment that we have proposed, the user does not ask for specific data stored in the DBMSs, but rather expresses its informational request to the system which in turn communicates with the individual DBMSs. As it is only one system with which the user interacts (see fig. xx), there will obviously be a "unified access method", so in that sense, we meet the first goal. The point now, is yet to choose the best method for the user interface with the system. There are several

THE DIFFERENT LEVELS OF COMMUNICATION

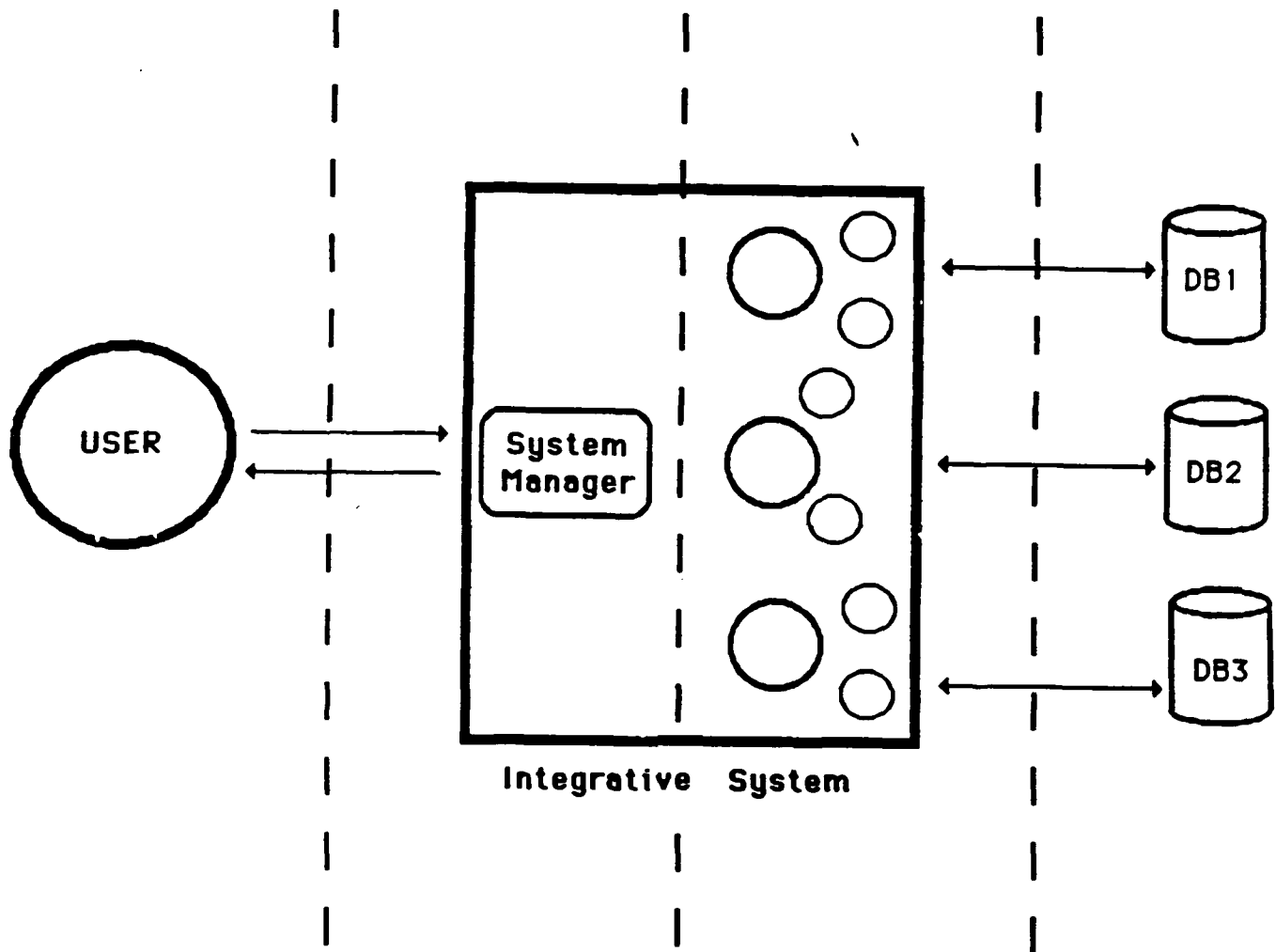


Figure 4.2

the user does not need to get involved with the actual local schemas, a DBL does not make sense any more. More user-friendly options are in place. Among these, we identify Natural Language and menu-driven interfaces. The selection of the most appropriate interface in this case can be left to the system designer. Yet we will describe what we think is the best method.

As we mentioned in Chapter 3, we do not think that the technology for processing Natural Language is ready yet. Even in the event that it were, we think that this approach presents many problems for the system, as it cannot constraint the type of questions made by the user. This approach unnecessarily increases the complexity of the user interface as the system has to implement complicated mechanisms for the translation between the user's request and the global schema.

On the other hand we have the menu-driven option. Some may say that it is not as elegant as the Natural Language option, however it presents good advantages for both the user and the system. In terms of the user, this option is very easy to use. It does not require major typing -in most cases it will involve typing one or two characters or a click to the mouse. The process consists of the system presenting different options to the user according to his previous answers and thus, helping him to navigate through the different alternatives. From the system point of view, this process is advantageous because it allows it to keep control at all times. As it is the system who present the options, it can constrain the user to those options available in the system. The system keeps record of the path followed by the user, so while the user is selecting the options, the system can be building the request. This makes the translation process is almost straight forward.

By contrast, this same translation would require a much more complex processing in systems using DBL or Natural Language user-interfaces.

For these reasons we will recommend the adoption of a menu-driven interface.

2) System --> System:

This involves the cooperation of several actors within the system perform the necessary operations. See Figure 4.4. This can be divided in three groups. The first involves the System Manager delegating in each of the DBMS Managers. The second involves each DBMS Manager delegating in its constituencies: the DBMS criteria and the DBMS interpreter. The third one involves the DBMS Managers reporting their findings back to the System Manager.

a) System Manager --> DBMS Managers:

Once the System Manager knows the exact content of the user's request, it can start delegating the actual search of the data in the DBMS Managers. The System Manager does this by generating messages that contain the requested data, and then sending them to each of the DBMS Manager. It is important to notice that the System Manager does not perform any kind of mapping between global schema and local schemas. This job is left for the DBMS Managers to do.

The delegation of tasks, at this point, is very important because it automatically embeds parallelism into the activities of the system. As each DBMS group of actors¹³

¹³ The group includes: the DBMS manager, the DBMS criteria and the DBMS interpreter.

Communication between
the User and the System Manager

User Interface

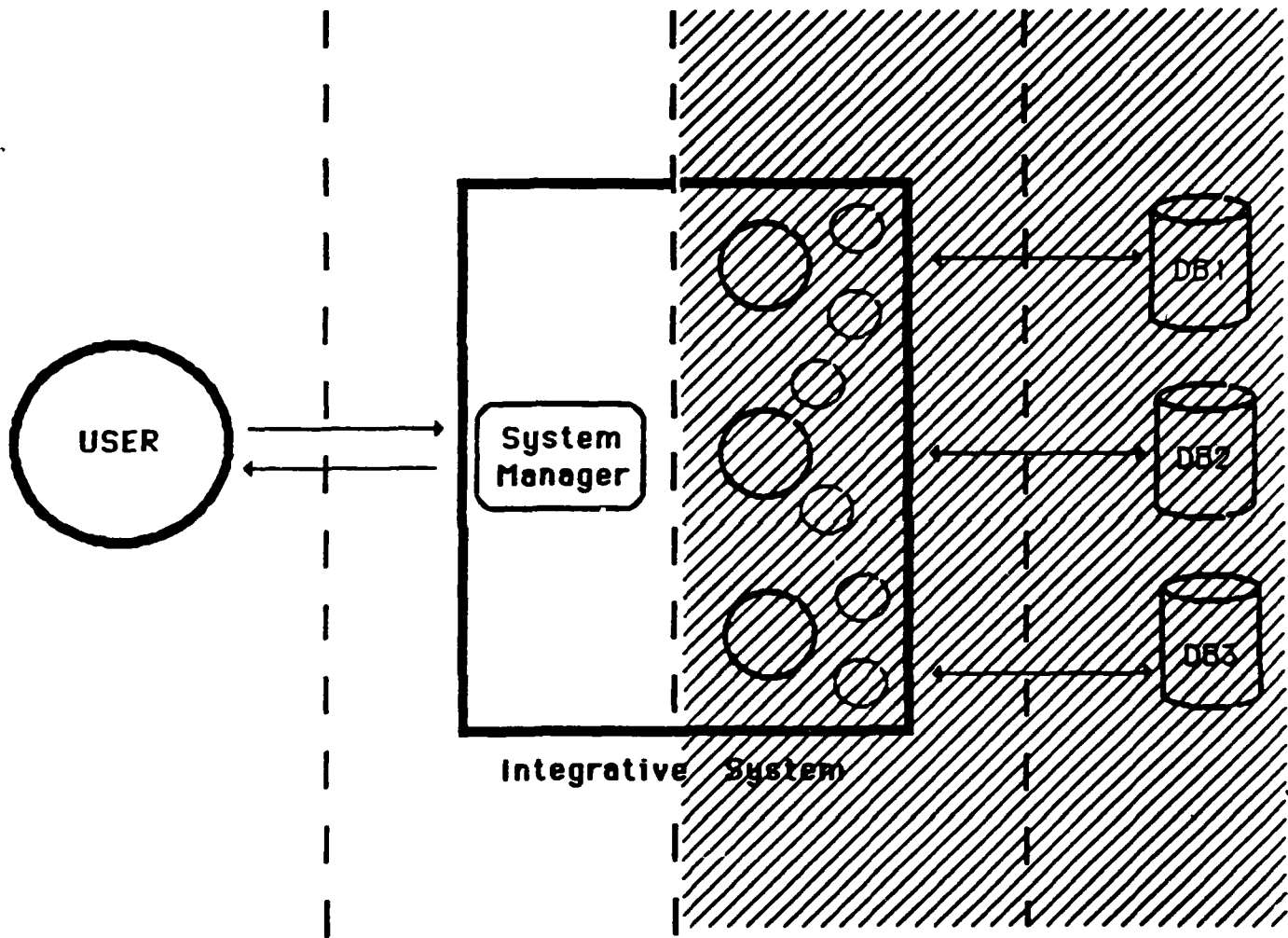


Figure 4.3

**Communication between
the System and the external DBMSs**

DBMS interpreter and physical DBMS

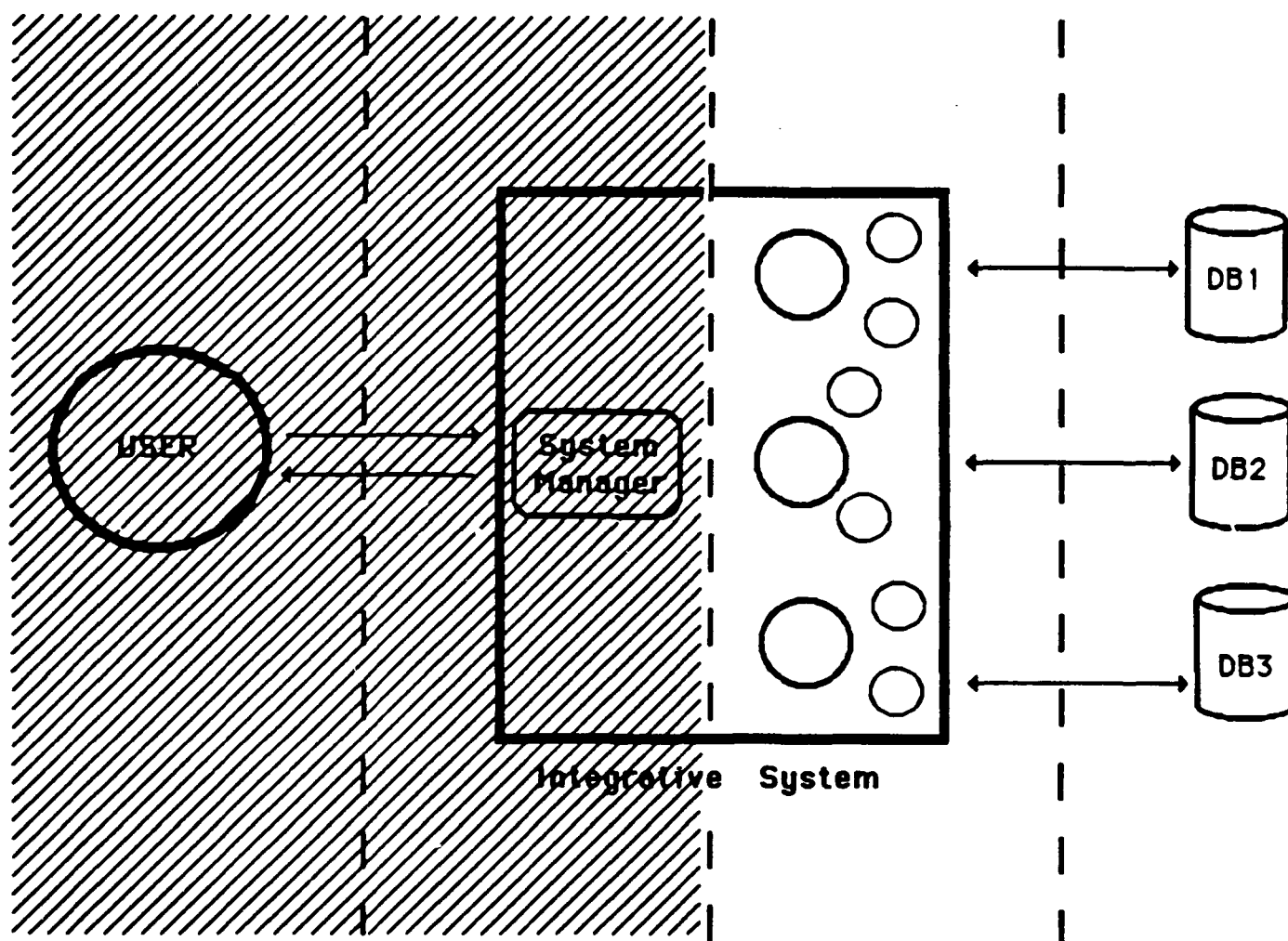


Figure 4.5

is completely independent from each other, they can perform their individual tasks in parallel. For example, each logical DBMS could be establishing communication with their corresponding physical DBMS at the same time. Of course, the degree of possible parallelism will ultimately depend on the architecture of the system used.

b) DBMS Manager with its constituencies: the DBMS criteria and the DBMS interpreter.

As mentioned in previous sections this actor is expected to know:

(i) about its DBMS domain: data supported by its database as well as relevant data that could complement a request for some particular information. (ii) where to look for data and information (criteria / DBMS / both), (iii) how to map the global schema to the local schema and viceversa. This actor does not access the physical DBMS, but rather delegates this task in the interpreter which only function is to build appropriate queries and establish communication with its DBMS and corresponding host.

The process followed at this stage is as follows. First, the DBMS manager has to transform the message sent by the System Manager, from global schema to local schema terms. After that, the usual process is that the DBMS manager sends request to both the DBMS interpreter and the DBMS criteria -the latter will be explained below. The message to the interpreter is a request to access the physical DBMS. The message contains the necessary data that the interpreter should need in order to build a sensible

query. Details about the functions performed by the DBMS interpreter will be given in the following subsection.

The DBMS actor relies heavily in its DBMS criteria for insightful information about the data requested. It is important to remember that the criteria actor possesses a detailed description of the local schema of the DBMS. The message to the criteria is a request for information about aspects of the data requested. One of the most valuable contributions from the DBMS criteria is the specification of default values or data that is not showed in the databases yet, is assumed to be part of the attributes of a field. As these are not explicitly stated in the physical DBMS, a mere consultation to the DBMS will not report them. This would produce misleading information.

c) **System --> External (physical) DBMSs:** This interaction takes place when the system requires to access data records that are stored in the external DBMSs. The communication involves the DBMS interpreters and the external DBMSs. See Figure 4.5.

The particular functions of the interpreter are as follows:

- (i) to generate the appropriate "DBMS query"
- (ii) to establish the actual inter-connection with the external computer environment that hosts the DBMS and the DBMS itself.
- (iii) to send the query
- (iv) receive the response

Delegation of tasks within the System

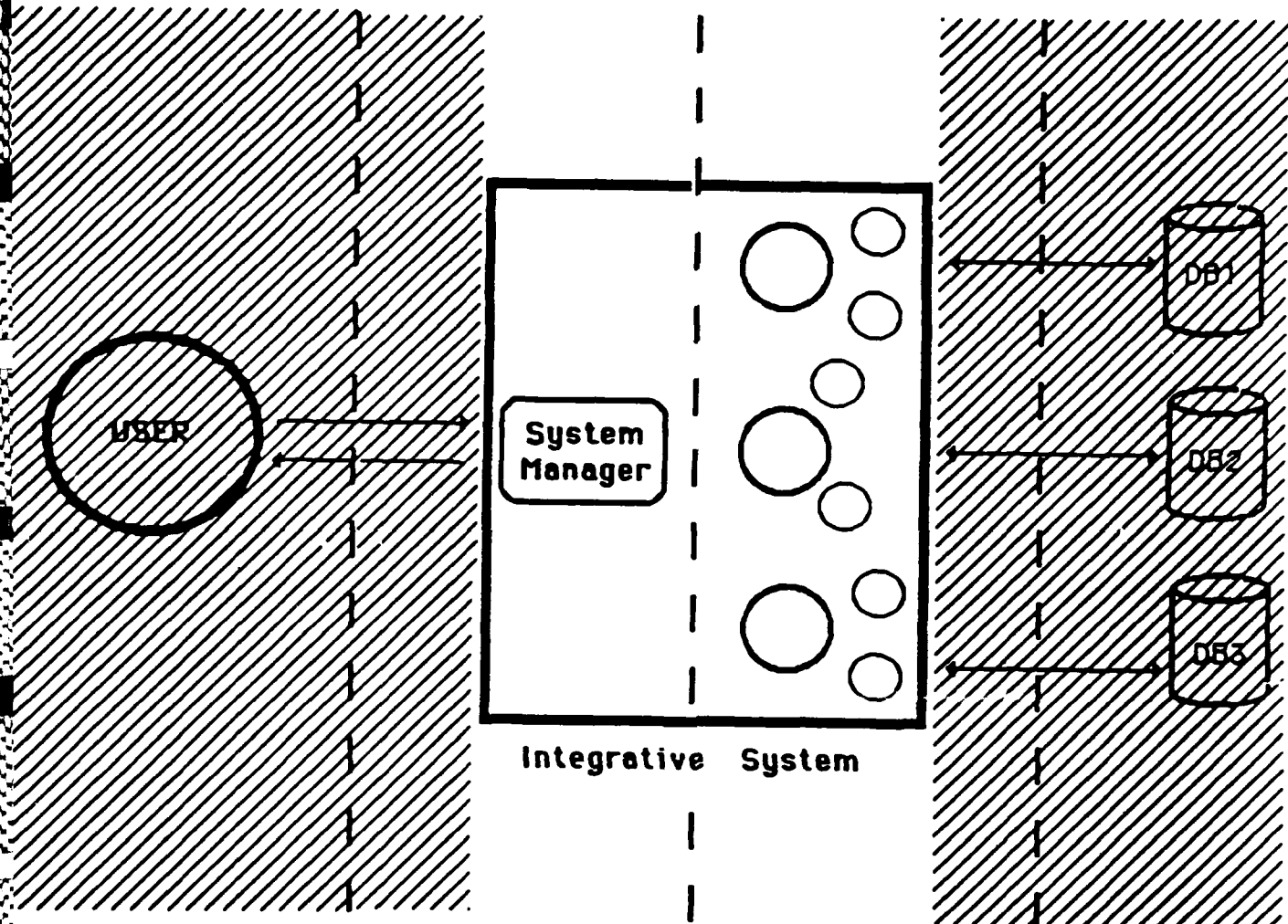


Figure 4.4

(v) send a message to whoever made the data request with the response obtained from the DBMS.

This process is very efficient given that each of these actors are customized to address one and only one DBMS. In this sense, they could be said to be special processors within the system for communicating with the corresponding DBMSs.

d) DBMS Managers to the System Manager.

Once the DBMS manager has gathered enough information from the physical DBMS and its criteria, it does a reverted mapping, this time from local schema to global schema. Then, it sends the results to the System Manager.

At this point, the System Manager takes control of the activities again, and proceeds to evaluate the information presented by the logical DBMSs. The most complicated problem is to solve the Subjective Semantic Conflicts that arise among the reports presented by the different DBMSs. This is one of the most important stages in consummating the integrative goal of the system. As a starting point, the system has to solve the Objective Semantic Conflicts among them. As it was explained in detail in chapter 3, the system will use different problem-solving strategies to tackle the Subjective Semantic Conflicts. These will be: 1) Comparison of evidence; 2) Credibility; 3) Search for Complementary data; and 4) Inferring from Similar data. For an introduction to these strategies, refer to Chapter 3, under "Problem-Solving Strategies".

Some times the simple comparison of the various reports give us an immediate answer. However there are other cases where more subtle reflection has to be made.

This is when the Credibility concept comes into play. We will extend at this point the explanation by describing a possible implementation of this concept.

First of all, the designer, after an accurate research on the matter, assigns Credibility Factors -weights- to the DBMSs. The assignment involves each field of the database. This is important because the actual comparisons of credibilities between the sources will depend on the field in question. For example, in the context of a Market Research, the DBMS produced by company "A" might be very good specifying "market shares" for different companies. However, it can be flaky when talking about "list of products" produced by each company. On the other hand, in the DBMS produced by company "B" things may be reversed. Company "B" may focus on the "list of products" for each company while not being very specific with respect to Market shares.

There are several ways in which the Credibility Factors can be quantified. For an example, see chapter 6. The idea is to assign different weights that could influence the final evaluation of the assertions involved. This means, that more weight -or more points- would be given to whatever assertions were made by sources which are particularly accurate in certain field.

There are some cases where no straight answer seems possible. This are usually cases in which the opinions of the different DBMSs strongly disagree. We detect these cases by checking the Certainty Factors associated with the assertion. This factor is basically a measure for the confidence with which the system can support its conclusions. This factor is the result of the weighted aggregation of the assertions made by each guide; "weighted" because the value of each assertion is assessed

according to its guide's credibility factor¹⁴. The value of this factor will reflect the degree of conflict among the different assertions. For example, if too much controversy is found in trying to define whether a field is positive or negative, the Certainty Factor will have a low value. For example, in the case of a range between 0 and 1, a low value would be something less than .5. If on the other hand, a confident conclusion can be made, the Confidence Factor is assigned a high value -i.e. greater than .7. A high value means that no further search for confidence is necessary. By contrast, a low value warns the system on the flakiness of the conclusion made and invites it to try other methods in an effort to increase the degree of confidence for the conclusion.

In this cases, the system uses Strategies 3 or 4. Strategy 3 consists on getting other relevant data from the system, that could help clarify the uncertainty. See chapter 6, for extensive example of this strategy.

Strategy 3 applies to those cases where lack of information block the possible resolution of a conflict between DBMSs opinions. In these cases the system tries doing some research or survey on the values of other similar data. The actual research depends on the context as well as on the concept involved. For example, let us say that we are interested in knowing the value charged for a particular service by certain company; but, there is no information about it in the database. A possible solution may be to check the rate values charged by other similar institutions for the same service. This could be done by asking the DBMS to provide records that belong to a

¹⁴ Among others, this differs from the Credibility Factor in that the Certainty Factor is assigned post-evaluation and to the assertion or conclusion itself, not to the individual fields.

certain type of institution and that perform the desired service. Furthermore, if the system explains the problem to the user, the latter can use this estimate as a reference point for comparison. Then, the most appropriate approach to manipulate the rates will be to program conventional procedures in a high level language -i.e. Lisp, C. This strategy leads towards finding estimates that some times can contribute in clarifying the conflicts, thus, increasing the Certainty Factor. See chapter 3.

At the end all the assertions or conclusions made by the system will have their corresponding Certainty Factor (CF). This will help the System Manager to present the user with an accurate sense of the final degree of confidence with which the system can support each of the conclusions. For example, in cases of high CF, the system will fully support the conclusions. In cases of medium CF, the system will say something like: "There are *some chances* of X being ...true/white/35...". In cases of very low CF, the system will say: "There are *very few chances* that".

4) System --> User:

After the System has made its conclusions, these are presented to the user together with their likelihood of occurrence. At this point the user might want to inquire about any particular result. In this event, the system should use the explanatory capabilities of the KBS to show the user the exact conditions -rules, etc.- that lead the system to reach such conclusion. This should help the user to further understand the reasoning followed by the system.

This concludes this Chapter. We have described a framework or architecture to design fully integrated systems of heterogeneous DBMSs using a Knowledge-Base approach. In the following chapter we present an example of an implementation of such a system, using this architecture and the corresponding design principles.

5. A CASE STUDY: THE HOTEL INFORMATION SYSTEM.

Effective communication is only possible when all the parties agree on the "meaning" of the concepts used. Clarification of the meaning is accomplished by an explicit definition of the set of underlying assumptions associated with the concepts used.

In the previous chapter, we described a framework to design fully integrated systems of heterogeneous DBMSs using a Knowledge-Base approach. We will like now to present an example of an actual design and implementation of such a system using this framework, in the domain of Hotel Guides.

5.1.Problem Definition

As an illustration of the applicability of the framework defined, we will look at the problem of extracting information from different sources. We want to identify a domain with certain characteristics. First, there must be a number of alternative sources that provide information about the domain. Second, each source must have an individual way of expressing the information. Third, the problem must be common enough as to show immediate practical application of the solution proposed.

The domain we have chosen is "hotels' information". This domain matches all our criteria for an interesting problem in which to test our hypothesis. First, there are enough guides that hold this type of information. Second, each guide focus on certain aspects of the hotels, thus, in order to find appropriate information, we will probably have to go through more than one guide. For example, most guides provide -in one

way or the other- data on rates; however, not all guides offer information about location or insightful comments on the quality of the hotel. Third, this is a common problem that can be appreciated by many users.

For example, let us say that you will be going to Paris in 3 weeks. You have been recommended a couple of hotels, however, you will like to know more about them, in order to make a good decision. Let us assume that you have a computer at home -or at work- that can be connected, through a network, to an Integrated System for Consulting different Hotel Guides. Wouldn't be nice for you to use such a system to find information about the hotels ? Furthermore, you can do it at your own pace and be as meticulous as you like because "the system does not become impatient". The greatest advantage of all is that you do not have to deal with either the technicalities involved in accessing each DBMS, or the particular criteria for the guides definition.

The system proposed will act as a mediator between the user and a number of guides describing hotels in the same city. The guides are stored in different DBMSs. The purpose of the system will be to provide the user with consolidated, composed information about hotels by sensing the different opinions of the guides. In our particular application we will use 3 guides for hotels in Paris: "Michelin", "Official Hotel & Reservation Guide (OHRG)" and "Fielding's". Furthermore, they will be assumed on-line and using the DBMSs provided by Oracle, DBase3 and Ingres.

The fact that the information is coming from different sources with different data models should be almost transparent for the user. The function of the system is to understand a user's request, and to use its own knowledge about hotels in general, to ask the appropriate information to the different guides. Also, by using its knowledge about each guide's criteria plus its knowledge about the strengths and weaknesses of

those, reconcile the underlying semantic conflicts and provide the user with a sensible and useful answer.

A detailed description of the model, using the framework defined in the previous chapter, is given in the following chapters.

5.2. Characteristics of the Problem.

The problem of solving differences of opinion between guides is characterized by ambiguities, inconsistencies and uncertainties. The problem-solving strategies are mainly based on heuristics. Conventional programming methodologies that expect very well specified and unambiguous problems are certainly, not the best environments to host them.

Furthermore, we have the issue of knowledge representation. To solve semantic conflicts in the hotels context, we require a lot of information about hotels in general, about the guides and their criteria, about the general assumptions made by the system in interpreting different concepts and about problem-solving strategies. The complexity of the knowledge needed demands creative methods to provide for easy and flexible access and manipulation of it. It will be certainly possible to use conventional methods, but also tedious and difficult as to make it not effective. Unfortunately, the natural relations among the elements would be obscured by the implementation itself.

On the other hand, we consider that these same characteristics make the framework proposed before an excellent environment to host the problem. To use this architecture seems suitable for: 1) representing the necessary knowledge; 2) handling

the problem-solving needs of the system ; and 3) supporting access to external sources of information.

The framework for organizing the knowledge, based on an object-oriented model, provides a great environment to represent inheritance features between classes and sub-classes of the same element and relations between different elements.

The resolution of the Subjective Semantic Conflicts is primarily a cognitive process that involves chains of reasoning. The heuristics can be easily expressed using rules.

Finally, the capability of accessing external DBMSs enriches the abilities of the system. The Knowledge-Base System does not have to hold the information about the guides within itself. Rather, it receives samples of it upon request.

5.3. Knowledge Acquisition

The knowledge, on which this system is based, is the product of a careful study of hotel guides both individually and in comparison to one another. This included the study of the descriptions, that for particular hotels, were given by different guides as well as comparison of hotels of similar category. It was also of great help to interview many travel agents who perform this kind of tasks daily.

However, the most important aid in understanding the nature of the Semantic Conflicts involved, was going through the reasoning process of solving them in practice; trying to download the specific methods we use to extract useful information from the data, even when it is ambiguous or data is missing. This reasoning process is

embedded in the in the various sets of heuristics used in the system to solve the semantic conflicts.

5.4 The Knowledge of the System

As we said in the previous chapter, the success of the system will greatly depend on the amount of knowledge it has about the domain. In this sense, we could roughly divide the knowledge needed in 4 groups:

Knowledge about hotels in general: the system needs to know about the general characteristics of hotels. This means to know that a hotel has rooms and facilities; that rooms have beds and bathrooms; that each room has an associated cost, etc.. This is especially necessary in order to understand each guide's individual information in the context of the overall system.

Knowledge on how to address each guide: the system needs to know how to address the different guides, how to ask for the different concepts. For example, it has to ask one guide for "quality" while another one for "ratings"; or one guide for "prices" and the other one for "rates".

Knowledge about the criteria used by each guide: the system needs

to have the set of underlying criteria used by each of the guides in order to understand the data received. It needs to the exact definition of the fields as well as the assumptions made in terms of data omissions -default values.

Technical knowledge on how to access each guide: as the guides will be physically stored in databases, the system will also need to know about the mechanisms to communicate with the external DBMSs. This involves knowing the particular Database Languages in order to build the appropriate queries.

The activities of the system are based on inter-relations among elements that manipulate the knowledge at different levels.

5.5 System Description.

In this section we define the Hotel Information System according to the framework defined in the previous chapter, for building Fully Integrative Systems.

5.5.1. The Objects in the Hotel Information System.

The system is organized as a community of objects. The role of each object is to represent a particular element or actor in the HIS. Each object is an individual entity

that: 1) has information about itself, and 2) knows how to interact with external objects. The internal information includes both definitions of the object and the operations it is able to perform. See Figure 5.1

In the following sections we describe the objects that compose our system together with the inter-relations that occur among them.

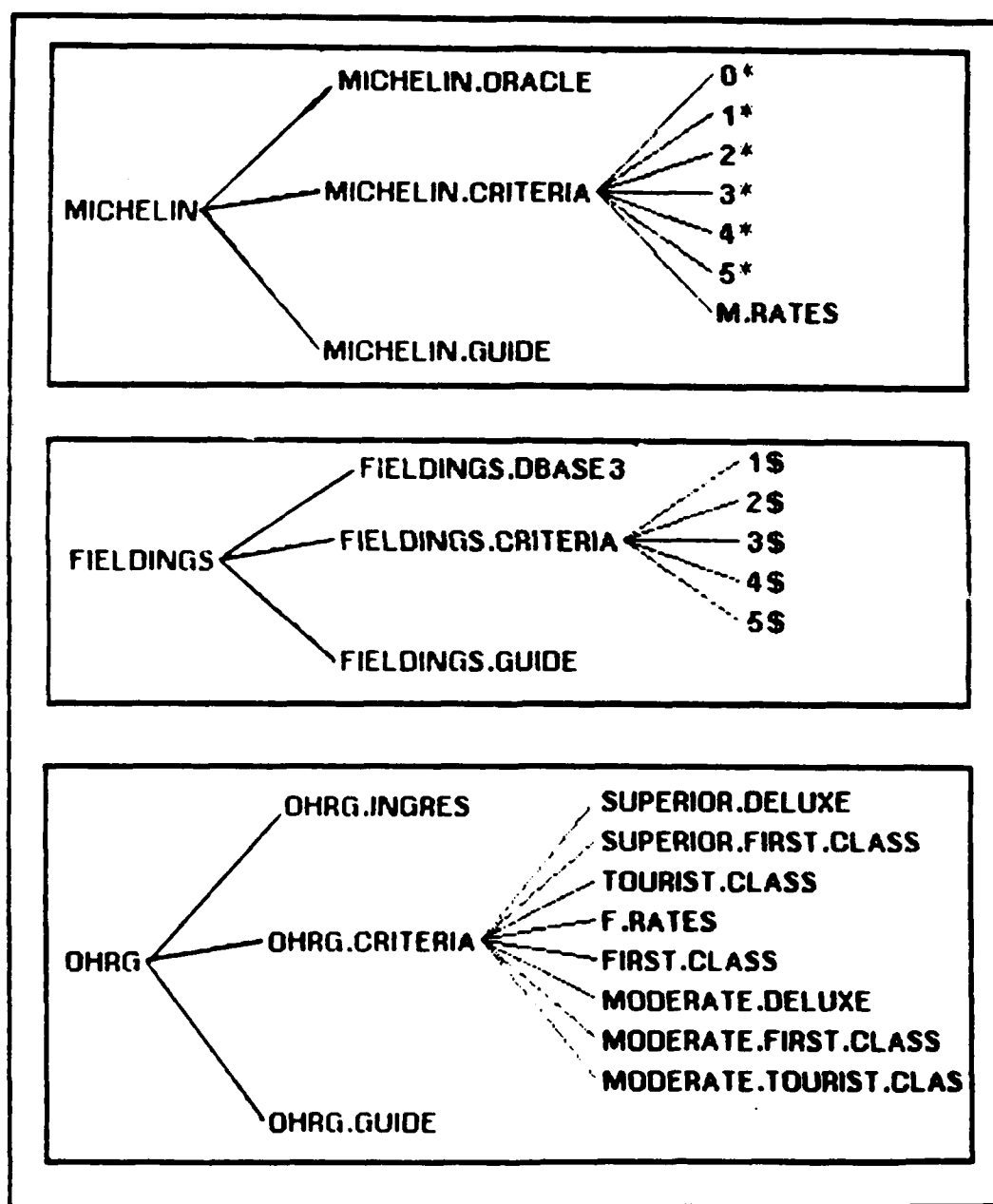
We will start by describing the permanent actors of the HIS.

5.5.1.1. The System Manager or Director-Object.

This object is in charge of assigning and coordinating the activities of the different objects. It is the receptionist of the system, thus responsible for the user interface. This is the actor that bears the ultimate responsibility of solving the semantic differences that arise from the opinions of the various guides. These will include both solving Objective and Subjective Semantic Conflicts between the different DBMSs. In doing so, this actor relies: 1) in the System-criteria or Global Schema; 2) in a set of rules which hold problem-solving strategies to tackle different problems; 3) in the guides-criteria or local schemas; and 4) in other objects which hold complementary information about the domain.

As an example of controversy between guides we have that, one guide may say that "*all rooms have bath-tub and shower*" while another says that "*rooms have only bath-tub*". Depending on its knowledge about the questioned item, the System Manager will act in different ways to solve the conflict (see chapters 3 and 4). It may decide to ask the guides to provide other pertinent information that could lead it to

Figure 5.1: The Permanent Actors for Each Guide.



clarify or support any of the statements. It could otherwise decide to give more weight to one of the assertions because the corresponding guide has a better credibility in terms of describing hotel rooms. In the event where the conflict cannot be solved, it will have to present the user with the ambiguous information together with the probabilities of certainty derived from the consultation to the different guides. In this case, the user will have to draw its own conclusion.

5.5.1.2. The System-Criteria.

This object provides the "common language" -global schema- needed by the system to accomplish effective communication. It provides the set of underlying assumptions -mentioned in previous sections- about the concepts used in the system. These assumptions are used by the different elements in the system to interchange information consistently. For example, in the case of "ratings" this object defines a full set of categories: "Excellent", "Very-good", "good", "cheap". Each of these include a complete descriptions of features that hotels in the same rank should have in common -like, types of facilities, decoration, # of bathrooms, etc.. These arguments are used by the various guides to map their individual ranks with the ranks of the system. This object is similar to the guide-criteria-object but with a broader scope as it sets the rules for the whole system.

5.5.1.3. The Guide-Manager.

Each hotel guide is represented by an object of this type. This object is responsible for processing any system request to the guide. It has the necessary information on how to translate the system concepts into the specific concepts of the

guide -global schema into local schema. For example, it might know that a system request for "*# of rooms* " should be translated into "*# of units* ". Or maybe, that in order to find "*style* ", it has to look into "*comments*" and "*rating*" for any kind of useful data about it.

It can communicate with the guide-criteria-object which provides information on the assumptions made in defining the data as well as exact data definitions. This object does not perform any actual search on the physical DBMSs. Instead, the guide-manager relies on the guide-interpreter object for this task.

There will be as many instances of this object as hotel guides are used by the system. In our particular application we use 3 guides for hotels in Paris: "Michelin", "Official Hotel & Reservation Guide (OHRG)" and "Fielding's". Therefore, there will be 3 instances of this object, Michelin-guide, OHRG-guide and Fieldings-guide.

5.5.1.4. The Guide-Criteria.

This object knows the specific criteria on which a hotel guide has been written. This object helps the guide-object to find information that is not explicitly stated in the guide. The object consists of elements representing the different aspects of a hotel. Each of this elements contains a clear definition of the concepts used. For example, the "*rating* " element of the criteria records the type of facilities that a hotel has to provide in order to be placed in one or other category -deluxe, superior, etc.. See Figure 5.2.

With respect to rates, it records all the characteristics associated with them, like "*all rates include taxes* " or "*all rates are per person and include breakfast* ". It also

Figure 5.2 : A Subclass of the OHRG Criteria

Unit: SUPERIOR.DELUXE in knowledge base ACCOMMODATIONS
MemberSlot: *COMMENTS from SUPERIOR.DELUXE Inheritance: VERRIDE.VALUES Cardinality.Min: 1 Cardinality.Max: 10 Comment: "Recommendation" Values: (An exclusive and expensive hotel, often palatial, offering the highest standards of services, accommodations and facilities.) (Elegant and luxurious public rooms.) (A prestige address.)
MemberSlot: *FACILITIES from SUPERIOR.DELUXE Inheritance: VERRIDE.VALUES Cardinality.Min: 0 Cardinality.Max: 10 Comment: "Equivalent to facilities in Michelin" Values: A/C HEATING RESTAURANT BAR
MemberSlot: *ROOMS from SUPERIOR.DELUXE Inheritance: VERRIDE.VALUES Cardinality.Min: 0 Cardinality.Max: 10 Comment: "Services in the room" Values: PRIVATE-BATH PHONE TV CABLE MINIBAR
MemberSlot: *STYLE from SUPERIOR.DELUXE Inheritance: VERRIDE.VALUES Cardinality.Min: 1 Cardinality.Max: 3 Comment: "The style of hotels in the category" Values: (Extremely luxurious)

knows what omissions mean in the guide, for example, that omission in mentioning "*air-conditioned*" means "*all rooms have it*".

This object is critical for the system as it helps it understand the real meaning of the data provided by the guides. There should be as many objects of this type as guides are used by the system.

5.5.1.5. The Guide-Interpreter.

This is the technical actor of the system. There should be as many actors of this type as guides are used by the system. Each of them will be assigned to a different DBMS. This actor is the one responsible for accessing the physical DBMS. This actor has enough knowledge about the corresponding Query Language as to build the appropriate queries to retrieve data from the database. It also know the process to establish actual communication with the physical DBMS. This actor is called every time there is a need for retrieving data from the database. Its knowledge is limited exclusively to one DBMS.

Having described the Permanent actors of the system, let us now describe other actors whose role is to support the activities of the system. This group usually comprises general information like scale conversions, i.e. miles --> km., lbs --> kg, etc.. This helps the system to solve some Objective Semantic Conflicts that arise between the guides. It also helps the guide managers to do the translation between local schemas and global schemas. An important example of these actors is the object containing monetary exchange rates. It needs to be updated daily for accuracy and aids the system in calculating and expressing the hotels rates in different currencies.

Another example of supporting actors could be an object containing the Hotel Regulations of the different countries. For example France requires that all rates shown by a hotel include taxes. This is not the case for other countries. Knowing this type of regulations helps the system to make better reports of the data contained in the different guides.

An important advantage of using object-oriented philosophy in representing the knowledge is that any object can be replaced by a new version of it and be incorporated to the system without causing any disruption. The only requisite is that the new object keeps the same protocol of communication with the other external objects.

5.5.2 General Overview of the Interaction between the Actors.

The System Manager is the only object that communicates with the outside world. This object interacts with the user in order to specify the type of information that he wants to obtain from the system. This interaction takes place in the form of questions accompanied by option menus. The user answers by selecting an option from the menu presented.

Once the Director obtains the necessary information from the user, it starts sending messages to the relevant objects. The first objects called are the guide-managers. In the messages, the Director asks each guide-manager to find information relevant to some specific data. The usual type of message will include the hotel's name and the questioned item(s). The guide-manager delegates in the guide-criteria and the guide-interpreter for accomplishing these tasks.

Once all the guide-managers have presented their reports, the System-manager starts solving the Objective Semantic Conflicts between them. Finally, it starts evaluating the different opinions and thus, solving the Subjective Semantic Conflicts. At the end, the findings are presented to the user together with their likelihood of occurrence.

5.6. Semantic Conflicts in the Context of Hotel Guides.

According to our previous discussion, achieving effective communication implies relying on a common language that all elements in the system can use to understand each other. The mapping from the individual languages -local schemas- onto this common language -global schema- involves the solving of semantic conflicts at different levels. Our decentralized object-oriented design implies that many objects will be involved in this task of solving semantic conflicts.

We will now draw upon the Taxonomy of Semantic Conflicts developed in chapter 3, by applying it to the context of Hotel Guides.

5.6.1. Taxonomy of Semantic Conflicts.

We will illustrate the different Semantic Conflicts that appear in this context by showing actual examples for each one of the categories:

5.6.1.1. Objective Conflicts.

These involve the simplest type of conflicts in terms of approaches to solve them. These occur at two points. First, between global and local schemas and is handled at the guide-managers level. All these decisions are taken by the guide-manager relying on the knowledge recorded in the guide-criteria.

Objective Conflicts also occur at the System Manager level. This time the problem arises between the different databases. The problem appears at the time when the System Manager starts comparing the assertions of the different guides. These conflicts have to be solved before attempting to solve any Subjective Conflict. All these decisions are taken by the System Manager using the knowledge recorded in the system-criteria as well as in each guide-criteria.

They could be further divided in:

Syntax: It is very common to find each guide using its own terminology. For example, one guide may talk about "facilities of a hotel" and refer to things like, restaurant, parking, pool, etc.. Another guide may refer to the same items as "services offered by a hotel". Same thing happens with concepts like, "rates" and "prices", "quality" and "rating". These show probably the most basic conflict that we could find while comparing different hotel guides.

Sometimes, this involves more than just a switching of words; the manager needs to know about relevant data to support a particular request for information. For example, a request for "quality" could be translated by one guide into queries for "rating", "special-features" and "view".

Ambiguity: This is the case where the same word are used with different meanings. Just building up on the previous example, there may also be the case in

which two guides have a field called "services". However, a closer look would reveal that while one is referring to items like those mentioned before, the other one is referring to special services offered by the hotel like, "theater-booking", "dry-cleaning", etc.. The important point here is that a negligent comparison of those would lead to terrible mistakes and thus, senseless answers. It is important to find which is the actual concept of the guide that refer to the same kind of elements in order to do sensible comparisons.

The most important case of ambiguity probably happens with respect to concepts that involve numerical units. It would obviously be an unforgettable error to try to make comparisons, or even worse, mathematical manipulations -i.e. average-between numbers reported by different guides, without taking the time to determine the actual meaning of the numbers. It can be that in one guide, rates include taxes and breakfast while in the other guide rates just reflect the net cost of a room. It could also be that rates in one guide are represented in dollars while in the other, rates are represented in francs.

Distance from airport is also a common source of conflict as some guides will express them in kilometers while others in miles. For a person that does not realizes the difference in units, a selection between two hotels based on distance to airport may bring unpleasant surprises.

5.6.1.2. Subjective Conflicts.

This type of conflict involves differences in opinions among the guides. The System Manager is the actor responsible for solving them. The conflicts arise at the

end, when the Manager is evaluating the reports presented by each guide -and the corresponding Objective conflicts have been solved. The Subjective Semantic Conflicts will be solved by using the problem-solving strategies defined in chapters 3 and 4. They are: 1) Comparison of evidence; 2) Credibility of sources; 3) Search for complementary data; and 4) Inferencing from Similar data.

These problem-solving strategies are captured in various sets of heuristics.

According to the general Taxonomy, we can identify two categories in which to group the Subjective Semantic Conflicts. Again, I will present examples in the context of Hotel Guides:

Contradiction: This is the most difficult problem to reconcile as it involves some sort of subjective approach to solve it. This happens when the guides report values that contradict each other. For example, what if one guide says that "hotel X has parking" while another says that "hotel X has no parking" ? What can we say about parking in the hotel ?

Another case of contradiction is when the values given by the various guides differ to a certain extent. A very common example happens in the context of "rates". For instance, one guide asserts that the rates for double-rooms in hotel X are \$35 and \$45 -for the least and best room respectively. Another guide reports for the same item a price of \$50 and \$58. These conflicts usually require a combination of the problem-solving strategies defined in chapter 4, to solve them.

Incompleteness: What if one guide asserts that "rooms have *air-conditioned*" and the other two guides say nothing ? Can we conclude that the rooms have "air-conditioned" ? A combination of strategies will be usually needed. A very

important strategy, however, is to try finding relevant data. For example, in the case of a high-quality hotel, lack of information about "*air-conditioned*" may not be very relevant, as we can assume that a hotel of such category has "*air-conditioned*". The same assumption should not be made with respect to a hotel of a lower category. So in the latter case, lack of information could weight negatively in concluding whether the hotel has such feature. In this example the "rating of the hotel" helps to overcome the lack of data.

5.7 Where is the intelligence in this system ?

The intelligence of this system is located in all the places where non-trivial decision have to be taken. If we were to group them from an overall point of view, we would do as follows:

1) Knowledge and reasoning are needed when the System Manager has to decide whether to ask all guides for some information or discriminate among them. In making the right decision, the object needs to know which is the specific usefulness of each guide. For example, *guide Y is very good for rates* while *guide Z will not help at all*; or *pay special attention* -assign more weight- to comments in *guide T*. This is done by using applying heuristics represented in rules.

2) Knowledge and reasoning are also needed at the guide-manager level; when it has to decide what type of data to gather from the database. This is, given a request for "quality", to know which other data from the database could be useful to the system in its task of finding the information requested by the user. This is done by executing specific procedures.

3) Knowledge and reasoning are needed at the end, when the system wants to compose the individual information provided by the various guides in a sensible and useful answer for the user. At this point, it has to make the best decisions based on what the information mean and what the relevance of each piece is. This is done by using applying heuristics represented in rules.

5.8 Implementation.

A real implementation in the domain of Hotel Guides was done with the purpose of testing the feasibility of the framework proposed. This included testing the design principles for organizing the system and identifying the contributions of this approach with respect to conventional methods. Probably the main goal was to demonstrate the feasibility of solving Objective and especially, Subjective Semantic Conflicts product of the integration of heterogeneous sources of information. In the next sections, we explain the environment used to host the application and present some sample sessions to illustrate the behavior of the system.

5.8.1. Environment.

In this section we describe the tools used for building the system in the proposed framework.

5.8.1.1. Hardware.

The computer equipment used consists of a Lisp machine: XEROX 8000.

5.8.1.2. Software.

The Software consists of a Knowledge Base System environment named **Kee** from **IntelliCorp**. This software interacts with **InterLisp** to provide for a very powerful programming environment. This software is different from many of the Shells available in the market in the sense that it provides an Object-oriented environment for applications development. Other shells provide just for the use of Rules to capture all the knowledge of the system. The latter presents a major drawback because as the set of rules increases so does the complexity of the system. They can soon become monsters extremely difficult to debug and maintain.

The possibility of using Objects -frames- and Rules provides the necessary flexibility for selecting the most suitable approach for knowledge representation depending on the characteristics of it. For example, factual knowledge might be better captured in frames composed by attributes and values. By contrast, knowledge about the different processes involved in problem-solving approaches may be better described by heuristics thus, better captured by means of rules. The ability of **Kee** to perform Forward and Backward chaining also provides great flexibility in the design of approaches for addressing problems. This flexibility allows the designer to describe problem-solving heuristics in the most convenient way.

Again, it allows to program a system according to the designer's perspective as opposed to the machine perspective.

Kee's interface with **InterLisp** allows for the attachment of complex Lisp procedures -methods-, to objects. This increases the power of the system as Lisp can be used to develop all kinds of sophisticated procedures.

Kee also supports inheritance. This is very useful because it allows for the building of structures that capture the relationships among attributes.

Finally, I must admit that the process involved in learning Kee is quite painful as the manuals are not well written for beginners. Once, the first obstacles are surpassed, the learning curve is upward sloping. I consider Kee to be a very powerful environment for hosting applications involving Knowledge-Base approaches.

5.8.2. Knowledge Representation.

As we mentioned in several occasions, there will be 3 ways in which to represent the knowledge: 1) Objects-attributes and values, 2) Methods or procedures attached to the objects, and 3) Rules.

The first type, namely actors, was already explained in detail. We will now take the time to describe Methods and Rules.

5.8.2.1. Methods.

Methods are procedures that can be attached to the slots of an object or actor. These are basically sequential programs describing a particular process. These are used throughout the system to address those problems that are fully specified. We will now present some examples that illustrate the type of problem that is addressed by using procedures. For example, every time the system starts a new session, some slots around the system have to be reset. The necessary steps to make this happen are put together in a program. Every time that the system receives a user's request, it should pass it to the guides for processing. The message passing process will also be encoded

in a Lisp program. On the other hand, the guide that receives the message should react accordingly. The steps involved in serving the message are also encoded in a Lisp program attached to one of the guide-manager's slot. In our particular case, given that Kee interfaces with InterLisp, this will be the language used to write the programs. We will now present the different kinds of programs written.

At the system level, 3 main programs are written and there are all attached to slots of the System Manager. The first is called RESET and is responsible for preparing the system for a new session. This program basically consists of statements to ask the guides to reset themselves. This program uses the services of another program called UNTIMSG that does the actual communication.

The second program is called WELCOME and is in charge of the interface with the user. It presents menus and records the answers. The result produced by this program will become the request to be sent to each guide.

The third program is called WAKE-UP. Its purpose is to distribute the user's request among the guides. The request is transmitted by sending key words as parameters. There can be two types of requests, general and particulars. The former asks each guide to provide options for hotels that match certain parameters. The second gives the name of a particular hotel together with some attributes of particular interest for the user. The task of the guide is to find as much information as possible about them. We implemented the latter case of requests.

At each guide community of objects, namely guide-manager, guide-criteria and Guide-Interpreter, more methods are attached. Although for each guide the programs are customized to address specific problems or cases of the guide, the basic structure

and function of each program is the same across guides. We will refer to Michelin examples but they can be safely generalized to the other guides.

Starting with the guide-manager, we have a method called SELECT. This program is very important as it is in charge of serving the system request for information. First thing this program does is to check the parameters sent. The first should be the name of a specific hotel, and the rest should be the attributes to be found. As they are expressed in the global schema terms, the guide-manager has to check whether mapping into the local schema is needed or not. For example in the case of the Michelin guide, the name of the concept that expresses information about Rooms is different between global and local schema - a conflict of Syntax. In this case, if one of the parameters sent is Rooms, the guide-manager should translate it in the local name: Room.Services.

See Figure 5.3 for an actual example of the program SELECT; this particular method is attached to the Michelin guide-manager. Once the program has solved the Syntax problems, thus, translated the global schema into the local schema, it starts delegating tasks on the other members of the group. It sends a message to the Guide-Interpreter, requesting to retrieve the appropriate data for the particular hotel (see RETRIEVE below). It will also ask for the "quality" of the hotel to be obtained. The value of "quality" is sent to the guide-criteria together with the attribute to be found (see INFORM.ME below). The guide-manager asks both actors to deposit their findings in one of its slots called ANSWER.

This slot has the ability to check the data before receiving it. Hence, it makes sure that data is not repeated and that just the appropriate information is accepted. The latter can be accomplished because the guide has defined a class of elements that belong

Figure 5.3 : An Example of a Method.

(OUTPUT) The SELECT slot

```

OwnSlot: SELECT from MICHELIN.GUIDE
Inheritance: (METHOD in KB KEEROLES)
ValueClass: (METHOD in KB
KEEDATATYPES)
Comment:
  "A command for retrieval of information"
Facet Inheritance: OVERRIDE
Values:
ACCOMMODATIONS>MICHELIN.GUIDE::SELECT!method
  [LAMBDA (MICHELIN.GUIDE USER.HOTEL.NAME
            ITEM)
    (PUT.VALUE (QUOTE MICHELIN.GUIDE)
                (QUOTE NEW.HOTEL.NAME)
                USER.HOTEL.NAME)
    (COND
      ((EQUAL ITEM (QUOTE ROOMS))
        (PUT.VALUE (QUOTE MICHELIN.GUIDE)
                    (QUOTE QUESTIONED.ITEM)
                    (QUOTE ROOM.SERVICES))
        (SETQ ITEM (QUOTE ROOM.SERVICES)))
      ((PUT.VALUE (QUOTE MICHELIN.GUIDE)
                  (QUOTE QUESTIONED.ITEM)
                  ITEM)))
    (SETQ RAT (GET.VALUE USER.HOTEL.NAME
                        (QUOTE QUALITY)))
    (PUT.VALUE (QUOTE MICHELIN.GUIDE)
                (QUOTE QUALITY)
                RAT)
    (UNITMSG (QUOTE MICHELIN.CRITERIA)
              (QUOTE INFORM.ME)
              RAT ITEM (QUOTE
                        MICHELIN.GUIDE))
    (UNITMSG (QUOTE MICHELIN.ORACLE)
              (QUOTE RETRIEVE)
              USER.HOTEL.NAME ITEM
              (QUOTE ANSWER])

```

to each concept. Anything that is not in the class will be rejected. There are many slots like this throughout the different objects of the system. They are worth mentioning because they contribute in eliminating some objective conflicts, especially of syntax as well as redundancy.

The criteria object has another method called INFORM ME. The criteria has several subclasses. Each subclass represents one particular category of hotel and comprises all the assumptions made about it. This means all the characteristics that a hotel should have in order to be assigned to such rank. The characteristics are grouped by fields, like facilities, rooms, rates, style, etc. When the method INFORM ME is called, it selects the sub-class that matches the "quality" sent in the message. Then it checks for the item requested, like "facilities" and send the content of the slot back to the guide-manager.

The DBMS- interpreter also has a method attached and is called RETRIEVE. This method expects a list containing the fields to be retrieved from the database. This method is in charge of building the appropriate query to be sent to the physical DBMS. In the case of Michelin, the guide is stored in an ORACLE DBMS. Then to a list of parameters like (Madeleine-Place facilities ANSWER), it will produce a query like the following:

```
SELECT Madeleine-Place FROM ACCOMMODATIONS
```

```
WHERE FIELD = facilities
```

(ACCOMMODATIONS is the name of the database).

Apart from these methods there are many small programs aimed to facilitate some tasks across the system or to perform very specialized functions. An example of the latter is a program called CRITERIA TRIGGER. This method is strategically placed in the a slot of the OHRG-manager object where the value of quality is returned by RETRIEVE. As soon as the value of the slot changes, the CRITERIA TRIGGER program is activated. This program follows the necessary steps to send a message to the guide-criteria with the value of quality. This especial method covers for the guide-manager task of sending this message. The advantage is that in the case of the guide-manager, it will have to wait or keep checking to see whether RETRIEVE has already sent the value of "quality". The CRITERIA TRIGGER, on the other hand can react automatically without causing any other delays.

This type of methods are called Active values because their function is to monitor certain slots and react accordingly. They can react when a value is added, deleted, or retrieved from the slot. These are very useful in the sense that they can automatically trigger the execution of programs. For example, every time that a value is added to a particular slot, a program is executed that checks for all the accepted values and make the appropriate corrections of syntax. This is used in the program for mapping the local schemas into the global schemas.

This ends our discussion about methods.

5.8.2.2. Rules.

Rules are the other mechanism that the system uses to represent knowledge. It should be said that one of the most important features about Knowledge-Base Systems

is their capabilities for explaining the conclusions made by the system. At the end of a session, the user has the opportunity to ask about the rules used in making a conclusion. This transparency is hardly provided by conventional programming techniques. The line we have adopted in this implementation is to use Rules to represent knowledge that involves heuristics. Rules allow us to exploit the "inferencing capabilities" of the system. In this sense, we have reserved its use for the representation of the strategies to solve Subjective Semantic Conflicts among different guides. We will present here the set of rules created for solving a subset of the conflicts reported with respect to facilities. We will present the rules developed just for solving PARKING conflicts. We use Forward Chaining for firing these sets of rules.

The rules can be divided in 3 groups: 1) Rules to compare the different assertions made by the 3 guides; 2) Rules to clarify conflicts -using relevant data-; and 3) Rules to present the information to the user.

The first set of rules does a careful comparison of the assertions made by the guides. The possible assertions are **Parking**, **No Parking** and **don't know**. The credibility concept is embedded in the rules. In this way, more credibility will always be given to the assertions made by the Michelin guide and the Fielding's guide as opposed to OHRG. However, OHRG's opinions may help to solve a conflict between Michelin and Fielding's. It is also important to mention that **don't knows** will be treated differently depending on the guide. In this sense, a **don't know** from Michelin will be considered closer to a negative rather than a positive assertion. This is done because Michelin is usually very accurate in mentioning all the facilities included. On the other hand, a **don't know** from OHRG or Fielding's will be almost ignored. The reason behind is that they are not so strong about being exhaustive in their

description - although in the case of Fielding's any assertion is very accurate. See Figure 5.4 for an English translation of the actual Lisp version of these Rules.

As a result of this evaluation of the assertions, the system defines a likelihood of occurrence. This means that having taken into account the credibility and accuracy factors for each guide, the system derives a number (between 0 and 1) that expresses the chances that the specific hotel has for offering Parking facilities.

The system also assigns a value, Confidence Factor that measures the degree of conflict among the guides. In this sense, if the system makes a confident decision the Confidence Factor is assigned a value of Ready. This means that no further evaluation is needed. If on the other hand, there was enough conflict among the guides the system assigns to the Confidence Factor a value of Doubtful. This invites the system to try other strategies in order to clarify the conflict.

The second group of rules apply to those cases when the Confidence Factor (CF) of a final assertion has a value of Doubtful. In this case, the strategy that the system follows is to ask for relevant data that could clarify the conflict. In the case of Parking the most relevant data would be Location. Fielding's guide actually records location so the system asks the guide to find the value for it. The values for location can be Downtown, Suburb, Express-way and Resort.

Then, the rules consist of checking this value and thus, decreasing or increasing the previous Likelihood factor calculated for Parking in the hotel. In the case of Downtown the value will be lowered by 20 % because the system knows that Parking in city downtowns is very difficult. On the other hand, if the hotel is located in a Suburb, the Likelihood factor is increased by 20 %. If the hotel is located in an

Figure 5.4 : An Illustration of a Typical Set of Rules

```

DEBIT OF UNIT FACILITIES RULES

RULE F4005
  (IF (NOT EQUAL (QUOTE NO-PARKING)
    (MEMBER (QUOTE NO-PARKING)
      (SET-VALUES (QUOTE FIELDINGS GUIDE)
        (QUOTE ANSWER))))
    AND (NOT EQUAL (QUOTE PARKING)
      (MEMBER (QUOTE PARKING)
        (SET-VALUES (QUOTE
          MICHELIN GUIDE)
          (QUOTE ANSWER))))))

  THEN (THE DEBIT OF PARKING IN HOTEL IS 2)
        AND (THE CONFIDENCE OF PARKING IN HOTEL IS READY)

RULE F4006
  IF (EQUAL (QUOTE PARKING)
    (MEMBER (QUOTE PARKING)
      (SET-VALUES (QUOTE MICHELIN GUIDE)
        (QUOTE ANSWER))))
    AND (NOT EQUAL (QUOTE PARKING)
      (MEMBER (QUOTE PARKING)
        (SET-VALUES (QUOTE
          FIELDINGS GUIDE)
          (QUOTE ANSWER))))))
    AND (NOT EQUAL (QUOTE NO-PARKING)
      (MEMBER (QUOTE NO-PARKING)
        (SET-VALUES (QUOTE
          FIELDINGS GUIDE)
          (QUOTE ANSWER))))))

```

Figure 5.5 : An Illustration of a Typical Set of Rules

```

DEPT of Unit FACILITIES RULES

RULE F4002
IF THE CONFIDENCE OF RESTAURANT IN HOTEL IS READY)
AND
  SEC CAR LAST CAR OVER QUOTE (THE OF OF
                                RESTAURANT IN HOTEL
                                IS READY)
  THEN WILL DO PRINT
  ***** There is at least one RESTAURANT in the hotel *****

RULE F4007
IF THE CONFIDENCE OF PARKING IN HOTEL IS READY)
AND
  SEC CAR LAST CAR OVER QUOTE (THE OF OF
                                PARKING IN HOTEL
                                IS READY)
  AND
  SEC CAR LAST CAR OVER QUOTE (THE OF OF
                                PARKING IN HOTEL
                                IS READY)
  THEN WILL DO PRINT ***** Although not certain, there is

RULE F4011 IF THE CONFIDENCE OF PARKING IN HOTEL IS READY)
AND THE OF OF PARKING IN HOTEL IS 50%
THEN THE CONFIDENCE OF PARKING IN HOTEL IS
  READY
  DO
  PRINT ***** There is controversy among

```

Express-way or Resort, the chances of Parking are even higher so the Likelihood factor is increased by 25 %. The system does not know about any other strategy to clarify conflicts about Parking so a value of Ready is assigned to the Confidence factor. This is done as a way to tell the system to stop trying other rules. .

The final set of Rules decides what type of answer should be given to the user, according to the final Likelihood factor (LF) of the assertion. In this sense, if the $LF > .85$ the system assures that the hotel has Parking. If on the other hand, $.7 < LF < .84$, the system tell the user that *there are great chances for Parking in the hotel*. In the same lines, if $.4 < LF < .6$, the system tell the user that *there is too much controversy among the guides, thus it can not make any conclusion*. See figure 5.5 for a sample of these rules.

First Group of Rules: Confronting the different data taking into account the Credibility factor.

- 1) IF answer of Michelin is Parking and
 answer of Fieldings is Parking and
 answer of OHRG is Parking
 THEN LF of Parking = 1 and
 CF of Parking = Ready

- 2) IF answer of Michelin is No Parking and
 answer of Fieldings is No Parking and
 answer of OHRG is No Parking
 THEN LF of Parking = 0 and

CF of Parking = Ready

- 3) IF answer of Michelin is Parking and
 answer of Fieldings is Parking and
 answer of OHRG is No Parking
THEN LF of Parking = .8 and
 CF of Parking = Ready
- 4) IF answer of Michelin is Parking and
 answer of Fieldings is No Parking and
 answer of OHRG is Parking
THEN LF of Parking = .6 and
 CF of Parking = Doubtful
- 5) IF answer of Michelin is No Parking and
 answer of Fieldings is Parking and
 answer of OHRG is Parking
THEN LF of Parking = .6 and
 CF of Parking = Doubtful
- 6) IF answer of Michelin is Parking and
 answer of Fieldings is No Parking and
 answer of OHRG is No Parking
THEN LF of Parking = .4 and
 CF of Parking = Doubtful
- 7) IF answer of Michelin is No Parking and
 answer of Fieldings is No Parking and

answer of OHRG is Parking

THEN LF of Parking = .2 and

CF of Parking = Ready

8) IF answer of Michelin is Unknown and
 answer of Fieldings is Parking and
 answer of OHRG is Parking

THEN LF of Parking = .8 and

CF of Parking = Ready

9) IF answer of Michelin is Unknown and
 answer of Fieldings is Parking and
 answer of OHRG is No Parking

THEN LF of Parking = .6 and

CF of Parking = Doubtful

10) IF answer of Michelin is Unknown and
 answer of Fieldings is No Parking and
 answer of OHRG is Parking

THEN LF of Parking = .4 and

CF of Parking = Doubtful

11) IF answer of Michelin is Unknown and
 answer of Fieldings is No Parking and
 answer of OHRG is No Parking

THEN LF of Parking = .2 and

CF of Parking = Ready

Second Group of Rules: Using relevant data for clarifying a conflict.

- 1) IF CF of Parking = Doubtful and
 Location of Hotel = Downtown
 THEN LF of Parking = LF of Parking * .75 and
 CF of Parking = Ready
- 2) IF CF of Parking = Doubtful and
 Location of Hotel = Suburb
 THEN LF of Parking = LF of Parking * 1.20 and
 CF of Parking = Ready
- 3) IF CF of Parking = Doubtful and
 Location of Hotel = Express-way or Resort
 THEN LF of Parking = LF of Parking * 1.25 and
 CF of Parking = Ready

Third Group of Rules: Presenting the information to the user.

- 1) IF CF of Parking = Ready and
 LF of Parking <= .24
 THEN *** There is no Parking in the hotel. ***
- 2) IF CF of Parking = Ready and

LF of Parking $\leq .44$ and LF of Parking $\geq .25$

THEN *** The chances of Parking in the hotel are very low. ***

3) IF CF of Parking = Ready and

LF of Parking $\leq .64$ and LF of Parking $\geq .45$

THEN *** There is too much controversy among the guides,
I cannot make any conclusion ***

4) IF CF of Parking = Ready and

LF of Parking $\leq .74$ and LF of Parking $\geq .65$

THEN *** Although not certain, there may be Parking in the hotel. ***

5) IF CF of Parking = Ready and

LF of Parking $\leq .79$ and LF of Parking $\geq .75$

THEN *** There are great chances for Parking in the hotel. ***

5) IF CF of Parking = Ready and

LF of Parking $\geq .8$ THEN

THEN *** There is Parking in the hotel. ***

Finally , a brief discussion of explanation capability.

5.8.3. Interacting with the system

In this section, I present some examples of the system behavior in various cases. The purpose is to illustrate different aspects of the problem based on the

experience gained from the case study. The examples present different kinds of semantic conflicts and different ways of dealing with them.

Some assumptions are made about the physical databases. In this sense, the Michelin guide will be assumed to be stored in an Ingres DBMS; the OHRG in DBase-3 and Fielding's in Oracle. This means that each object-guide should know how to produce the appropriate queries to communicate with its database.

5.8.3.1. Sample Session 1:

The case of a request about "facilities of a hotel".

This case presents some interesting issues that highlight the usefulness of using a knowledge-base approach. Basically, how to understand and infer information from data that is not explicitly stored in the databases. This will require interaction with the guides' criteria to understand the semantics of omissions and simplification of concepts.

Start:

- 1) Do you want some information about a particular hotel ?
YES

What is the name of the hotel ? Madeleine-Place

Please select the item that corresponds to the kind of
information you are interested in:

Facilities (pool,elev.,etc)	Rates	Rooms	Quality	Location	Comments

==> Facilities

(this item is selected by clicking on the menu)

(see Figures 5.6 and 5.7 for examples of the actual menus used in the system)

Inside the system ...

The "System Manager object" sends a message to each of the guide-managers:
Michelin-guide, OHRG-guide and Fieldings-guide.

The message contains 1) an operation code, 2) the hotel's name, and 3) the item selected. For example:

Select "Madeleine-Place " "facilities"

1

2

3

(Select is the generic name for the methods in charge of servicing the request in each guide)

Now, this actor waits for the guide-managers to respond.

In the Michelin-guide:

Figure 5.6 : The WELCOME Menu

The image shows a graphical user interface window titled "welcome". Below the title bar, it says "HOTEL IMAGES'S TEXTSLOT". The main area of the window contains the following text:

WELCOME TO THE EXPERT HOTEL FINDER

I heard that you're planning a trip .

May I suggest PARIS ?

At the bottom of the window, there is a section labeled "choose:" which contains two buttons: "YES" and "NO".

Figure 5.7 : The SELECTION Menu

WELCOME'S EXPLANATION

Please select from the menu below, the item that corresponds to the information you want about the hotel:

WELCOME'S INTRODUCTION

FACILITIES	ROOMS	STYLE	COMMENTS	RATES

As soon as Michelin-guide receives the message from the System Manager, it sends a new message to the Oracle-Interpreter to produce the appropriate query to be sent to the physical Oracle DBMS. In the query, the object asks the database to return the value of the "facilities" slot as well as the "rating" slot that correspond to the hotel's name. The "rating" concept groups hotels according to their quality. This value will be used for inquiring the Michelin-criteria.

In order to be accurate, Michelin-guide has to check with the guide criteria for any especial meaning of the field "facilities" like, omissions and default values. The criteria records this information by hotel categories. Thus, Michelin-guide should send the rating of the hotel together with the concept "facilities". For example, the 5* (5 stars) subclass of Michelin-criteria contains those facilities that should always be offered by such type of hotels. Same thing with the rest of the categories. This is the reason why Michelin asks the Interpreter to retrieve the "rating" of the hotel from the DBMS.

The Michelin-Interpreter retrieves two values for facilities: **Conference Rooms** and **Pool**. For the value of "rating" it returns **4***.

The value of "rating" for the Madeleine-Place hotel is sent to Michelin-criteria. In this case, Michelin-criteria responds by providing some default values. These are: **Lift, Rest, and Parking**. These default values should be combined with the answer produced by the database.

Before asserting the new data into the system, Michelin-manager has to work on making it compatible to the system schema. For example, translating "rest" for "restaurant", and "lift" for "elevator". Once this is done, Michelin-manager can

proceed to send the information collected to the System Manager. The actual information will look like:

Elevator, Restaurant, Parking, Conference.Rooms, Pool.

In the OHRG-guide:

When this actor receives the message, it also proceeds to call its guide-Interpreter to build the appropriate query to be sent to its DBMS. However, this time it has to make a conversion between "facilities" and the word that OHRG uses to refer to it, in this case "services".

The consultation with the criteria-object, could not be necessary or the same, since each guide-manager knows which fields or concepts are worth-while checking (this knowledge is represented in the methods attached to the object). However in the case of "services" -or facilities-, OHRG-criteria also has insightful information about default values. Hence, OHRG-manager sends messages to the Interpreter to find both, data about services, and the "rating" of the Madeleine-Place according to OHRG.

The guide-Interpreter access DBase-3 and retrieves the following values for "services": **Pool, No.Parking**. For "rating" it reports: **Deluxe**.

Now, OHRG-manager sends a message to the OHRG-criteria with the **Deluxe** value. The criteria checks in its frame for Deluxe hotels and finds two default values which sends back to OHRG-manager. These values are: **Restaurant, Elevator**.

No semantic mapping is needed in this case, thus, OHRG-manager can report the new data to the system. This consists of:

Pool, No.Parking, Restaurant, Elevator.

In the Fieldings-object:

When this object receives the message, it knows that the guide does not have specific data for "facilities". However, it knows that it could find some information from Fieldings-criteria (this is encoded in the Select method attached to the guide). See Figure 5.8. In fact, it finds that ratings of hotels in this guide actually depend on the facilities offered by the hotel. The values are assumed to be a necessary condition for hotels to be assigned into one or other rank, thus no more data is offered in the database about it. In this case, it then needs to know which is the "rating" assigned to the specific hotel. Therefore, such is the message it sends to its Ingres-Interpreter. The Interpreter in turn, communicates with the physical DBMS asking for retrieval of the corresponding "rating" of the "Madeleine-Place" hotel. Once it has this data, it sends it to Fieldings-manager. The latter builds a new message composed by the "rating" obtained from the database that is: **4\$** -according to the ranking used by Fieldings-, and the field "facilities". Fieldings-criteria interprets this message as: *Retrieve the slot "facilities" from the frame "4\$"* . This means find out which are the facilities that according to the guide, correspond to a hotel of such category.

Fieldings-criteria returns the following values:

Restaurant, Bar.

Figure 5.8 : A Subclass of the FIELDING'S Criteria

Unit: FIELDINGS.CRITERIA in knowledge base ACCOMMODATIONS
MemberSlot: *CLOSEST.T.STATION from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES
MemberSlot: *FACILITIES from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES
MemberSlot: *LOCATION from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES Values: Unknown
MemberSlot: *NUMBER.OF.UNITS from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES ValueClass: INTEGER Cardinality.Min: 0 Cardinality.Max: 1
MemberSlot: *OBSERVATIONS from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES Comment: "Insightful comments about the hotel"
MemberSlot: *PRICE.CATEGORY from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES ValueClass: (ONE.OF 1\$ 2\$ 3\$ 4\$ 5\$) Cardinality.Min: 1 Comment: "Hotel category according to price"
MemberSlot: *PRICES.NEWS from FIELDINGS.DBASE3 Inheritance: OVERRIDE.VALUES Comment: "Information about taxes, service-charges, breakfasts, etc. that may be included in the hotel rates"

Now, Fieldings-manager can report the new information to the System Manager.

Meanwhile, the System Manager waits until all guides are done.

Once this happens, the System Manager prepares itself for a new set of problems: interpreting the data asserted by the various guides in order to produce sensible and useful information from it. Then, the basic question is how to build a unified view of the conflicted information; or, if this is not possible, how to present the conflicts to the user.

There are many possible scenarios. The case we have now is the following:

Michelin: Elevator, Restaurant, Parking, Conference.Rooms, Pool.

OHRG: Pool, No.Parking, Restaurant, Elevator.

Fieldings: Restaurant, Bar.

The System Manager has to evaluate the different assertions made for each field. The easiest case of all is Restaurant that is asserted by all the guides. For the other items, the solution is not that easy. In these cases the System has to recur to other strategies. A very important one is Credibility of the sources. As it was explained in chapter 4, the System should know which are the strengths and weaknesses of each guide for each particular field. This means that the System might tend to believe in some guides more than in others depending on the issue in question. In this case the

System knows that Michelin is much more careful than OHRG with respect to information about "facilities" (this knowledge is captured in rules). On the other hand, it knows that although Fieldings do not specifically talk about the facilities for each hotel, it is very strict in assigning ranks to the different hotels. This is, for a hotel to be in a particular rank it has to fulfill all the requirements of such category.

Let us now illustrate the behavior of the System in addressing the conflicts of opinion among the guides. For reasons of clarity, we will pick one of the elements asserted by the guides to be further developed. In this way the reader can see the different details of the process involved. Let us use the example of Parking in the hotel.

Focusing now on the item in question, we look at the opinion of each guide:

Michelin: Parking

OHRG: No Parking

Fieldings: ----- (don't know)

According to the Credibility of the guides, Michelin and Fieldings are more accurate with respect to facilities. In this sense, Fieldings does not help very much. On the other hand, the System cannot just ignore the fact that OHRG is denying Parking availability. How could it solve this conflict ?

We will now explain the approach we use in our implementation. It is important to say that the following approach is embedded in the set of rules. On one side, we assign a Credibility factor between 0 - 1, to each of the guides for each one of

the fields. The Credibility factors should add up to 1. In the case of facilities we assume the following:

Michelin = .4

OHRG = .2 and

Fieldings = .4.

On the other side, we assign values to the assertions made, again in a range of 0 - 1. The possibilities are: positive assertion, negative assertion and don't know. These are assigned, values of 1, 0 and .5, respectively.

For example, an assertion of **Parking** will obtain a 1. An assertion of **No Parking** will obtain a 0, and don't know will obtain a .5.

Then we make a weighted average calculation as follows:

$$LF(y) = \sum_i C_{ix} * A_{iy}$$

where: C_{ix} = Credibility of guide i with respect to field X.

A_i = value of the assertion made by guide i with respect
to element y of X.

$LF(y)$ = Likelihood Factor or Probability for y.

($0 < i \leq \#$ of available guides).

In our example:

$X =$ Facilities

$y =$ Parking

according to our notation, we rewrite the Credibility of each guide as:

$C_{\text{Michelin, facilities}} = .4$

$C_{\text{OHRG, facilities}} = .2$

$C_{\text{Fieldings, facilities}} = .4$

Then, the result is:

$$LF(\text{parking}) = (.4 * 1) + (.2 * 0) + (.4 * .5) = .6$$

This result is slightly higher than .5 that represents don't know and should not surprise anyone given the conflict of opinion by the different guides. If the result had been strong ($LF(y)$ lower than .3 or higher than .7), the system might have accepted it. However, in this case where the result is still uncertain, the system recur to other strategies to see if it could reenforce its confidence about the issue.

The strategy that the system will follow, now, is to find other relevant data that could help it clarifying the uncertainty. In the case of Parking, knowing about the location may be useful. For example, if the hotel is located in Downtown, the possibilities of having Parking are low. Yet, if the hotel is located in the suburbs or maybe in a resort, or close to an Express-way, the possibilities increase.

The system knows that Fieldings guide has information about Location, so it sends a message to the Fieldings-manager to arrange a search for location of the hotel. Fieldings-manager sends the corresponding message to the Interpreter, and later, the information received to the System Manager.

There has to be a mechanism to incorporate the information provided by the Relevant data into the calculation performed. We do it by assigning a ratio to the value of such data -this can be done by grouping the values-, and then multiplying it by the calculated probability.

In our case, we say that the percentage of influence that the new value can have over the calculated result is 20 %. If the value of Location is Downtown then we lower the probability calculated before in 20 %. This means that the new probability is:

$$\text{New LF}(\text{parking}) = .6 * .8 = .48$$

If on the other hand, the location of the hotel is any other than Downtown, namely, Suburban, Resort or Express-way, the probability calculated before can be increased by the same percentage. Then,

$$\text{New LF}(\text{parking}) = .6 * 1.2 = .72$$

In the first case, the probability of Parking is slightly lower than .5, thus, very uncertain. The System will have to tell the user:

*** There is controversy among the guides about

Parking in the hotel, I cannot conclude anything ***

The user should have the possibility to ask the system about any of its conclusions. In the event that he asks to know more about Parking, the system should show the following information:

Michelin: **Parking**

OHRG: **No Parking, and**

Fieldings: ----- (says nothing)

Besides, the system has gathered that:

Location of hotel is **Downtown**

In the second case, the probability of Parking is fairly high: .72. Therefore, the System can tell the user:

*** There are good chances of Parking in the hotel ***

The purpose of this example has been to illustrate the behavior of the system for solving Subjective Semantic Conflicts. Although for clarity reasons, we picked one case to develop, namely, Parking, the approach can be generalized to any other aspect of "facilities in a hotel". The same types of calculations will apply, and there will almost always be some relevant data to manipulate. The latter, of course, will vary according to the questioned element.

An example of the overall information presented to the user follows:

The Madeleine-Place hotel has the following facilities:

Restaurant

Coffee-Shop

Elevator

A/C & Heating

There are good chances for Parking.

5.8.3.2. Sample session 2: A Request about "Rates"

This case presents other kind of semantic problems that frequently arise when one interacts with different sources or databases. This is the problem of ambiguity. Specifically, same concepts yet, different meanings. A very common example is the case of some guides showing "rates" in dollars and some others in francs. Also, the case of a guide showing "rates" "per room" and other guide showing them "per person" and so on. "Rates" is probably the field that presents the greatest amount of variety, as each guide seems to present it in a different way. Without an explicit criteria that define the concepts in each guide it becomes very difficult to get any sensible meaning from any combination of data among guides.

Sample session:

Let us assume, again, that you have passed the first questions, and that you are at the point of choosing an item:

Facilities (pool,elev.,etc)	Rates	Rooms	Quality	Location	Comments

==> Rates

then, the system tries to constraint the user request:

Are you interested in any particular kind of room, like single,
double or suite (yes/no) ?

Inside the system . . .

If the user says "no", the system assumes he is interested in general
information about rates. In this case, it sends the usual message to each guide-manager.
The message consists of:

Select *Madeleine-Place* *rates*

On the other hand, if the user says "yes", the system proceeds to ask what kind
of room he wants. It does that by presenting a new menu:

singles	doubles	suites

=>

This information allows the system to give a much more accurate response to the user. The system could discriminate among the guides and concentrate in those that contain information about the user's particular request. This is especially helpful, given the tremendous incompatibility, that with respect to this field, exists among the various guides.

Some other questions will be asked later.

I will develop here the case of General Information. I choose to do so, because it is the one that presents more interesting examples of semantic conflicts. The other cases can be considered as subsets of this general case. As mentioned before, accomplishing the tasks in the system involves cooperation among the different actors. A detailed description of the functions and interactions among the actors follows:

Inside the Michelin-guide ...

Michelin-guide receives the message. It knows that its database has many info about rates but that it is distributed among various fields. Thus, it asks the guide-Interpreter to retrieve the following information from the external DBMS:

1) Rates

2) Rates.news. This include insightful data, like:

Service-charge (included/not included)

Full-Board (true/false)

If false then:

Breakfast-included (t/f)

If false then:

Price-of-breakfast

Once it receives this information it checks for different values. For example, if Full-Board is not part of the answer, there should be data specifying whether breakfast is included in the price of the room. If this is not included the next data should be the price of breakfast.

If on the other hand, Full-Board is part of the answer, Michelin-manager sends a message to Michelin-criteria to find out the exact meaning of Full-Board. To this message Michelin-criteria answers by explaining that the price given for Full-Board is per person.

All these details must be accounted and clear as the starting point for a sensible comparison of opinions later on.

The rates in this guide are expressed as a range, i.e. 400-680, where the first number represents the lowest price for a single room, and the second, the highest for a double. All prices are in "francs".

NO 1155-853

OBJECT-ORIENTED APPROACH TO INTEGRATING DATABASE
SEMANTICS VOLUME 4(U) MASSACHUSETTS INST OF TECH
CAMBRIDGE A GUPTA ET AL DEC 87 MIT-KBIISE-4

474

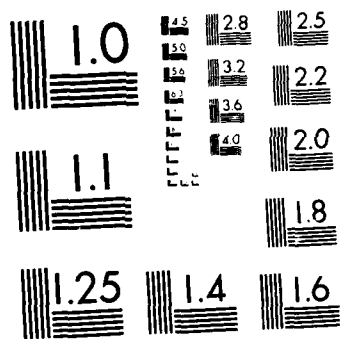
UNCLASSIFIED

DTR557-85-C-00083

F/G 12/7

NL





The real question with "rates" is how to map the local schema with the global schema. The approach I follow - and just apply to rates is to leave the data in the local schema terms. The idea is to allow the System Manager, that has a broader scope of knowledge about the system, to handle the raw data as it is. This is necessary given the complexity of the semantic conflicts associated with rates.

Finally, Michelin-object asserts its data as a range and expressed in French Francs.

Inside the OHRG-guide . . .

This guide is completely inconsistent about the type of information it provides for each hotel. In this sense, it can give information about DBW/42-56 that stands for "doubles with bathrooms" for one hotel and give SB that stands for singles in another. It could even provide 3 kinds of information about prices for a particular hotel, i.e. SB/20 D/32 X/7 -where X stands for "extra-person" in the room- while none for another. The criteria does not specify the meaning of omissions.

On the other hand, this guide is very specific about the meal plans offered by each hotel. All the prices are in American dollars.

Again, the guide is asked to assert the data as it is in the hotel-object.

Inside the Fieldings-guide . . .

This guide does not provide specific information about rates for the different hotels. Rather, it categorizes hotels in groups like, "\$", "\$\$", "\$\$\$", "\$\$\$\$". The guide criteria assigns a number to each of these groups and describes them as the average price one could expect to pay in each particular group. These prices are expressed in dollars.

Inside the System . . .

The big problem is now for the Director to figure out what to do with the individual data.

The first problem we have to solve is to, at least, express the data in the same units. The decision on which system to use might be left to the user. At this point the system asks him by saying:

I can present the information about rates in the monetary units that you prefer, could you please select among the following choices ?:

dollars	francs	pounds	yens

==> francs

The units will be converted to francs.

The second problem is how to consolidate the different data. The various possibilities could be 1) to use all the data in trying to calculate an average to present to the user; 2) in the case where the user is not interested in any particular kind of room, forget about ORGH data and calculate an average with Michelin's and Fieldings' data alone; 3) same as #2 but calculating a range; 4) if the data presented in the various guides differs in a considerable way, we might want to show this to the user. In this way we protect him (and us) from unpleasant surprises later.

As a possible approach we will pick #2, calculation of averages. Following the same directions developed in the first sample session, we will rely on Credibility factors associated to each guide with respect to "rates". Finally, we will calculate a weighted average. This time the assertions of the guides are not just "yes", "no" and "don't know", but numbers representing averages. This, however does not disturb the calculation. Now, instead of calculating a Probability as in the previous example, we calculate the average price per room. The calculation is as follows:

$$\text{Average Price} = \sum_i C_{ix} * A_{iy}$$

where: C_{ix} = Credibility of guide i with respect to field Rates.

A_i = average price per room, by guide i. (0 < i <= # of available guides).

This is only one approach and it would be useful to try others.

An interesting problem that often happens with respect to rates is lack of data. For example, as mentioned in the case of the OHRG guide, sometimes there are prices only for double rooms while other times there are only prices for singles. This problem can be approached in many ways. Following the strategies presented in chapter 4 for addressing the problem of Incompleteness, we propose the following alternative.

First, let us say that we have only the price for singles in Hotel X, and we want to know the price for doubles. The system can always find similar data. For example, if the system obtains the "rating" of the hotel, it could try to calculate the percent difference between singles and doubles for various hotels in the same rank or quality. As the difference should not be significant, a good estimate could be calculated. If we apply this percent difference found to our value for single rooms we could be able to calculate an estimate for doubles.

Other approaches may apply to different cases.

The important point here is that the availability of many sources of information definitely helps finding the appropriate answer to the user's request. Giving the variety of aspects about rates in which each guide focus, a sensible composition of data is desirable.

Finally, the information presented to the user will look somewhat like:

Madeleine-Place hotel

The rates for the Madeleine-Place hotel in Paris can be expected to roughly range from 45 to 140 francs. The first number represents the lowest price to be paid for a single room, while the second one represents the highest price for a <double / suite>.

(This last information, in some cases, could be determined with the help of the OHRG data for particular types of rooms).

With respect to meals, the hotel offers "American plan", that means that rates per night include a continental breakfast .

All rates include tax.

5.9. Conclusions from the Implementation Experience.

The framework used in designing and implementing the Hotel Information System proved to be advantageous in several ways.

The combination of object-oriented model with the decentralized approach for organizing the activities of the system allows the designer to completely isolate the different aspects of the system. This proved to be extremely convenient for debugging and updating the system. I consider it one of the best features of the system. The system is extremely easy to understand as everything can be seen from a high level of abstraction. This would have been very difficult using conventional programming tools as the principal aspects of the system would have been obscured by the implementation.

The message passing technique that is adopted in the framework further imposes clean and clear interaction among actors. It makes it natural to identify the actors and their relations.

By assigning a group of actors as the logical representatives of each physical DBMS, we achieve great performance in the execution of the tasks associated to specific DBMSs. This occurs because we build programs to address very particular problems instead of writing very general programs to cover for all the possible cases of the different DBMSs.

This framework that relies on a combination of Knowledge-Base Systems and DBMS technologies proves to be an excellent environment to host applications aimed towards the full integration of heterogeneous DBMSs. The combination of Knowledge-Base Systems and DBMS technologies allows for the building of powerful systems that combine large amounts of knowledge with inference capabilities to reason through it.

The most important contribution to Decision Support Systems is the demonstration that Subjective Semantic Conflicts can be resolved efficiently. This is achieved by exploiting the Inference Engine power of the Knowledge-Base Systems.

6. CONCLUSION

An enterprise that in the recent past seemed to present overwhelmingly complex problems is transformed now, in a readily accessible project. The powerful combination of AI and DBMS technologies, enables us to build effective and efficient systems to fully integrate heterogeneous sources of information.

The usefulness of these systems is restricted to Decision Support applications where accuracy of the information presented is not required. Rather, the value of the system lies in its ability to provide estimates or suggestions that could help the user in his decision making process, without the user getting involved in the technical details of making the actual consultation to the sources.

Integrative systems are especially relevant for applications where the facts cannot be easily verified, and the best support can be given by consulting different opinions on the matter.

The primary advantage of our approach is that it eliminates the obstacles of accessing heterogeneous DBMSs mentioned in previous chapters. These obstacles otherwise, play the role of deterrents for casual users. With this approach a user can exploit the strategic power of accessing a variety of information sources without having to deal with the technical characteristics of them. In the context of an organization, this means that data from the various components of the company like manufacturing, sales, finance, becomes easily accessible to top management or to the various components themselves.

The time savings achieved by using this kind of integrative systems are tremendous because there is no need now for training into the different DBMSs

specificities. Furthermore, the casual user, that would have not invested the time in learning, is now given the possibility to access a variety of systems. The job of understanding the individual Database Languages as well as the individual data models has been transferred to the machine.

A system like this preserves the autonomy of the local DBMSs as it does not require any type of change to the local dbs. This is an important consideration in open environments of interconnected computers, characterized by a variety of users who perceive the DBMSs according to their own interests and applications. A system like this allows for the pre-existing DBMSs to continue serving their constituencies of users without disruption of activities.

Knowledge-Base technology provides the necessary environment in which to develop intelligent systems capable to access data from heterogeneous DBMSs and still produce sensible and useful information. The use of the Knowledge-Base System approach has enabled us to effectively and efficiently address the resolution of Subjective Semantic Conflicts. The resolution of this problem is probably the most important ingredient in making possible the Fully Integrative goal.

By using external DBMSs the KBS is relieved from having to accumulate all that data within itself. This increases the power of both systems as we put reasoning capabilities through large amounts of knowledge.

From the system designer's perspective, the programming methodology used brings important advantages with respect to more conventional methodologies. The Object-oriented model simplifies the design and organization of the system, enhancing

clarity, modularity and functionality. The Object-oriented model represents a shift from machine-oriented data models to user-oriented semantic models.

A particular advantage of the proposed design is that the utility of the framework is independent from the specific data representational model -relational, hierarchical, networks- used in the local DBMSs.

Another feature of such design is that the re-specification of elements in the system does not cause any disruption for the other elements of the system. The only requirement is that the modified element maintains the same protocols of communication with the other elements of the system. This is important in assuring the evolvement of the system.

The machines of the future will be parallel. The exploitation of the parallelism offered by an architecture depends, to a great extent, in the amount of parallelism that the applications themselves exhibit. The architecture proposed organizes elements and activities that reflect inherent parallelism. Most of the activities of the different elements of the system can happen concurrently.

We conclude that the combination of KBS techniques and DBMS capabilities constitutes a very favorable environment in which to build integrative systems that will in turn make on-line DBMSs available to a wider spectrum of end-users.

As a final word, some research extensions. One of the most important features of using a Knowledge-Base System approach is the ability to exploit its explanatory capabilities. Our implementation, unfortunately, do not make use of these capabilities. It will be very interesting to explore this topic in depth.

Also, this framework just deals with the case of retrieval of data from the DBMSs. A necessary extension of this work would be to study the possibility of also modifying the heterogeneous DBMSs.

7. REFERENCES

- [Agha:87] Agha, Gul A. (1987), **ACTORS: A Model of Concurrent Computation in Distributed Systems**, Cambridge, MA: The MIT Press.
- [Al-Zo:86] Al-Zobaidie, A., and J. B. Grimson, "*Expert Systems and Data Base Systems: How Can They Serve Each Other?*," mimeo, Department of Computer Science, University of Dublin, Trinity College, Dublin, Ireland.
- [Ceri:84] Ceri, S., and G. Pelagatti, **Distributed Databases Principles and Systems**, New York: McGraw-Hill.
- [Daya:84] Dayal, U., et. al. (1984), **Knowledge-Oriented Database Management**, Final Technical Report CCA-84-2, Cambridge, MA: Computer Corporation of America.
- [Glig:83] Gligor, V.D., and E. Fong (1983), "*Distributed Database Management Systems: An Architectural Perspective*," **Journal of Telecommunication Networks** 2, #3, pp. 249-270.
- [Glig:84] Gligor, V.D., and G. L. Luckenbaugh (1984), "*Interconnecting Heterogeneous Database Management Systems*," **IEEE Computing** 17, #1.
- [Glig:86] Gligor, V.D., and R. Propescu-Zeletin (1986), "*Transaction Management in Distributed Heterogeneous DBMSs*" **Information Systems** 11, #4, pp. 287-297.
- [Goos:81] Goos, G., and J. Hartmanis, eds. (1981), **Distributed Systems-Architecture and Implementation, An Advanced Course**, Lecture Notes in Computer Science, vol. 105, New York:Springer-Verlag.
- [Hamm:82] Hammer, M., and D. J. McLeod (1982), "*Database Description with SDM: A Semantic Datamodel*," **ACM Transactions in Database Systems** 6, #3 (September), pp. 351-386.
- [Jark:83] Jarke, M., and J. Koch (1983) "*Range Nesting: A Fast Method to Evaluate Quantified Queries*," **SIGMOD Record** 13, #4 Proceedings of The Annual Meeting, San Jose, California., pp. 196-206.
- [Kell:86] Kellog, C. (1986), "*From Data Management to Knowledge Management*," **IEEE Computer** 19, #1 (January), pp. 75-84.
- [Kimb:81] Kimbleton, S. R., Pearl Wang, and B. W. Larson (1981), "*Applications and Protocols*," Chapter 14 in Goos, G., and J. Hartmanis, eds., **Distributed Systems-Architecture and Implementation, An Advanced Course**.

- [Kech:86] Kerchberg, L. (1986), **Expert Database Systems**, New York: The Benjamin Cummings Publishing Company.
- [Kita:82] Kitawa, T. (1982), "*Model and Architecture of Distributed Data Base Sharing Systems Associated With Knowledge Information Processing Systems*," Chapter in R. P. Van de Riet, and W. Litwin, eds., (1982) **Distributed Data Sharing Systems**.
- [Miss:86] Missikoff, M., and G. Wiederhold (1986), "*Towards a Unified Approach for Expert and Database Systems*," Chapter 11 in Kerchberg, L. (1986), **Expert Database Systems**.
- [Scio:86] Sciore, E., and D. S. Scott (1986), "*Towards and Integrated Database-Prolog System*," Chapter 10 in Kerchberg, L. (1986), **Expert Database Systems**.
- [Smit:86] Smith, J. M. (1986), "*Expert Database Systems: A Database Perspective*," Chapter 1 in Kerchberg, L. (1986), **Expert Database Systems**.
- [Spac:84] Sparck, K. (1984), "*Natural Language Interfaces for Expert Systems: An Introductory Note*," **Research and Development of Expert Systems**, pp. 85-94.
- [Vand:82] Van de Riet, R. P., and W. Litwin, eds. (1982), **Distributed Data Sharing Systems**, New York: North Holland Publishing Company.
- [Vick:87] Vickery, A., and H. Brooks (1987), "*Plexus - The Expert System for Referral*," **Information Processing and Management** 23, #2, pp. 99-117.

Hotel Guides Used:

Michelin (1987), **The Michelin Guide to Paris**, Paris: Michelin.

Fielding's (1987), **Fielding's Guide to Europe**, Fielding's Publishers.

OHRG (1987), **The Official Hotel and Reservation Guide to Europe**.

END

DATED

FILM

8-88
STIC