NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

THESIS

**THE ROLE OF EXPERT SYSTEMS
IN FEDERATED DATABASE SYSTEMS**

by

Levent Ince

March 2000

Thesis Advisor: J. Bret Michael
Second Reader: Thomas Wu

Approved for public release; distribution is unlimited.

20000525 055

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 2000	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE The Role of Expert Systems in Federated Distributed Multi-Database Systems.	5. FUNDING NUMBERS
6. AUTHOR(S) Ince, Levent	

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

A shared information system is a series of computer systems interconnected by some kind of communication network. There are data repositories residing on each computer. These data repositories must somehow be integrated. The purpose for using distributed and multi-database systems is to allow users to view collections of data repositories as if they were a single entity. Multidatabase systems, better known as *heterogeneous multidatabase systems,* are characterized by dissimilar data models, concurrency and optimization strategies and access methods. Unlike homogenous systems, the data models that compose the global database can be based on different types of data models. It is not necessary that all participant databases use the same data model. Federated distributed database systems are a special case of multidatabase systems. They are completely autonomous and do not rely on the global data dictionary to process distributed queries. Processing distributed query requests in federated databases is very difficult since there are multiple independent databases with their own rules for query optimization, deadlock detection, and concurrency. Expert systems can play a role in this type of environment by supplying a knowledge base that contains rules for data object conversion, rules for resolving naming conflicts, and rules for exchanging data.

14. SUBJECT TERMS Multidatabase Systems, Federated Databases, Expert Systems, Semantic Networks.	15. NUMBER OF PAGES 240
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THE ROLE OF EXPERT SYSTEMS IN FEDERATED DISTRIBUTED MULTI-DATABASE SYSTEMS.

Levent Ince
Lieutenant .J.G, Turkish Navy
B.S., Turkish Naval Academy, 1994

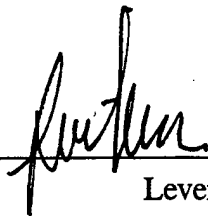Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

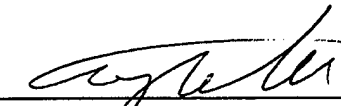## NAVAL POSTGRADUATE SCHOOL
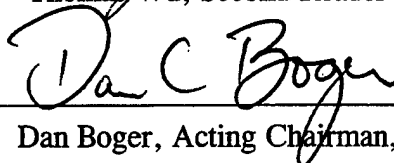### March 2000

Author: _____

Levent Ince

Approved by: _____

J. Bret Michael, Thesis Advisor

_____

Thomas Wu, Second Reader

_____

Dan Boger, Acting Chairman,
Computer Science Department

ABSTRACT

A shared information system is a series of computer systems interconnected by some kind of communication network. There are data repositories residing on each computer. These data repositories must somehow be integrated. The purpose for using distributed and multi-database systems is to allow users to view collections of data repositories as if they were a single entity.

Multidatabase systems, better known as *heterogeneous multidatabase systems,* are characterized by dissimilar data models, concurrency and optimization strategies and access methods. Unlike homogenous systems, the data models that compose the global database can be based on different types of data models. It is not necessary that all participant databases use the same data model.

Federated distributed database systems are a special case of multidatabase systems. They are completely autonomous and do not rely on the global data dictionary to process distributed queries.

Processing distributed query requests in federated databases is very difficult since there are multiple independent databases with their own rules for query optimization, deadlock detection, and concurrency.

Expert systems can play a role in this type of environment by supplying a knowledge base that contains rules for data object conversion, rules for resolving naming conflicts, and rules for exchanging data.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank for my wife, Devrim, for her understanding and support; and my daughter, Ceren (Snow White), for letting her daddy finish his work.

I would like to thank my advisor, Professor James Bret Michael, for his assistance and advice on my thesis.

Special thanks to MSHN group for providing the computers and resources necessary to complete this thesis.

I. INTRODUCTION

In this thesis we investigate the use of expert systems in Federated Multi-Database environments to resolve semantic and structural conflicts. Some models for this kind of database exist and have been adopted by different database communities, but there is no consensus on how to implement the models. We illustrate the principles of multidatabase systems and the role of expert systems to provide a virtually centralized database from these distributed heterogeneous databases. First, we examine the spectrum of approaches representative of the "database engineering." Next we introduce the roles of expert systems in these models.

A. BACKGROUND

A shared information system can be considered as a series of computer systems interconnected by some kind of communication network. There are data repositories residing on each computer which must somehow be integrated. Unlike homogenous distributed database systems, the data models that compose the global database can be based on different types of data models; it is not necessary that all participant databases use the same data model. The goal of using distributed and multi-database systems is to allow users to view collections of data repositories as if they were a single entity.

One can classify these database systems as distributed database systems, multi-database systems, or federated database systems. Distributed database systems are homogenous systems and are characterized by similar access methods, optimization

1

strategies, concurrency strategies, and data models. Multi-database systems, better known as *heterogeneous multi-database systems,* are characterized by dissimilar data models, concurrency and optimization strategies and access methods. Distributed and multi-database systems have at least one feature in common. They both utilize a global data dictionary or schema to assist users to access data objects from the remote site.

Federated distributed database systems are a special class of multi-database systems. They are completely autonomous and do not rely on the global data dictionary to process distributed queries. Processing distributed query requests in federated databases is very difficult since there are multiple independent databases with their own rules for query optimization, deadlock detection, and concurrency. Also the absence of a global schema and global mapping algorithms further complicate the architecting of such systems.

The rapid evolution of the Internet has spurred the growth of federated database systems. Users of the Internet want to reach any kind of information just by typing a couple of letters with the representation of information remaining transparent to them. Many research groups have worked on this issue, often referred to as. "integration of heterogeneous databases," "data interoperability," or "management of multiple information sources". Among all these research projects, only the following have been well prototyped and demonstrated:

- Tsimmis (The Stanford-IBM Manager of Multiple Information Sources)

- Garlic (IBM Almaden Database group)

- The Information Manifold Project (AT&T)

- Strudel (AT&T)

- Disco (Distributed Information Search Components at INRIA, Rocquencourt)

B.   GOAL

Our goal is two-fold. First, we intend to identify the challenges in building integrated information systems and the best methods to address the challenges. Second, we plan to show how expert systems can be used to apply these methods. Expert systems can play a role by supplying a knowledge base that contains rules for data object conversion, rules for resolving naming conflicts, and rules for exchanging data.

All units, headquarters, and supply centers in the Turkish Navy keep their own database designs separate from each other. Although all these databases are domain specific, we believe that our thesis will help to build a unique integrated database all over the Turkish Navy.

C.   THESIS ORGANIZATION

Chapter II describes the current architectures for Federated Distributed Multi-database models. It discusses two main models, the agent-based and mediator-wrapper models. Chapter III contains detailed information about the related work done in this area. Chapter IV presents the semantic representation of data models. In

Chapter V, we discuss the integration of information and different methods of integration. Chapter VI provides conclusions, recommendations and future work. Appendix A presents the fundamentals of expert systems.

II. CURRENT ARCHITECTURES FOR HETEROGENEOUS DISTRIBUTED MULTI-DATABASE MODELS

Information systems that provide interoperation and some degree of integration among multiple databases have been called multidatabase systems [Ref.1], federated databases [Ref.2], and more generally, heterogeneous distributed database systems (HDDBSS) [Ref.3]. The term *federated database system* is used to imply the role of the autonomy ([Ref.4]. There is always a trade-off between sharing and autonomy: the more sharing, the less autonomy. For instance, the use of schema integration increases data sharing dramatically-while reducing database autonomy to almost nothing.

Effective sharing and use of data and functions can be achieved using different forms. Common forms include integration, interoperability, interdependency and exchange. Data integration generally implies uniform and transparent access to data managed by multiple databases. A mechanism to achieve this integration is "an integrated schema involving all or parts of the component schemas that are integrated" [Ref.3]. In HDDBSS, it is not necessary to have a single global integrated schema in the entire system.

We can classify the existing solutions to the data integration problem into three categories: global schema integration, federated databases, and multidatabase language approach. These categories are presented according to the level of integration each component systems demonstrates. We briefly explain general approaches and associated architectures for heterogeneous distributed databases in the following section.

A.   GLOBAL SCHEMA INTEGRATION

Global schema integration is one of the first attempts at data sharing, and is based on the complete integration of multiple databases in order to provide a single view. The advantage of this approach is that users have a consistent and uniform view of and access to data. Users are unaware that the databases they are using are heterogeneous and distributed. Multiple databases logically appear as a single, homogenous database. However, global schema integration has several disadvantages.

This type of database is hard to automate because of the difficulty of identifying the relationships among the qualities of two schemas as well as the relationships among data object types of different databases. The general problem of integrating relational schemas does not have a clear-cut solution. Human understanding is required to solve many types of semantic, structural, or behavioral conflicts.

Autonomy is often sacrificed to solve semantic conflicts. Because the global integration process requires complete semantic knowledge prior to commencing, all databases involved need to reveal information about their conceptual schemas or data dictionaries. Sometimes, this process may even require a local database to alter its schema to make the integration easier.

If there are more than two databases, different integration methods exist. One may either work with all schemas at once, or consider two at a time and the combine them in the final stage of the integration process. Depending on the order in which schema are integrated, only incomplete semantic knowledge is used at each step. As a result, some semantic knowledge may be missing from the final global schema unless integration is

completed in one step and considers all exported schemas at once. It is hard to prove correctness of a global schema.

It is obvious that global schema integration is both time consuming and error prone. This approach is not suitable for given the size of database networks which require frequent dynamic changes of schemas, as it may be necessary to re-do the entire integration process.

B. FEDERATED DATABASE SYSTEMS (FDBSS)

The goal of FDBSs architecture is to remove the need for static global schema integration, and FDBSs are a compromise between no integration and total integration approaches. In contrast to global schema integration, the amount of integration does not have to be complete. This architecture allows each local database to have more control over the sharable information. As FDBSs depend on the needs of the users, the systems may either be tightly or loosely coupled. Typical FDBS architecture has a common data model (CDM) and an internal command language, and relies on the following types of schemas and processors:

- *Local schema*: Is the conceptual schema of a component database, expressed in the data model of component DBMSS.

- *Component schema*: Alleviates data model heterogeneity facilitates negotiation, schema integration (for tightly coupled), and specification of views and queries (for loosely coupled). A local schema is translated to the

7

common data model of the FDBS. Each local database should store one-to-one mappings between the local data model and CDM schema objects during schema translation [Ref.2].

- *Transforming processor (generally called "wrapper")*: Uses the one-to-one mappings between local and CDM schema objects to translate commands from the internal command language to the local query language, and data from local format to CDM format. The transforming processor exists between local and component schemas, and is provided by each data source. [Ref.3]

- *Export schema*: Can classify objects as "sharable" to other members of the FDBS, and contain access control information (i.e., only specific federation users can access certain information.) As a result, association autonomy is maintained.

- *Filtering processor:* Uses the access control information specified in the export schema, and acts as an access controller sitting between the component and export schemas. A filtering processor limits the set of allowable operations submitted to the corresponding component schema.

- *Federated schema*: Can be a statically integrated schema or a dynamic user view of multiple export schemas. The integrated schema is managed and controlled by the FDBS administrator if the FDBS is tightly coupled. The view is managed and controlled by users if the FDBS is loosely coupled. There can be multiple federated schemas, one for each class federation users.

- *Constructing processor:* Uses the distribution information stored in the federated dictionary. The constructing processor performs query decomposition from one federated schema to one or more export schemas, and merges data produced by several processors into a single data set for other single processor, i.e., negotiation and schema integration" [Ref.3].

- *External schema:* Is mainly for customization when the federated schema is very large and complicated. "External schema is another level of abstraction for a particular class of users/applications, which only require a subset federated schema" [Ref.3]. It contains additional integrity constraints and access control information. This schema is not needed for loosely coupled FDB but is essential for those that are tightly coupled [Ref.2]. The data model of external schema can be different from that of a federated schema, necessitating a transforming processor for command and data translation.

- *Data dictionary:* Contains external, federated, and export schema. In a tightly coupled system, component and local schema objects are sometimes contained. External federated schemas, export schemas and other mappings between schemas are all stored in the data dictionary of the FDBS at distinct locations. Other information such as statistics and heuristics for query optimization, schema-independent information such as functions for unit/format transformations, network addresses, communication facility, etc., are stored in the data dictionary.

1. Loosely Coupled FDBSs

It is the user's responsibility to maintain and create the federation schema in loosely coupled FDBSS-- control is not enforced by the federation system or by its administrators. "Creating a federated schema corresponds to creating a view against relevant export schemas" [Ref.2]. Federated schemas are dynamic and can be created or eliminated at any given point. In that respect, each user must be knowledgeable about the information and structure of the relevant export schemas in order to create views.

Loosely coupled systems have many advantages. For example, by using dynamic attributes, federation users have the flexibility to map different or multiple semantic meanings among the same set of objects in export schemas. Loosely coupled systems can also cope with dynamic changes of component or export schemas better than tightly coupled systems, because it is easier to construct new views than to create original global schemas.

Unfortunately, FDBSs also have some disadvantages. "If two or more independent users access similar information from the same component databases, they create their own mappings/views and don't know if the others have done the same mappings/views" [Ref.3]. Thus, there is a potential for duplicate work in view creations and the understanding of the export schemas. Another difficulty is understanding export schemas when the number of schemas is large; because of multiple semantic mappings between objects, view updating cannot be well supported.

2. Tightly Coupled FDBSs

The objectives of tightly coupled FDBR's are to provide location, replication, and distribution transparency. Federation administrators have full control over the creation and maintenance of federated schemas and access to export schemas in tightly coupled FDBSS. A single federated schema helps maintain uniformity in the semantic interpretation of multiple integrated component data [Ref.2]. Multiple federated schemas are harder to maintain, as multiple constraints from multiple export databases are difficult to enforce and could lead to inconsistencies in semantics [Ref.3].

Disadvantages of tightly coupled systems arise as a result of the fact that FDBS administrators and component DBAs negotiate to form export schemas. During negotiation, administrators may be allowed to read the component schemas without any data access, clearly violating autonomy. Additionally, when there are changes in the export component schemas, integrations need to be done from scratch for each federated schema. Therefore, once a federated schema is created, it is very difficult to change (i.e., the schema becomes static).

For the Federated Database system, two different architectures are relevant, the Mediator-Wrapper architecture (discussed in Section D) and the Intelligent Agent approach (discussed in Section E).

C. MULTIDATABASE LANGUAGE APPROACH

The multidatabase language approach is intended for users of a multidatabase systems who do not use a predefined global or partial schema. The aim of a multidatabase

11

language is to create mechanisms that can simultaneously perform queries involving several databases. Queries can specify data from any local participating database.

Multidatabase language systems are more loosely coupled than the classes covered in the previous sections. Such language has features that are not supported in traditional languages. For instance, a global name can be used to identify a collection of databases, (e.g., restaurants, airlines). Databases that cover the same subject are grouped under a collective name using a special query language (e.g. MSQL).

One major criticism of the multidatabase language approach is its lack of distribution and location transparency for users. Because of this hindrance to ease of use, users have to find the right information in a potentially large network of databases. Moreover, users are responsible for understanding schemas and detecting and resolving semantic conflicts. The multidatabase language provides adequate operators and expressive constructs for users to perform the resolution of semantic conflicts at various abstraction levels [Ref.6].

In general, users of this approach are faced with the following tasks: finding the relevant information in multiple databases, understanding each individual database schema, detecting and resolving semantic conflicts, and performing view integration.

D.   MEDIATOR-WRAPPER ARCHITECTURE

To address the common problems in data interoperability, database communities have attempted to enrich common multi-database architecture with *wrappers* and *mediators* (Figure 2.1).
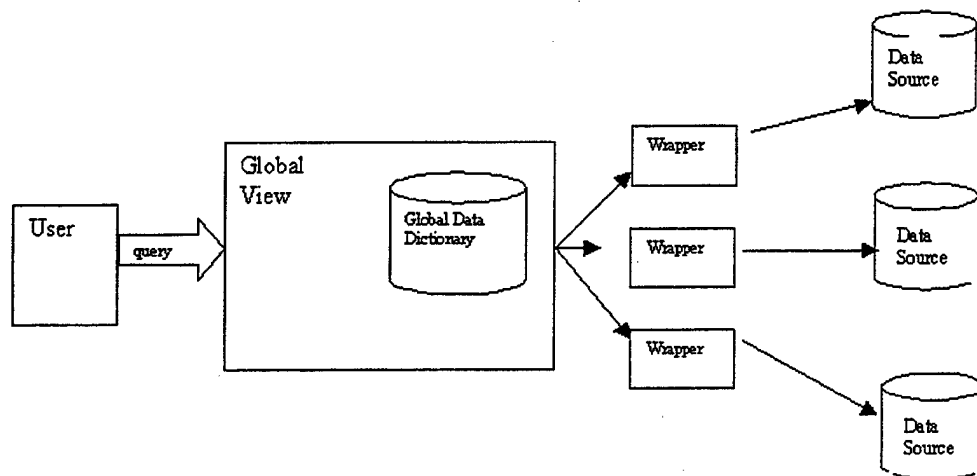
Figure 2.1. Mediator-Wrapper Architecture

The federated data sources are attached to the system by *wrappers* that handle technical and data model heterogeneities. A wrapper logically converts the underlying data objects to a common information model. To accomplish this logical translation, the wrapper converts queries following a common model into a request that the source can execute. The wrapper then converts the data returned by the source into the common model understood by the system.

Above the wrappers lie the mediators. A mediator is a software module that reorganizes information from one or more sources. A mediator embodies the knowledge that is necessary for processing a specific type of information. For example, a mediator for "current events" might know that relevant information sources are the CNN and the ABC databases. When the mediator receives a query, say for articles on, *"earthquake in*

13

Turkey", the mediator will know to forward query to those sources, and may also process answers before forwarding them to the user.

Implementing a mediator can be very complicated and time-consuming, but much of the coding involved can be automated. Each wrapper is assigned a homogenization mediator that transform user queries into terminology understood by the wrapper. If we write all this processes in two basic steps:

Step 1: For each data source, a wrapper exports some information about its source schema, data and query processing capabilities.

Step 2: A mediator centralizes the information provided by the wrappers in a unified view of all available data (stored in the global data dictionary), decomposes the user query in smaller queries (executable by the wrappers), gathers the partial results and forwards the answer to the user.

The Mediator-Wrapper architecture has several advantages. The specialized components of the architecture allow the various concerns of different kinds of users to be handled separately. Since mediators typically specialize in a related set of data sources with similar data, they typically export schemas and semantics related to a particular domain. The specialization of the components leads to a flexible and extensible distributed system.

The main drawback of this architecture is a result of the fact that the creation of mediators is a highly labor-intensive task. Since most potential applications use a large number of information sources and new sources are frequently added to the system, a large number of mediators is needed – some of which have to be generated at the run-time of the system.

There is no consensus on how wrappers describe their sources' capabilities, nor how much of this information is exposed to the mediator. However this Mediator-Wrapper architecture is the most frequently adopted abstraction for the information integration.

E. INTELLIGENT AGENT-BASED ARCHITECTURE

In intelligent agent based architecture, each node in the federation has an export schema and an import schema. The export schema is used to identify the data objects that the node is willing to share with other nodes in the federation. The local import schema is simply the union of the export schemas with all of the other members, and contains data object description of information that the other nodes in the federation are willing to share with this node. The distributed queries generated at each node are formed according to the information present in the local import schema.

The members of the federation agree on general communication protocols and methods for routing queries and data. Processing distributed query requests in federated databases is very difficult since we are dealing with multiple independent databases with their own rules for query optimization, deadlock detection and concurrency [Ref.10].

Intelligent agents can play a role in this type of environment by supplying a knowledge base containing rules for data object conversion, resolving naming conflicts, and for exchanging data. The agents would take as its input rules, the distributed query and the export/import schema information to generate a result transfer plan. This plan would be used to convert the data to any required format acceptable to the requesting site.

15

Agents work together in a cooperative manner to solve problems that they could

not solve their own. Each agent has a knowledge base that has rules used to generate a

result transfer plan. Figure 2.2 shows the simplified diagram of such an agent. [Ref.9]

Distributed Query
Fragments

Export
Schema

Import
Schema

Local
Expert
System

Knowledge
Base

Result
Transfer
Plan
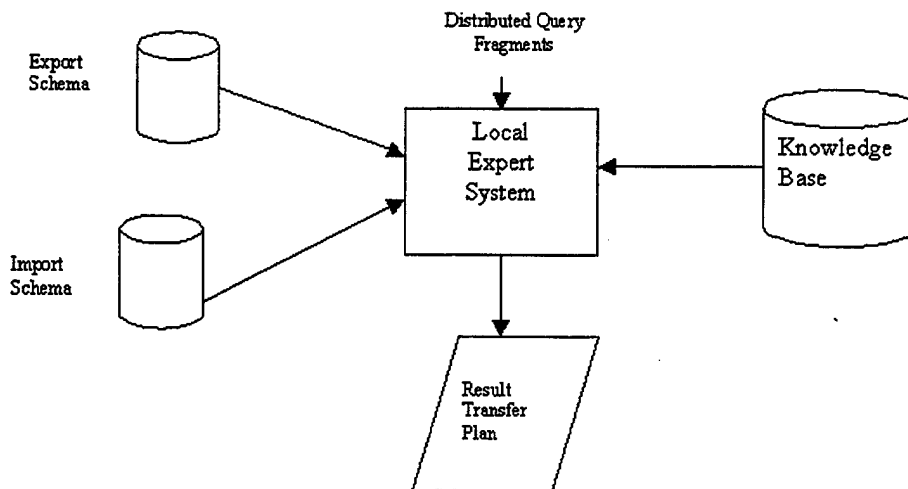
Figure 2.2. Intelligent Agent Architecture.

A query from outside is submitted to the agent. Then the agent consults with its

knowledge base and import/export schema in order to generate a response plan. Any

modification to queries is done at this stage so that they are compatible with the syntax

and requirements of the local DBMS. For example, the query received may be a

spreadsheet macro language command, and the required data may reside in a relational

database. It is the task of the agent to translate these macro commands into a SQL query

in order to retrieve the required data. The result is sent back according to the response

plan generated by the agent.

F. SUMMARY

In this chapter, we have discussed the general architectures for heterogeneous distributed multidatabase models. Information systems that provide interoperation and integration among multiple databases are called multidatabase systems, federated databases, and more generally heterogeneous distributed database systems. The existing solutions to interoperation and data integration are global schema integration, federated databases, and multidatabase language approaches.

In the global schema integration approach, multiple database schemas are integrated into a unique, global schema. As a result, multiple databases logically appear as a single, homogeneous database. This approach is not suitable for frequent changes of component data schemas. Hence, global schema integration approach is not preferred for the integration of large number of databases. Contrarily, the federated database approach does not require a static, wholly global schema integration; instead, integration is realized when needed. In multidatabase language approach, there is no schema integration at all. The answers to queries in multidatabase language from component databases are integrated only (query views).

A more specific approach to the data integration problem is *mediator-wrapper architecture*, which is a kind of federated database system architecture. In this architecture, wrappers logically convert the underlying data objects to a common

information model. Above the wrappers, mediators embody the knowledge that is necessary for processing a specific type of information.

Another type of federated database architecture is *intelligent agent-based architecture*. In this architecture, intelligent agents supply a knowledge base that contains rules for data integration, conflict resolution etc. Each agent has knowledge base that has rules that are used to generate a result data integration plan. In the next chapter, we examine the related work done in this area.

III. RELATED WORK

In this chapter, we show the related work done about Heterogeneous Multidatabase systems. First we introduce the groundbreaking work done by different international organizations and companies. In Section A, we show the basics of these early projects, and place special emphasis on the two main architectures adopted by these projects. In Section B, we introduce the current projects and approaches in this field.

A. EARLY PROJECTS IN MULTIDATABASE SYSTEMS.

In this section, we present a brief history of multidatabase systems, the main issues in multidatabases, and the early work of different organizations and companies from all over the world. Most of these projects were completed in the late 1980s and only deal with the database engineering aspects of Multidatabase systems. We discuss two main architectures and classify the multidatabase systems accordingly. Towards the end of the section, we explain some of the projects that merit more detailed description.

1. Motivation to Multidatabase Systems from Classical Databases

Database systems were proposed as a solution to the problem of shared access to heterogeneous files created and used by different autonomous applications and users. Database systems are used to provide a homogenous access to information by eliminating the difficulties caused by the heterogeneity of information sources. These autonomous files were replaced by a centrally defined collection of data called a *database*. The

19

authority responsible for the centralized control, or the database administrator, worked to integrate the database by defining it and eliminating duplications and heterogeneities. The database could then be managed under a centralized control by a system, or *database system (DBS)*. The DBS gives each application a unique data profile providing consistency and efficiency.

This idea was successful to a large extent. There are many databases in different environments all over the world. However this approach has some drawbacks. There are fundamental problems resulting from the sharing of data between users and applications. Some of these problems are:

- The users must explain their needs to an administrator, and this process may be very difficult and time-consuming and may require need many changes (TO WHAT? WHAT KIND OF CHANGES?).

- There may be a great variety of structures to represent the users' data, and it is very difficult to combine similar structures under the same conceptual schema [Ref.11].

- Users tend to prefer to their own naming conventions, and creating a global naming scheme that satisfies every user a challenging task

- Occasionally, the user data may be badly affected as a result of other users' manipulations.

- The administrator is in charge of optimizing the usage of the database, and this local optimization may not be aligned with a global goal

Briefly, database approach is an attractive idea because it proposes to free users of many annoying aspects of data manipulation. However, its downfall lies in the user's loss of control over his own data, which ultimately means a loss of user autonomy.

These drawbacks become more pronounced in modern information technology where a user has the capabilities to access even very large collections of data (e.g. Internet or LANs of large companies). The data may especially reside in many autonomous databases. These databases present heterogeneities similar to that were in files triggered the whole database idea. We can think that the database idea should be reapplied and we should create a huge (distributed) database involving many small databases. This would mean that a centralized control and at least the presence of a common conceptual schema, namely global schema is required. As the size of the member database number increases, the importance of the drawbacks of classical database approach increases as well. This leaded the database groups to prohibit the reapplication of this approach. This reason has motivated people to propose the multidatabase approach.

The multidatabase design principles were based on the database approach, but intended to maintain its advantages while avoiding or lessening the drawbacks. Thus it was proposed that the data the user needed to manage collectively reside in several databases without a global schema. As a result, the databases were not integrated, and data in different databases began to show duplications and discrepancies in naming and data structures as well as inconsistencies the database design was intended to eliminate.

The user should be able to manipulate data not only in his database, but also to combine data from different databases. These multidatabase manipulations should be

simple in spite of the heterogeneity of the system. To obtain these goals, database groups developed two main design approaches presented in subsections 3 and 4 of Section A. In the next subsection, we will explain the important issues in Multidatabases.

2. Important Issues In Multidatabases

a. Site Autonomy

Multidatabases, as opposed to distributed databases, retain complete control over local data processing for each DBMS. This quality is called "site autonomy". Each site itself determines what information it will share with the global system, what global requests it will service, when it will join the multidatabase, and when it will stop participating in it. Joining the multidatabase does not modify the DBMS itself. Global changes, such as the addition or deletion of other sites, will not affect the integrity of local member databases. [Ref.12].

In spite of its desirable aspects, site autonomy brings a large burden to global DBAs. Each site has its own local requirements and makes independent local optimizations to satisfy those requirements. Because of this independence, a potentially large number of participating sites, global requirements and desirable global optimizations are likely to conflict with local ones. The global DBA must work on these conflicts in initial global system design and ongoing global maintenance. Due to the heterogeneity of local DBMSs, the global system may be required to devote global resources in order to compensate for any missing local information or function.[Ref.12].

22

Some of these problems may be reduced somewhat if the local DBAs agree to cooperate and conform to certain global standards.

b. Differences in data representation

There are many ways to model real-world objects and represent them in a structured database. Since local databases are developed independently with different local requirements, a multi database system may have several different models or representations for the same or similar objects. [Ref.12]. On the other hand, a global user desires an integrated presentation of global information without any duplication or heterogeneity. We can classify these differences as follows:

- *Name differences:* The term "synonym" means the same object has different representation names in different databases, and "homonym" means different objects have the same name in different databases. Each local database may have different conventions for naming objects which leads to the dilemma of synonyms and homonyms. A global system needs to recognize the semantic equivalence of the objects, and then link the local names to a single global name. The system must also recognize the semantic difference between items, and then map the common names to different global names (for semantic equivalence and difference, see Chapter-V).

- *Format differences:* Format differences include differences in data type, domain, scale, and item combinations. For example, a SSN number may be defined as an integer in on database while being defined an

23

alphanumeric string in another. Sometimes data items may be broken into components in one data base while the combination is recorded as a single quantity in another (e.g., an address may be represented as a combination of street, city, state and zip code or as different attributes for each component). Multidatabases typically resolve format differences by defining transformation functions between the local and global representations (see chapter-V). Some functions may be simple numeric calculations, such as converting US dollars to Turkish liras. Others may require conversion tables or algorithmic transformations. For example, a place may be represented by a unique zip code, or by the combination of city and state-- in this case a conversion table is required to convert city/state pairs to zip codes, or vice versa.

- *Structural differences.* An object may be structured differently in local databases depending on how it is used by a database. While a data item has a single value in one database, it may have multiple values in another. The same object may be represented as a data value in one place, an attribute in an other, and as a relation in a third.

- *Missing or conflicting data.* Databases that model the same real-world object may reveal conflicts in the actual data values recorded. Two databases may record the same data item but may assign different values to that item.

c. *Heterogeneous local databases*

Many multidatabases (such as the network, hierarchic and relational models) claim to support heterogeneous data models at the local level. [Ref.11]. Local models should be translated to common global model, typically relational. The support of local DBMS heterogeneity offers a compromise, writing translation code in exchange for improving participation. [Ref.12]. If the multidatabase developers are willing to write enough translation code (which means an extra cost and a decrease in execution efficiency), the multidatabase can accept a wide variety of local DBMSs.

Also, developers should consider that any local functional deficiencies must be programmed by global system software. [Ref.12]. If minimizing translation code cost is an important factor, then the variety of DBMSs allowed to join the multidatabase will be limited, and only those with interfaces close to the global can be integrated into the multidatabase.

d. *Global Constraints*

Because different local databases may represent semantically equivalent data or semantically related data, the global system needs a way of specifying and enforcing integrity constraints on dependencies and relationships within the various databases. [Ref.12]. These constraints are called *global constraints*, and additional semantic information about the data items involved may be represented by these global constraints.

Global integrity constraints may be stored in a separate auxiliary database or defined as part of the global schema. [Ref.12]. The query processor checks the auxiliary databases during queries to enforce the constraints.

e. Global query processing

In multidatabases, global schema is used to submit global query. In the case of a multidatabase language, the query itself contains all necessary information for the retrieval of local data. [Ref.12]. Then, the query is decomposed into a set of sub-queries-- one for each local DBMS that will be involved in query execution. An access strategy created by the query optimizer will specify the relevant local databases, what each of them will do, and how to combine the results. [Ref.12]. Then the access strategy is executed.

In query execution, global constraints must also be checked and enforced. During the execution, queries may be translated several times as they travel through the system layers. Translations allow different languages and representations at different layers and also resolve representation differences.

f. Security

Achieving security in the distributed system is difficult, with problems arising from insecure communication links, varying levels of security at different nodes, and the need to support large number of global user types. [Ref.13]. Site autonomy may provide some measure of local security, because local DBAs can restrict the information

available to global users. The use of views for global users is also an important security measure.

Accessing multiple systems means entering multiple identification and authorization codes. Within a single system, accessing multiple data items can mean acquiring multiple authorizations. A global system must automatically manage these multiple security requirements while preserving the integrity of the security mechanisms. [Ref.13]. Little work has been done on the specific security requirements of the multidatabase environment.

g.   *Local node requirements*

To adequately perform global functions, multidatabases require global data structures and software modules. [Ref.12]. Although site autonomy assures local DBMSs' integrity while joining a multidatabase, the local machine will have to share some of the global storage and processing requirements. Some multidatabases distribute the load evenly over all participating sites and some use server machines to perform most of the global functions. Whit this arrangement, small limited capacity machines can participate in the multidatabase.

Global data structures and global software functions vary among multidatabase systems. The most common data structures include global schema, auxiliary databases for global constraints, space for intermediate query results, and temporary workspaces for global functions. [Ref.12]. Common software functions do translations between local and global languages, transformations between local and global information representations, query processing and optimization, and global system control.

3. Design Architectures

During the first developmental stages of multidatabases, two main designs, *global schemas* and *multidatabase languages*, were adopted by database groups, and current projects still use these approaches. Initially, the global schema approach was used in multidatabase design and continues to be a popular choice for many projects, whereas the multidatabase language approach was developed in partial response to the problems inherent in the global schema approach, and also to address the need for simpler overall system architecture. [Ref.12]. A third approach, federated database systems, was developed to overcome the problems of a global schema.

a. *Global schema*

The global schema approach to multidatabases is a direct descendent of distributed databases. [Ref.12]. The global schema is just another layer, above the local external schemas, that provides additional data independence. Two major differences between distributed databases and multidatabases are: (1) in multidatabases, the global system cannot obligate local systems to conform to any standard schema design (local schemas are developed independently), and (2) the global schema cannot control changes involving local schemas. [Ref.12]. Another major difference is that a multidatabase global schema may integrate local schemas from multiple data models while a distributed database accepts a unique data mode.

The global schema approach has the benefit of making global access a user-friendly process. [Ref.12]. Global users essentially see a single, but still large,

integrated database. The global interface of the database is independent from all the heterogeneity in local DBMSs and data representations. The global schema is usually replicated at each node for efficient user access.

Global schema design: *Global* schema design takes the individually developed local schemas, resolves any semantic and syntactic differences among them, and after that creates an integrated summary of all their information. This is also called as view integration.

As a result of differences in representations and interdependencies between data at different nodes, this process of integration is more complex than the simple union of the input schemas. [Ref.12]. The common techniques for integrating multiple, distinct schemas are covered in chapter-VI. But during the first development era of multidatabases, schema integration process was very human-labor intensive.

Global schema maintenance: A global schema can be very large, so it's very difficult or impossible to replicate it at nodes which have limited storage capabilities. The popularity of personal computers and small DBMSs reside on them may want to join the multidatabase system and this constitutes an important problem. Some systems only replicate the global schema at specified server nodes to compensate this problem. However, "this means queries cannot be processed at query origin nodes which don't have global schema" [Ref.12].

Global DBAs must also maintain the global schema in the face of arbitrary changes to local schemas" [Ref.12]. Changes to local schemas must be propagated to the global schema immediately. "The integration techniques used in global schema design

and the types of changes in local data representations can complicate the mapping of changes to the global schema" [Ref.12].

b. Multidatabase language

The multidatabase language approach was an attempt to resolve some of the problems stemmed from the global schema approach, such as "up-front knowledge required of DBAS, development time to create the global schema, significant maintenance requirements, and processing/storage requirements on local nodes" [Ref.14]. A multidatabase language system puts most of the integration responsibility on users rather than administrators, but also eases their tasks by giving them many support functions and by providing more control over the information. Most multidatabase languages are relational, similar to SQL in standard capabilities, but the functions are extended significantly.

"Many of the language extensions beyond standard database capabilities are involved with manipulating data representations" [Ref.12]. Since there are representation differences in local databases, when a user submits a query, the language must be capable of transforming source information into the representations that the user can understand. "Multidatabase language system users must have a means to display what information is available from various sources" [Ref.12]. The user is supposed to know what information is required and where it probably resides. Otherwise, the magnitude of the information available globally will make finding necessary data an overwhelming task. The language should provide the ability to limit the scope of a query to the pertinent local databases [Ref.12].

In summary, the multidatabase language approach passes the burden of integration to the users and local DBAS from global DBAs. Users must have some global knowledge of representation differences and data sources, but only about the information actually used. "Multidatabase language systems trade a level of data independence (the global schema hides duplication, heterogeneity, and location information) for a more dynamic system and a greater control over system information" [Ref.12].

c. Federated Databases

A federated database is a loosely coupled set of its components [Ref.15]. In federated database systems, there is a federation of loosely coupled databases without a global schema. The main principles for a federation establishment are as follows:

- To establish a federation, each database presents a schema called export schema. This schema is either the "actual conceptual schema (in this case, member database permits all its data to be used) or a derived schema hiding the private data" [Ref.16].

- Data to be manipulated by a user are defined by an import schema. "This schema may group data from several export schemas" [Ref.17].

- There are mechanisms called derivation operators to produce the import schema. "There is also a mechanism for negotiation between databases along a dedicated protocol" [Ref.18].

- "Each federation has a single federal dictionary, which is a distinguished component whose information province is the federation itself" [Ref.17].

An import schema is an external schema. A private schema is either the internal logical schema or the conceptual schema at the multidatabase level. An export schema may be considered equivalent to a conceptual schema at the multidatabase layer. However "it does not seem to be specified in the federated architecture whether the user may manipulate the export schemas directly, separately or jointly" [Ref.17].

"The key words for the federated approach are indeed autonomy and additionally cooperation in interdatabase sharing" [Ref.17]. The global schema multidatabase approach also has these goals, but it requires a global data dictionary. A conceptual schema at the multidatabase layer (Federative layer) may be termed an export schema. Instead of a unique centralized schema, negotiation protocols work between exports and imports schemas of federation.

4. A General Review Of Early Projects In Multidatabase Field

In this subsection, we show the early multidatabase projects reported in the literature. Tables 3.1 through 3.3 review most of the early projects took place in the late eighties. These projects come from a variety of countries, institutes and companies. Some of them are just research projects and others are commercial systems. The variety of work indicates the given importance to this area at this time. The tables were taken from a yearly publication of IEEE Computer Society in 1994 [Ref.19]. Some of the projects take place in tables is introduced in subsequent subsections. The tables compare high-level details of the projects (organizations, global data model and system emphases or key features).

System Name/Organization	Global Data Model	System Emphases and Key Features
ADDS (Amoco Distributed Database System), Amoco Research Center	Extended Relational	Comprehensive function, powerful user interface, global constraints
Dataplex, General Motors Research	Relational	Query decomposition and optimization
DQS (Distributed Query System), CRAI, Italy	Relational	Query optimization
EDDS (Experimental Distributed Database System), University of Ulster	Relational	Small machines can join system
HD-DBMS (Heterogeneous Distributed DBMS), UCLA	Entity-relationship	Global access path information, external views in multiple data models
JDDBS (Japanese Distributed Database System), Japan Information Processing Development Center	Relational	Based on a broadcast network
Mermaid, Unisys	Relational	Query optimization
Multibase, Computer Corporation of America	Functional	Comprehensive function
Multistar, Consortium headed by CRAI, Italy	Relational	Query processing, ability to link to other multidatabases
NDSM (Network Data Management System), CRAI, Italy	Relational	Query optimization
Preci, University of Aberdeen	Relational	Replicated data, nodes can support different levels of global function
Proteus, British universities	Abstracted conceptual	Star network topology, multiple global access languages
Scoop, Universities of Paris and Turin	Entity-relationship	Study mapping algorithms between system levels
Sirius-Delta, INRIA, France	Relational	Translations to/from pivot system (global data model/language
Unibase, Institute for Scientific, Technical, and Economic Information, Poland	Relational	Global constraints
XNDM (Experimental Network Data Manager), National Bureau of Standards	Relational	Data translations, use of server nodes for global processing

Table 3.1. Global Schema Multidatabase Projects [Ref.9]

System Name/Organization	Global Data Model	System Emphases and Key Features
Heimbigner, University of Colorado	Object oriented	Language support for data transformations, negotiation protocol for input schema creation
Ingres/Star, Relational Technology Inc.	Relational	Can define multiple import schemas at a node
Superdatabases, Columbia University	Relational	Hierarchical system structure, concurrency control

Table 3.2. Federated Database Projects [Ref.9]

System Name/Organization	Global Data Model	System Emphases and Key Features
Calida, GTE Research Labs	Relational	Query optimization
Hetero, Felipo Carino, California	Extended relational	Powerful user interface
Linda, Technical Research Center of Finland	Relational	Close to being an interoperable system rather than a multidatabase
MRDSM (Multics Relational Data Store Multiple) INRIA, France	Relational	Comprehensive function, many language features, global constraints
Odu, University of Wales	Entity-relationship	Small machines can join systems
SWIFT (Society of Worldwide Interbank Financial Telecommunication, SWIFT, Europe	Relational	Transaction structure and processing
VIP-MDBS (Vienna Integrated Prolog-Multidatabase System) Vienna Technical Uiversity	Relational	Global language is Prolog

Table 3.3. Multidatabase language system projects

34

The majority of systems under consideration are global schema multidatabases. The reason is that, this approach was a good initial step from classic distributed databases to heterogeneous multidatabases. So most of the organizations have firstly concentrated on this issue and came up with some projects (see Table 3.1). However, problems of size and complexity made global schema multidatabases impractical for large distributed systems. Since the trend turned toward more interconnection (by development of internet and large interconnected systems) at that time, multidatabase language systems – the other major design approach – seemed more practical and attracted the interest of some organizations. Thus some projects have been started and database groups were canalized to this issue. These projects are reviewed in Table 3.2. We reviewed the early projects on Federated Databases in Table 3.3.

Among the early projects, MRDSM deserves a special place. Because the functions defined in this project constituted the basement for other projects.

a.   MRSDM (Multics Relational Data Store Multiple)

MRDSM is multidatabase language system project sponsored by INRIA in France. "MRDSM allows integration of various relational database systems" [Ref.20]. The integration is performed largely at two levels. First, database definition of integrated relations is described in the MRDSM data definition language, and there is also a *multidatabase data manipulation language* available to users of the system. "The database definitions of the integrated view of cooperating databases are created dynamically by multidatabase administrator and either exist for the duration of the user's session with MRDSM or they are stored in the directory for further use" [Ref.21].

The data manipulation language allows users to combine data in different databases, to transform the actual attribute values into user-defined value types dynamically, to retrieve data from different databases in the same query, and aggregate data from different databases using various built-in functions. [Ref.20]

"The data definition language includes capabilities to define the data that a cooperating database willing to share and define user access rights" [Ref.22]. For a collection of databases being integrated by MRDSM, the database administrator assigns a unique name called a multidatabase name.

On a set of relations integrated by MRDSM, the database administrator can define three types of dependencies: manipulation, privacy, and equivalence. "A manipulation dependency triggers processing in one database when processing is conducted in another database" [Ref.20]. As an example, an insertion of a tuple in some relation may trigger insertion of the same tuple into some other relation. "A privacy dependency triggers a check of the user's rights for the database to be accessed" [Ref.20]. "An equivalence dependency identifies for each database to be accessed its primary or candidate keys" [Ref.20]. Equality of primary keys from two separate relations indicates that the same real objects are stored in these relations.

"The query and data manipulation languages of MRDSM are based on a tuple calculus and patterned after QUEL" [Ref.21]. It includes retrieve. modify, store, delete, copy, move, and replace statements. To manipulate one or more databases, a user has to submit an open command that includes each database to be manipulated along with the access mode (shared or exclusive). To realize access of several databases in one user

query, several new concepts are proposed in MRDSM. Among them are: multiple identifiers. and semantic variables.

"A multiple identifier refers to the same relation name at two different sites" [Ref.20]. It is assumed that if two relations at two different local databases have the same relation name, then they contain at least semantically related information. For example, if a relation at one site named 'ship' and a relation at the other site also is called 'ship', then both relations probably contain (but not necessarily the same) information about the ships. "A semantic variable allows user to identify several relations that contain semantically relevant information with one name in the user query" [Ref.21].

Users of MRDSM are capable of defining dynamic attributes for either duration of the query or the duration of the user's session with MRDSM. A dynamic attribute is a function definition on actual attributes that the user is authorized to access. Finally, by using copy and move commands, relations can be copied and moved form one site to another. Users may create a relation as a result of their query and move it to an indicated data location. [Ref.20]

The MRDSM includes a rich function set. While some of these functions are intended as general notions, others are specific to the relational data. These functions are basically

- the definition and alteration of multidatabases,

- "Cooperative data definition: single statement creation (alteration, drop,..) of a relation in several databases, import of data definition, etc." [Ref.17].

37

- Classical retrievals and updates of relations, however "being in different databases, called elementary multidatabase queries in the MRDSM terminology" [Ref.17],

- "So-called multiple queries, performing relational operations on sets of possibly heterogeneous tables" [Ref.17].

- Possibility of multiple identification of data objects bearing the same name, to deal with data duplication and fragmentation [Ref.23].

- Possibility of dynamic unification of heterogeneous names of data objects to deal with name heterogeneity [Ref.24].

- Implicit joins for queries to databases with similar data, but different decomposition into relations [Ref.25].

- Dynamic attributes, for "ad-hoc transforms of heterogeneous data values to a user defined basis" [Ref.17].

- Various new built-in functions. For instance, "for transformation of data names into data values subject to relational operations (names in one database may correspond to a data value in another)" [Ref.17].

- View definition, using the (multidatabase) query modification technique

- Multidatabase external schema definition

- Interdatabase queries for data flow between databases.

- "Auxiliary objects like manipulation dependencies, equivalence dependencies and procedures (transactions, stored queries ....)" [Ref.25].

In summary, MRDSM is a heterogeneous database system that is able to integrate pre-existing relational databases. This system allows users to join data from different relations in different databases, to define dynamic attributes, and to dynamically aggregate data from different relations. From this point of view, it was the most complete prototype built so far at that time. On the other hand, "MRDSM does not deal with physical data distribution, nor with the issues of query optimization in the heterogeneous environment" [Ref.20].

b.   *Sybase*

This system was designed by Sybase Inc. in Berkeley, California. It was a high performance relational system, and it is still available on SUN workstations. The implementation of Sybase on the SUN may be entirely on one machine or the front-end software (client) on one machine and the server software on another. Several front-ends may share a server and a front-end may access several servers [Ref.20]. This first version of Sybase was not a distributed system, however the distributed version was released in 1988.

Sybase language is an extension of SQL, called Transac-SQL. Also, one may use a more user friendly interface called Visual Query Language (VQL). Transac-SQL and VQL were the first multidatabase languages on the market. They have several interesting features:

- "The user may qualify the relation name, let it be T, with the database name, let it be B using the form B.T. Thus one may formulate elementary multidatabase queries" [Ref.20].

39

- The user may define multidatabase views, but not virtual databases.

- The queries may include implicit joins. Unlike in MRDSM, they are however "limited to relations with a single connection through primary or foreign keys" [Ref.20].

- "The user may formulate interdatabase queries using multidatabase INSERT and UPDATE statements. The latter statement then takes values in a table and puts them accordingly into a target table" [Ref.20]. These statements map column names only by order of their enumeration in the SELECT clause, while MRDSM also allows columns to be mapped by name.

- The user may define interdatabase manipulation dependencies. Thus "a manipulation of one database, may trigger that of another" [Ref.20]. The length of the chain is however limited to 8 elements, to avoid infinite cycles.

- In the distributed version of Sybase, the language allows multiple queries to be formulated.

The differences in the user interface of Trans-SQL and VQL with respect to these functions, compared to MRDSM and MSQL are as follows:

- The user opens explicitly only one database at a time, through USE <database name> statement. This database constitutes the default scope for table and column names. All other databases remain however, available to the user, provided he has the access rights. The access to a

database is triggered by the use of its name as the prefix. In contrast, "MRDSM allows the user to open explicitly several databases and does not allow other databases to be used" [Ref.21]. The database name is then required as the prefix only if table names conflict.

- "The multiple queries will most likely be generated through the new statement FOR EACH <table names> <elementary query>" [Ref.20]. This is somewhat more procedural than the use of multiple identifiers or semantic variables in MRDSM. It also makes the "query formulation less open to the local autonomy" [Ref.20].

Sybase is an important system that was widely used and selected by Microsoft to become the Microsoft system for IBM-PS2, replying to OS2/DB of IBM. It was also selected by Apple for Mac SE and Mac-2 and by Ashton-Tate to replace the famous Dbase [Ref.20].

c. *Empress V2*

This system was made by Rhodius Inc, in Toronto, Canada and it was one of the first well developed multidatabase language systems. The version described below is V2, following the "classical" version 1. Unlike Sybase, Empress V2 is a "distributed system that runs on a number of computers over the Ethernet network: Sun, Vax, Apollo, IBM-PC/PS and etc." [Ref.20].

It uses a multidatabase extension of SQL that was as follows:

- Table names in a query may be prefixed with database names. The database names may themselves be further prefixed by multidatabase names that are ultimately the site names.

- Several databases may be open simultaneously.

- The user may define multidatabase views and virtual databases. Both views and virtual databases may be distributed. "A virtual database is manipulated as a single actual one, with location transparency, except for some updates" [Ref.20].

- "Empress V2 supports distributed updates using two phase locking and two phase commitment" [Ref.20].

Empress V2 has multidatabase features that Sybase has not and vice versa. Using multidatabase names in front of database names, in particular allows the resolution of name conflict between database names and also using database names with table names helps to resolve the naming conflicts between entity and relation names.

An auxiliary , but worth to be mentioned feature of Empress V2 is that, for the first time it supported multimedia data in multidatabase systems. "These data may be declared as a particular "bulk" column of a table" [Ref.20]. They may then be interpreted as text, image or voice data. This feature was an opening towards future multidatabase and multimedia systems and towards the interoperability with information systems other than DBSs.

d. *Distributed Ingres*

Distributed Ingres, also called Ingres/Star, was a software layer to Ingres systems and, in the future, to other types of DBSs, supporting an SQL interface. The component providing this interface is called "gateway". Ingres is one of first example of a federated database system.

There are not any interdatabase dependencies in Ingres architecture. Thus, the consistency of replicated data cannot be guaranteed, unlike in Sybase.

The main features of Ingres/Star are as follows:

- The system allows the definition of any number of the external multidatabase schemas over subcollections of SQL databases. "The virtual database defined by this schema is called a distributed database (DDB) and its elements are called links" [Ref.20]. Once the DDB is created, it is used as an actual Ingres database, except for update limitations. The DDBs may in particular share an actual table or database.

- In fact, if a DDB creation is requested over n databases, then it is created over n+1 databases [Ref.20]. The latter database is a hidden actual database created at the node of the DDB schema definition. "This database is named upon the DDB and allows a DDB user to transparently invoke the CREATE TABLE statement" [Ref.20]. These tables may be updated, altered etc.

- "The system does not allow the user to directly formulate multidatabase queries to actual databases or, more precisely, to their export schemas" [Ref.20]. The reason for this is mainly implementation dependent, namely

43

the necessity of a dictionary entry, created when a link is declared. "The only way to formulate an ad-hoc query is to define a DDB whose links are the addressed tables and formulate the query to the links" [Ref.20]. The links may be declared temporary in which case the DDB is automatically dropped. Otherwise, the user must drop the DDB himself or keep it for further needs.

In both cases, the additional manipulations required clearly make Ingres/Star less flexible for ad-hoc multidatabase queries than Sybase and Empress. In addition there is a danger of system pollution with DDBs and the underlying hidden actual databases, created for a particular query and then forgotten. [Ref.20]

e.   Oracle V5

In this version, Oracle has become multidatabase system and adopted a multidatabase language system architecture. The creation of several databases at the same site and the formulation of elementary multidatabase queries were allowed in this version. The Oracle multidatabase language was termed SQL*PLUS. Unlike in Sybase or Empress, the database name does not prefix the table name, but postfixes it, after the character '@'. This capability allows the resolution of the name conflict, avoiding the use of the database name. "The user has also particular statements defining aliases for table names and for database names" [Ref.20]. This process is called database links (but it is different from the data link defined in Ingres/Star).

'The language offers also statements for interdatabase queries unknown to other commercial systems" [Ref.20] and largely similar to the corresponding ones in

MRDSM. All the multidatabase manipulations are moreover available for distributed databases, through the distributed database management component SQL*STAR.

f. Mermaid

This system was developed in the System Development Group of UNISYS. While it was initially intended as a classical distributed DBS, it evolved towards the federated architecture. Its overall features are like those of Ingres/Star and so we will not discuss it detailed here. However, there are numerous differences at the implementation level. In particular, "Mermaid used an original pivot language designed for easy translation towards heterogeneous relational languages" [Ref.26].

g. Calida

This system was developed in GTE Research Laboratories. The operational version was designed for the management of numerous databases of GTE, mostly the relational ones. The main features of the system are as follows:

- "Calida makes it possible to access relational and Codasyl-like databases" [Ref.20]. The internal logical schema and the corresponding manipulation are generated through the original rule processing system. This system provides a particularly flexible interface to data model heterogeneous databases.

- The multidatabase manipulation language is DELPHI not SQL. DELPHI language is used as a final language for the sophisticated user and as an intermediate language for a natural language for interface. DELPHI

45

allows the formulation of elementary multidatabase queries, including the updates, where database names may be used as prefixes to solve name conflict. The query decomposition is carefully optimized, using field statistics gathered by the system [Ref.20]. Calida moreover allows the definition of external schemas through the usual query modification technique.

"The system supports the implicit joins that, in particular, may concern columns in tables in different databases" [Ref.20]. This feature existed only in MRDSM, as it requires the definition of equivalencies between domains or tables of different databases. In the GTE system, the corresponding equivalence dependencies are stored in a so-called global dictionary. The algorithm for the query completion is that "it searches for a minimal spanning tree over the intersection of the non-connected query graph and the connected database graph whose nodes are relations and edges are connections through keys". However, the algorithm is limited to the case of a single connection between two relations (acyclic graphs). If there are multiple connections, the user is asked to make a choice. "One advantage is a fast recursive algorithm for the spanning tree edges computation" [Ref.20].

h.   DQS/Multistar

The Distributed Query System (DQS) prototype was a multidatabase system developed by CRAI, Italy and a sample of global schema multidatabase system. The DQS prototype leaded to a commercial version called Multistar. It "allows

multidatabase retrievals from IMS/VS, IDMS, ADABAS and RODAN databases, as well as from standard VSAM files" [Ref.27]. The objects of these databases are presented as relations through dedicated mapping commands. The retrievals are formulated in SQL over so-called global schema in DQS terminology. However the DQS global schema is in fact an import schema, as several different schemas may be defined which may be partial, and they may overlap. [Ref.27]

These schemas may also include views. Views are the principal data abstraction mechanism for aggregations and generalizations in DQS.

DQS had several interesting features, especially its algorithm for SQL query decomposition. Views are dealt with using the query modification technique.

The query is represented as a tree subject to the algebraic transformations to reduce intermediate relations. A heuristic algorithm is also used to produce the query tree optimized with respect to data movements between the sites. For execution, this tree is finally transformed to a Petri Condition-Event net. [Ref.27]

B.   CURRENT PROJECTS IN MULTIDATABASE SYSTEMS

There are currently many research projects on data integration. In this subsection, we only briefly mention a sample of them that have been well prototyped and demonstrated: Tsimmis at Stanford University, Garlic at Almaden Research Laboratories, DISCO at Inria (France), and finally Infomaster at Stanford University. Other projects that we can consider important but we didn't mention in our thesis are Information

Manifold at AT&T Research Laboratories, Ariadne at University of South California, Strudel at AT&T Research Laboratories, and Whirl at AT&T Research Laboratories.

1.   TSIMMIS

Tsimmis is a joint project of Stanford and IBM Almaden Research Center. As an acronym, TSIMMIS stands for "The Stanford-IBM Manager of Multiple Information Source". The goal of the Tsimmis Project is to develop tools that facilitate the rapid integration of heterogeneous information sources that may include both structured and unstructured data. In this subsection we show the system components that extract properties from unstructured objects, that translate information into a common object model, that combine information from several sources, that allow browsing of information, and that manage constraints across heterogeneous sites.

The goal of the Tsimmis project is to provide tools for accessing, in an integrated fashion, multiple information sources , and to ensure that the information obtained is consistent.

a.   *System Architecture*

Figure 3.1 shows the general architecture of the Tsimmis system. We describe the main components in the following subsections.

1.   *Translators and Common Model:* Figure 3.1 shows a collection of heterogeneous information sources. These sources may be any combination of databases, object stores, knowledge bases, file systems, digital libraries etc. Above

each sources a *translator* (or *wrapper*) resides which logically converts the data object in the source to a *common information model*. To make this conversion, the translator first converts the coming query in the form of common information model into a request query form that the source can execute. After the execution of query in the source, the result is again converted into common model.

Figure 3.1. TSIMMIS Architecture.

For the Tsimmis project, a simple self-describing object model called *Object Exchange Model (OEM)* is adopted. "OEM allows simple nesting of objects" [Ref.28]. We explain the OEM in subsection *b*. But the idea is that: each object

49

and their subobjects are described by *labels* associated with them. For example, the following objects represent a country:

< name-of-country, str, Turkey>

where the string "name-of-country" is a human-readable label, "str" is string type value, and Turkey is the value of the object.

To request OEM objects, Tsimmis group has also developed a query language called *OEM-QL*. QEM-QL is described in subsection *b*, but basically it is an "SQL-like language extended to deal with labels and objects nesting" [Ref.28].

2. *Mediators:* Above the translators in Figure 3.1 lie the *mediators*. A mediator is a "system that refines in some way information from one or more sources" [Ref.29]. "A mediator embeds the knowledge that is necessary for processing a specific type of information" [Ref.28]. For example, a mediator for "current events" might know that relevant information sources are the CNN and the ABC databases. When the mediator receives a query, say for articles on "earthquake in Turkey", it will know to forward query to those sources. "The mediator may also process answers before forwarding them to the user" [Ref.28]. For example, it can convert the date information into a common format or eliminate articles that duplicate information.

While data conversion is simple, eliminating duplicate information requires real intelligent. In Tsimmis, "relatively simple mediators based on patterns or rules are forced" [Ref.20]. The goal of the Tsimmis project is "to automatically or semi-automatically generate mediators from high level descriptions of the information processing they need to do" [Ref.28]. This is feature in the architecture is shown in

50

mediator generator box in the Figure 3.1. Similarly, a *translator generator* is provided to "generate OEM translators based on description of the conversions that need to take place for queries received and results returned" [Ref.28].

3.   *System and User Interfaces:* Mediators export an interface of information sources that is identical to that of translator to their users. Both translators and mediators take as input OEM-QL queries and return OEM objects. Hence, "end users and mediators can obtain their information either from translators and/or other mediators" [Ref.20]. This approach allows "new sources to become useful as soon as a translator is supplied, it allows mediators to access new sources transparently, and it allows mediators to be "stacked," performing more and more processing and refinement of the relevant information" [Ref.28].

End users (on top of Figure 3.1) can access information either by writing applications that request OEM objects, or by using one of the generic browsing tools that have been developed in the project. The browsing tool provides access through *Mosaic* or other *WWW* viewers. The user writes a query on an interactive worldwide-web page, or selects a query from a menu. The answer is received as a hypertext document. This tool provides easy interaction with heterogeneous information sources.

4.   *Labels and Mediator Processing:* It is important to note that "there is no global database schema, and that mediators can work independently" [Ref.20]. To build a mediator it is only necessary to understand the sources that the mediator will use. In fact, "it is not even necessary to fully understand the sources used" [Ref.20]. For example, returning to our "current events" mediator, suppose one source

51

exports objects with sub-objects labeled by title, date, author, and country. The mediator might always pass the author and country sub-objects to its client with no additional processing. Now let's suppose that, a second source provides topic and date sub-objects. The mediator will convert the date information from both sources into a common format, and it will also convert a mediator query about the subject of an article into the proper topic or title queries that will be sent to the sources.

When a mediator simply passes sub-objects to its clients (in our example these sub-objects are author and country), "it might append the source name to the labels so that the client can interpret the objects correctly" [Ref.20]. For example, a mediator sub-object might have label ABCNews.author, which means this author is from ABC source and follows its conventions for authors. Another object might have the label CNN.author.

The idea is that "a mediator does not need to understand all of the data it handles, and no person or software component needs to have a global view of all the information handled by the system" [Ref.28].

5. *Constraint Management:* Another important component in the Tsimmis architecture is constraint management, illustrated in Figure 3.1 by a *Constraint Manager* and two *Local Constraint Managers.* "Integrity constraints specify semantic consistency requirements over stored information; such constraints arise even when the information resides in loosely coupled, heterogeneous systems" [Ref.28]. Constraint management in the distributed, heterogeneous environments addressed by Tsimmis is more difficult and complex than constraint management in centralized systems: "Transactions across multiple information sources usually are not provided, and

each information source may support different capabilities for accessing and monitoring the data involved in a constraint" [Ref.20].

Since in a loosely coupled environment it is generally not possible to guarantee that every user or application sees consistent data every time it interacts with the system, the Tsimmis constraint manager enforces constraints with weaker guarantees than what a centralized system may provide. Tsimmis makes "relaxed" guarantees, e.g., a constraint is true from 8am to 5pm every day, or a constraint is true if some 'Flag' is set. Ensuring relaxed consistency is especially challenging because one now has to deal with the timing of actions and of guarantees. However, the advantages of being able to handle relaxed guarantees in heterogeneous systems are significant; knowing precisely what holds and what does not hold, and when, will clearly lead to more trustworthy systems. [Ref.28]

6.   *Classification and Extraction:* The final component of the Tsimmis architecture is the *Classifier/Extractors* shown at the bottom of Figure 3.1. Many of the important information sources are completely unstructured, consisting of plain files or incoming bit strings. Generally it is possible to automatically classify the objects in such sources (e.g., is the plain file a spreadsheet, a text file, or an image file?), and to extract key properties (e.g., creation date, author). The Classifier/Extractor performs this task. "The information collected by the Classifier/Extractor can then be exported (via a translator, if necessary) to the rest of the Tsimmis system, together with the raw data" [Ref.20].

b. *Object Exchange*

As described before, the Object Exchange Model (OEM) is used as the unifying object model for information that will be processed by Tsimmis components. OEM is used for the processing of logical queries, and for providing results to the user. Each object in OEM has the following structure:

Label	Type	Value	Object-ID

Label: A character string indicating what the object represents. For each label "a translator or mediator exports, it should provide a "help" page that describes (to a human) the meaning and use of the label" [Ref.28]. These help pages can be very useful during exploration of information sources, and for deciding how to integrate information.

Type: The data type of the object's value. Each type is either a basic data type such as integer, string, real number, etc., or the type *set* or *list*.

Value: A variable-length value for the object.

Object-ID: A unique variable-length identifier for the object.

Suppose an object representing a ship has label *ship* and a set value. The set consists of three sub-objects, a *name*, a *country*, and a *photo*. All four objects are exported by an information source via a translator, and they are examined by a client. The only way the client can retrieve the ship object is by posing a query that returns the object as an answer. If the data type of an object is *"set"*, then the values will be references to sub-objects in this set. So in our example, the value field of our OEM object will be

54

{o1,o2,o3} where oi is the reference to corresponding sub-object. Thus, on the client side, the retrieved object will look like:

<ship, set, { o1,o2,o3}, id0 >

o1: location of <name, str, TCG Doganay, id1>

o2: location of <country, str, Turkey, id2>

o3: location of <photo, bitmap, "some bits", id3>

"The primary reason for choosing a very simple model in Tsimmis project is to facilitate integration" [Ref.28]. "Simple data models have an advantage over complex models when used for integration, since the operations to transform and merge data will be correspondingly simpler" [Ref.20].

c. *Summary*

The basic aspects of Tsimmis project are as follows:

- Tsimmis focuses on providing integrated access to very diverse and dynamic information. The information may be unstructured or semi-structured, often having no regular schema to describe it. "The components of objects may vary in unpredictable ways (e.g., some pictures may be color, others black and white, others missing, some with captions and some without)" [Ref.28]. Moreover, the available sources, their contents, and the meaning of their contents may change frequently.

- Tsimmis assumes that "information access and integration are intertwined" [Ref.20]. "In a traditional integration scenario, there are two phases: an integration phase where data models and schemas (or parts thereof) are

55

merged, and an access phase where data is fetched" [Ref.28]. In Tsimmis environment, it may not be clear how information is merged until samples are viewed, and "the integration strategy may change if certain unexpected data is encountered" [Ref.20].

- Integration in Tsimmis environment requires more human participation. In the extreme case, integration is performed manually by the end user. For example, a stock broker may read a report saying that IBM has named a new CEO, then retrieve recent IBM stock prices from a database to deduce that stock prices will rise. In other cases, integration may be automated by a mediator, but only after a human studies samples of the data, determines the procedure to follow, and develops an appropriate specification for the mediator generator. [Ref.28]

In summary, the Tsimmis goal is not to perform fully automated information integration that hides all diversity from the user, but rather to "provide a framework and tools to assist humans (end users and/or humans Programming integration software) in their information processing and integration activities" [Ref.28].

2. Garlic

Garlic is a project of IBM Almaden Research Center. It is not an acronym. The team decided to give this name to their project, "because most members really like garlic and enjoy their laboratory's proximity to the Gilroy garlic fields!" [Ref.31]

"The goal of this project is to develop a system and associated tools for the management of large quantities of heterogeneous multimedia information" [Ref.31]. Garlic allows traditional and multimedia data to be stored in a variety of existing data repositories, including databases, files, text managers, image managers, video servers etc. "The data is seen through a unified schema expressed in an object-oriented data model and can be queried and manipulated using an object-oriented dialect of SQL" [Ref.32]. The Garlic architecture is extensible to new kinds of data repositories, and access efficiency is addressed by "a *middleware* query processor that uses database query optimization techniques to exploit the native associative search capabilities of the underlying data repositories" [Ref.31].

In the subsection *a*, we show the general Garlic architecture and its components. We describe the Garlic data model and query language in subsection *b*, finally we summarize the Garlic project in subsection *c*.

a. *System Architecture*

Figure 3.2 ([Ref.12]) shows the overall architecture of the Garlic system. At the leaves of the figure are a number of data repositories (relational and non-relational database systems, file systems, document managers, image managers, video servers etc). A repository wrapper resides on each data repositories which translates information about data schemas and queries between Garlic's internal protocols and that repository's native protocols. "Information about the unified Garlic schema, as well as certain translation-related information needed by the various data repositories, is maintained in the *Garlic metadata repository*" [Ref.32]. The other repository shown in the figure is the Garlic

complex object repository.   This repository is used to hold the complex objects that most Garlic applications will need for "gluing" together the underlying data in new and useful ways. Complex objects will be needed to integrate multimedia data with legacy data in situations where the legacy data cannot be changed, and as a place to attach methods to implement new behavior. For example, in an auto insurance application, Garlic complex objects could be used to link the images of a damaged car (stored in an image-specific repository) together with an accident report (stored in a document management system) and a customer's claim and policy records (legacy data residing in a relational database) in order to form a "claim folder" object to be dealt with by an insurance agent. [Ref.31]

Figure 3.2. Garlic Architecture.

58

The Garlic query services and runtime system component provide query processing and data manipulation services. This component presents Garlic applications with a unified, object-oriented view of the contents of a Garlic database and processes users' and applications' queries, updates and method invocation requests against this data; queries are expressed in an object-oriented extension of the SQL query language. [Ref.31] This component is also responsible for dealing with transaction management issues.

Finally, Garlic applications interact with the query services and runtime system through Garlic's object query language and a C++ application-programming interface (API). "Many applications do this statically, in which case the Garlic schema is presented to the application via a set of C++ classes that act as "surrogates" for the corresponding classes of the actual Garlic schema" [Ref.32]. Certain applications may require more dynamic access to the data, in which case they will use "a portion of the C++ API that provides dynamic access to information about the types and objects contained in a Garlic query/browser" [Ref.32]. This component of Garlic provide end users of the system with a friendly, graphical interface that supports interactive browsing, navigation, and querying of the contents of Garlic databases. In the following subsections we will describe the main functions of the system with more details.

Data Transformation at the Wrapper: In order to reach the data resides in different data repositories, a wrapper must be implemented specific to these data repositories. Hence, it is possible to extend the system by adding new kind of data repositories and writing new wrapper codes for these repositories. The most basic tasks

59

of a wrapper are: "To describe the data in its repository" [Ref.31] and, "provide the mechanisms by which users and the Garlic middleware engine may retrieve that data" [Ref.33].

Since a data source is not likely to conform to Garlic's data model and data format (see subsection *b*), the wrapper must perform some level of schema and data transformation. To make the task of writing a wrapper as easy as possible, the "Garlic wrapper architecture tries to minimize the required transformations, but wrappers can do more if desired" [Ref.31].

"The schemas of individual repositories are merged into the global schema via a wrapper registration step" [Ref.32]. In this step, wrappers model their data as Garlic objects, and provide an *interface* definition that describes the behavior of these objects. The interface is described using the Garlic Definition Language (GDL) (see subsection *b*). "The interface definition provides an opportunity for a wrapper to rename objects and attributes, change types and define relationships even if the data source stores none" [Ref.31]. For example, "a relational wrapper might model foreign keys as relationships" [Ref.32]. Developing interface files is typically not hard. For simple data sources, it may be best to generate them manually, as simple sources tend to have few object types, usually with very simple attributes and behavior. For more complex sources, the process of generating an interface file can be automated. For example, "a relational wrapper can decide on a common mapping between the relational model and the Garlic data model (e.g. tuple=object, column=attribute), and provide a tool that automatically generates the interface file by probing the relational database schema" [Ref.31]. Wrappers must also provide an *implementation* of the interface which

60

represents a concrete realization of the interface. The implementation cooperates with Garlic to assign a Garlic object id (OID) to its objects, and maps the GDL base types specified in the interface file to the native types of the underlying data source. [Ref.32]

A simple example taken from [Ref.31] illustrates the kinds of simple schema and data transformations that wrappers can perform. This is a hypothetical travel agency application. The agency would like to integrate an Oracle database of information on the countries and cities for which it arranges tours with a web site that contains up-to-date booking information for hotels throughout the world. The example in the next page shows the original table definitions and the -new interface definitions for the two relational tables, and the interface file for the web site. The relational wrapper renamed the HIGHESTPEAK field to highest_peak, and exposed the foreign key COUNTRY on the CITIES table as an explicit reference to a Country object in the integrated database. The wrapper must be able to map requests for this attribute from the integrated database (in OID format) into the format expected by the relational database (as a string), and vice versa. In addition, the POPULATION, ELEVATION and AREA columns are all stored as type NUMBER, yet population has type long in the interface file, while elevation and area are doubles.

Oracle Database Wrapper

Relational Schema

```
CREATE TABLE COUNTRIES{
   NAME VARCHAR(30) NOT NULL,
CLIMATE VARCHAR(256),
HIGHESTPEAK NUMBER(4),
PRIMARY KEY(NAME) }

CREATE TABLE CITIES {
NAME VARCHAR(40),
COUNTRY VARCHAR(30) NOT NULL,
POPULATION NUMBER(4),
ELEVATION NUMBER(7,2),
AREA NUMBER(7,2),
PRIMARY KEY(NAME)
```

Garlic Schema

```
interface Country_Type {
attribute string name;
attribute string climate;
attribute long highest_peak;
};

interface City_Type {
attribute string name;
attribute ref<Country_Type> country;
attribute long population;
attribute double elevation;
attribute double area;
};
```

Hotel Web Site Wrapper

```
interface Hotel_Type {
attribute string name;
attribute string street;
attribute string city;
attribute string country;
attribute long postal_code;
attribute string phone;
attribute string fax;
attribute short number_of_rooms;
attribute float avg_room_price;
attribute short class;
void display_location( );
};
```

Each hotel listing on the web site contains HTML-tagged fields describing that hotel, and a URL to map the location of a particular hotel given its key. In the interface definition file, the HTML fields are represented as attributes of the Hotel object, each with an appropriate data type, though the web site returns all data in string format.

The map capability is exposed as the display-location method. It is the wrapper's responsibility to map names to the fields on the HTML page, and to convert data from strings into appropriate types.

Data Transformation in the Middleware: "Views are an important means of reformatting data, especially for the middleware, as the data resides in data sources over which the user has little control" [Ref.32]. "Views provide the full power of SQL to do type and unit conversions not anticipated by the wrapper, merging or splitting of attributes, aggregations and other complex functions" [Ref.31]. In Garlic, *object views* allow further restructuring of data. An object view creates a new "virtual" object. Every virtual object in Garlic is based on another object. Garlic uses the OID of the base object as the basis for the virtual object's OID. The LIFT function provided by Garlic is used to map the base OID to the virtual object's OID. Another advantage of virtual objects is we can define new methods for these objects independent from the base objects. Also we can lift the base objects' methods to virtual objects. [Ref.31]

In summary, object views enhance the wrapper transformations with a general view mechanism for integrating schemas. "Object views support integrated cooperative use of different legacy databases, through query language based transformations" [Ref.31]. Such transformations are required to integrate overlapping portions of heterogeneous databases.

Building the Integrated Schema: In Garlic, a tool called Clio is used to create mappings between two data representations. But this is integration is not an automatic process, rather a semi-automatic process with user input. The Garlic team considers Clio beyond the state of the art for the following reasons.

63

- Clio makes both the schema integration and data transformation at the same time while other tools developed in other projects make only one of them.

- It employs a full database middleware engine, "giving it significantly more leverage than the ad hoc collections of tools available today, or the lightweight "agents" proposed by others" [Ref.32].

- "It exploits the notion of a target schema, and where it exists, target data, to make the integration problem more tractable" [Ref.32].

- It allows more complex transformations of both schema and data, because the middleware engine is being enhanced with more powerful transformation capabilities.

Clio has three major components: a set of Schema Readers, which read a schema and translate it into an internal representation; a Correspondence Engine (CE), which finds matching parts of the schemas or databases; and a Mapping Generator, "which generates view definitions to map data in the source schema into data in the target schema" [Ref.31]. The CE has three major subcomponents too: a GUI for graphical display of the schemas and relevant data items, a correspondence generator, and a component to test correspondences for validity. Initially, "the CE expects the user to identify possible correspondences, via the graphical interface, and provides appropriate data from source and target (using the meta query engine) for verifying the correspondences and identifying the nature of the relationship (again, initially relying on the user)" [Ref.31]. This is an iterative process. Over time, the Garlic team anticipates

"increasing the "intelligence" of the tool using mining techniques so that it can propose correspondences, and eventually, verify them" [Ref.31].

b. The Garlic Data Model

Garlic has adopted the "ODMG-93 object model as a starting point for the Garlic data model and the syntax of the ODMG-93 object definition language, ODL, as a base for the Garlic Data Language, GDL" [Ref.32].

In the ODMG-93 standard, the fundamental building blocks of the data model are objects and values. Each object has an identity that uniquely denotes the object, thus "enabling the sharing of objects (by reference)" [Ref.32]. Objects are strongly typed, and object types are expressed in the data model in terms of object interfaces (as distinct from implementations). The description of an object's interface includes "the attributes, relationships, and methods that are characteristic of all objects that adhere to the interface" [Ref.32]. The model also supports an inheritance mechanism by which a new interface can be derived from one or more existing interfaces. The derived interface inherits all of the attributes, relationships, and methods of the interfaces from which it is derived, making the derived interface a subtype of those interfaces.

"In the ODMG-93 data model, an object's identity is both unique and immutable" [Ref.32]. Some repositories, such as relational databases, do not provide an identity. To enable such data items to be modeled as objects in a Garlic database, references in the Garlic data model are based on a notion that is called weak identity; this is simply a means of denoting an object uniquely but not necessarily immutably within the scope of a Garlic database. "An object's weak identity is formed by concatenating a

token that designates the object's implementation with an implementation-specific unique key" [Ref.32]. The Garlic data model extends the concepts of the ODMG-93 object model in three significant ways. The first is the degree of support for alternative implementations of interfaces, the second is related to type system flexibility, and the third is an object-appropriate view definition facility. [Ref.31]

Garlic makes a sharp distinction between an interface and its implementations. "The type of an object is determined solely by its interface, and any number of implementations of a given interface are permitted" [Ref.32]. It is quite possible that several repositories may offer alternative implementations of an important multimedia data type (e.g., text or image). "Garlic supports both the notion of a type extent, which is the set of all instances of a given interface, and an implementation extent, which is the set of all instances managed by a given implementation of an interface of interest" [Ref.32].

The most significant extension that Garlic makes to the ODMG-93 data model is the notion of views. In Garlic, the primary purpose of a view is "to enhance (extend, simplify, or reshape) a set of underlying Garlic objects, usually by adding or hiding some of their attributes and/or methods" [Ref.32]. Garlic employs the notion of an object-centered view for this purpose. "An object-centered view defines a new interface, together with an implementation of the new interface (usually written using Garlic's object query language), that is based on an existing interface that the view definer wishes to enhance" [Ref.31].

c.   *Queries in Garlic*

Given the schema for a Garlic database that a user or application wishes to utilize, Garlic provides access to the database through a high-level query language. The query language of Garlic is an object-oriented extension of SQL. "To accommodate the object-oriented nature of the Garlic data model, Garlic extends SQL with additional constructs for traversing paths composed of inter-object relationships, for querying and materializing collection-valued attributes of objects, and for invoking methods within queries" [Ref.31]. Since the Garlic query language is intended for querying databases that contain data in a variety of repositories, including multimedia repositories with associative search capabilities, Garlic's SQL extensions also take the needs of such repositories into account.

On the other hand, the SQL extensions can help to integrate approximate match query semantics with more traditional (exact match) database query semantics. This is done by introducing into SQL the notion of graded sets. In such sets, "each object is assigned a number between 0 and 1 for each atomic predicate; this number represents the degree to which the object fulfills the predicate, with 1 representing a perfect match" [Ref.32]. To enable query writers to specify the desired semantics, the syntax of SQL is extended to permit the specification of the number of matching results to be returned and whether or not rank-ordering (rather than an attribute-based sort order, or an arbitrary order) is desired for the query's result set.

Garlic decomposes a user's query into an execution plan containing a number of smaller queries, each of which can be executed by an underlying repository. It is the wrapper's job to translate these smaller queries into the repository's native query

67

language. To do this decomposition, "Garlic needs descriptions of the query processing power of each repository" [Ref.31].

d.   Summary

Garlic's goal is to build a heterogeneous multimedia information system capable of integrating data from a broad range of data repositories. The architecture of the system is based on repositories, repository wrappers, and the use of an object-oriented data model and query language to provide a uniform view of the disparate data types and data sources that can contribute data to a Garlic database. A significant focus of the project is the provision of support for repositories that provide media specific indexing and query capabilities.

What distinguishes Garlic from other projects in this area is its focus on providing an object-oriented view if data residing not only in databases and record-based files, but also in a wide variety of media-specific data repositories with specialized search facilities.

3.   DISCO

DISCO is a data integration project at Inria, Rocquencourt, France. As an acronym, DISCO stands for "Distributed Information Search COmponents". DISCO is a prototype heterogeneous distributed database that access underlying data sources [Ref.34]. "The data sources can be databases, files, dedicated data servers, or HTML pages [Ref.34]. Thus, data can be structured, unstructured or semi-structured. The main

goal of DISCO is to provide uniform and optimized access to the underlying data sources using a common declarative query language.

a.   *System Architecture*

To scale up to large number of data sources, DISCO adopted a mediator-distributed architecture of specialized components consisting of applications, mediators, wrappers, and data sources as shown in Figure 3.3.

Applications: End users interact with applications written by application programmers. Applications access a uniform representation of the underlying sources through a uniform (SQL-like) declarative query language (see subsection **d**).

Figure 3.3. DISCO System Architecture.

Mediators: Mediators encapsulate a representation of multiple data sources and provide a value-added service"[Ref.35]. Mediators provide the functionality of uniform access to multiple data sources. They typically resolve conflicts involving the dissimilar representation of knowledge using different data models and database schema, and "conflicts due to the mismatch in querying power of each server"[Ref.35]. The distributed architecture of DISCO permits DBAs to develop mediators independently and permits mediators to be combined, "providing a mechanism to deal with the complexity introduced by a large number of data sources" [Ref.35].

Wrappers: To deal with the heterogeneous nature of databases, wrappers make the subquery transformation. Wrappers map from a subset of a general query language, used by mediators, to the particular query language of the source. "A wrapper supports the functionality of translating queries appropriate to the particular server, and reformatting answers (data) appropriate to each mediator"[Ref.35]. The wrapper implementor writes wrappers for each type of database. DISCO supports "type transformations to ease the incorporation of new data sources into a mediator" [Ref.35].

b. Wrapper Interface

For the database implementers (DBI), DISCO provides a flexible wrapper interface. "DISCO interfaces to wrappers at the level of an abstract algebraic machine (AM) of logical operators" [Ref.36]. When the DBI implements a new wrapper, he or she chooses a set of logical operators to support. "The DBI implements the logical operators, and also implements a call in the wrapper interface which returns the grammar

describing the supported logical expressions" [Ref.36]. The mediator interacts with the wrapper in two phases. In the register phase, "the wrapper communicates to the mediator its local schema, its specification, and a optional description of the cost of operations in its algebra" [Ref.37]. During query processing, a DISCO mediator query optimizer generates a logical expression for the wrapper. The mediator calls the wrapper interface to get the "grammar describing the supported logical expressions, and checks that the logical expression generated by the optimizer is legal with respect to the grammar describing the wrapper interface" [Ref.38].

"Explicit specification of the data sources, as objects, in the DISCO data model, gives the DBA the capability to express queries that range over an unspecified collection of data sources" [Ref.36], or queries that refer to particular data sources. "Inclusion of the data source specification within the model also allows DISCO to support a new query processing semantics" [Ref.35]. The type hierarchy and mapping supported by the DISCO model, allows the mapping of multiple data sources to a single type of a mediator, and also allows the mapping of a data source to multiple types of a mediator. "This aspect of the DISCO model supports scaling to a large number of data sources" [Ref.34]. New data sources may also be incorporated transparently, if they map to the same mediator type.

c. *Mediator Data Model*

DISCO is based on the ODMG standard. The ODMG standard consists of an object data model, an object definition language (ODL), a query language (OQL), and a language binding. The ODGM object model is based on a type system. Types can be

atomic (integer, Boolean, string etc.) or structured. The structured types are the set, bag, list, or tuple. The expressions are constructed through the recursive application of structured type constructors to atomic types and type expressions. Object types are described in the data model thorough an object interface using ODL. An object interface specifies the properties and operations or methods that are characteristic of the instances of this object type. The object types are organized along a subtype hierarchy. An object type extent is a set of instances of a given object type. In the data model, "an interface defines a type signature for accessing an object" [Ref.35]. When objects are created or destroyed, the extent is updated automatically. "Extents are the primary entry point for access to data" [Ref.35].

The DISCO group extended ODMG ODL in two ways, to simplify the addition of data sources to a mediator. (1) **multiple extents:** "This extension associates multiple extents with each interface type defined for the mediators" [Ref.36]. (2) **type mapping:** This extension associates type mapping information between a mediator type and the type associated with a data source [Ref.36].

In addition to these extensions, two (standard) ODMG interfaces were defined; Wrapper models wrappers and Repository models repositories. A repository is "essentially the address of a database or some other type of repository" [Ref.35]. Repositories typically contain several data sources. Each data source in a repository is associated with in extent, and this provides the entry point to the data source. DISCO extends the concept of an extent for an interface, to include a bag of extents for the interface, for any type defined for the mediator. Each extent in the bag mirrors the extent of objects in a particular data source, associated with this mediator type. Since this

extension is fully integrated into the OBMG model, the full modeling capabilities of the ODMG model are available for organizing data sources. DISCO evaluates queries on extents and thereby on the date sources. [Ref.35]

d. *Mediator Query Processing*

"The mediator includes a query processor containing a query preprocessor, a query optimizer and a runtime system"[Ref.38]. It also contains a catalog that records information on local data, local schema, global schema, etc. This catalog is updated "when a source is registered with the mediator by importing the local schema, capabilities and cost information of the source" [Ref.38]. The import is accomplished via the wrapper associated with the source.

The processing of a query is accomplished in several steps.

Step 1: performs "the reformulation of the query into local schemas" [Ref.38]. The query is parsed and type checked against the global schema and then it is reformulated into the local schemas of the sources. "Reformulation is accomplished through view definitions and the application of maps" [Ref.38].

Step 2: performs logical search space generation. The reformulated query is transformed into a "preliminary tree of logical operators in the relational algebra that is equivalent to the query" [Ref.38]. The logical operators in the tree belong to a universal abstract machine (UAM) of logical operators implemented in the mediator. DISCO has the usual logical operators of scan, project, join, etc. In addition, "transformation rules rewrite trees to equivalent trees" [rerf.27]. Finally, an additional logical operator, submit, "expresses the mediator call to a wrapper" [Ref.37].

Step 3: performs preliminary query decomposition.

Step 4: compares the "functionality required for each wrapper with the capabilities exported by the corresponding wrapper and modifies the wrapper structure to use only the capabilities of the wrapper" [Ref.37].

Step 5: transforms the final logical operators into an execution plan by transforming the logical operations in the mediator composition tree to physical algorithms in the mediator runtime system. "The logical operations done by wrappers are not transformed because the corresponding physical algorithms are executed by the wrapper itself" [Ref.37]. This physical algorithm is responsible for calling the associated wrapper. "The mediator runtime system is based on the iterator model " [Ref.39].

Step 6: "assigns a cost to the execution plan by considering the cost (in terms of total time and statistical information) of the physical algorithms in the mediator and the costs of the logical operations on the wrappers" [Ref.38]. If a wrapper has exported cost equations, those are also considered.

Steps 2 through 6 are repeated until the execution plan with lowest cost is generated.

Step 7: executes the lowest cost plan. During query execution, the mediator calls the wrappers and passes the final wrapper operation plans. The wrappers evaluate their operation plans and send their results back to the mediator. "The mediator runtime system combines the results using execution plan which represents the final composition query" [Ref.38]. If a wrapper is unavailable, a partial answer is produced.

74

e. ***Summary***

DISCO's architecture is based on the wrapper-mediator architecture, extended with several novel features. DISCO's mediators and wrappers operate independently: a mediator accesses a wrapper simply through URL-like description of the wrapper. So the wrappers can easily be shared among multiple mediators. Each DISCO wrapper exports its capabilities using a grammar-like description of the operations that the wrapper supports. "Mediators automatically adopt to the capabilities of wrappers by using an elegant distinction between the preliminary execution plan and the final execution plan., which accounts for wrapper capabilities" [Ref.40].

4. Infomaster

Infomaster is an information integration system developed and tested at Stanford University. Infomaster provides integrated access to multiple distributed heterogeneous information sources on the Internet, thus "giving the illusion of a centralized, homogeneous information system" [Ref.41]. It can be said that Infomaster creates a virtual data warehouse. Infomaster handles "both structural and content translation to resolve differences between multiple data sources and the multiple applications for the collected data" [Ref.42].

In this subsection we show the system architecture and the basic tasks of the system.

a. System Architecture

Figure 3.4 shows the general architecture of the Infomaster system. As we said Infomaster is a generic information integration tool for integrating existing information sources. Information sources that can be integrated vary from SQL databases to WWW semistructured data. Infomaster connects these various information sources to system using wrappers. "There are several WWW interfaces to Infomaster, including forms based and textual" [Ref.41].

Facilitator: The core of Infomaster is a facilitator that determines "which source contain the information necessary to answer the query efficiently, designs a strategy for answering the query, and performs translations to convert source information to a common form or the form requested by the user" [Ref.42]. "Formally, Infomaster contains a model-elimination resolution theorem prover as a workhouse in the planning process" [Ref.41]. As we said, Infomaster has a WWW interface that can be used to enter queries using menus, SQL, or ACL. The queries are then converted into ACL and passed to the nearest facilitator. "This facilitator may call on the resources of information sources and other agents and other facilitators it knows about. Information sources handle queries over the data they store" [Ref.42]. Other agents may handle such tasks as doing specialized translations, such as currency conversion. "Facilitators may specialize in particular domains of interest" [Ref.42].

Figure 3.4. Infomaster System Architecture.

Wrapper: Wrappers are used to access information in a variety of sources. For SQL databases, there is a generic ODBC wrapper. There is also a wrapper for Z39.50 (i.e. bibliography libraries) sources. For legacy sources and structured information available through the WWW, a custom wrapper is used. "Currently, a custom wrapper is written to access housing rental advertisements from several San Francisco Bay area newspapers on the WWW" [Ref.43].

77

Basically, wrappers are programs that translate between the language spoken by the information source and the language spoken by the core Infomaster system. "The internal content language used by the Infomaster system is the knowledge interchange format (KIF), essentially a lisp like syntax for representing first order logic expressions" [Ref.41].

Knowledge Base: Infomaster uses rules and constraints to describe information sources and translations among these sources. These rules and constraints are stored in a knowledge base. "For efficient access, the rules and constraints are loaded into Epilog, a main memory database from Epistemics" [Ref.42].

An important property of Infomaster is that: Harmonizing n data sources with m uses doesn't require n x m sets of rules (i.e. for every pair of user and source interaction, we don't need a rule). "By providing Infomaster with a reference schema, database users are allowed to describe their schemas without regard for the schemas other users and providers" [Ref.41]. This strategy is shown in Figure 3.5. Translation rules describes how each source relates to the reference schema. These translation rules are bi-directional whenever possible, "so information stored in one source's format may be accessed through another source's format" [Ref.42]. "The same type of rules are used to describe how clients want to access data" [Ref.41]. These rules are combined and interpreted by Infomaster during query optimization and query processing. "Because these translations reference each other essentially through the reference schema, entry and maintenance of translation rules is enhanced" [Ref.41].

Interfaces: Infomaster includes a WWW interface for access through web browser such as Netscape Navigator or Internet Explorer. "This user interface has two

levels of access: an easy-to-use, forms based interface, and an advanced interface that supports arbitrary constraints applied to multiple information sources" [Ref.43]. However, additional interfaces can be created specific to aim without affecting the core Infomaster.

Figure 3.5. Infomaster Reference Schema Architecture.

Infomaster has also a programmatic interface called Magenta, which supports ACL (Agent Communication Language) access. "ACL consists of KQML (Knowledge Query and Manipulation Language), KIF (Knowledge Interchange Format), as well as vocabularies of terms" [Ref.41].

b.   *Query Processing*

In Infomaster system, the information integration problem is abstracted in a hierarchy. In this abstraction hierarchy, the user interfaces and information sources are modeled by a set of relations. The queries entered by the users are abstracted as interface relations. Data available from an information source can also be seen as a relation, which is called site relation. The information integration problem can be reduced in this framework to the problem of relating the interface and the site relation in an appropriate way.

Base relations relate the interface and site relations to each other. Specific to the task, the base relations are created and after that site and interface relations are converted to these base relations. So new information sources are easily integrated to system and user interfaces can be changed without affecting the site relations.

The query processing in Infomaster is a three-step process based on the abstraction described above. Assume a user asks a query q. This query is expressed in terms of interface relations. In a first step, query q is rewritten into a query in terms of base relations. This step is being called reduction. In a second step, the descriptions of the site relations have to be used to translate the rewritten query into a query in terms of site relations. This second step is being called abduction. The query in terms of site relations is an executable query plan, because it only refers to data that is actually available from the information sources. However, the generated query plan might be inefficient. Using the descriptions of the site relation, the query plan can be optimized.

Reduction: The reduction step in the query processing sequence is very simple. "It is essentially a macro expansion" [Ref.42]. The user query is given in terms

80

of interface relations are defined in terms of base relations. Therefore, the reduction step requires only substituting interface relations by the corresponding definitions.

Abduction: The interesting step in the query processing sequence is the abduction step. 'It requires to translate a query in terms of base relation into a query in terms of site relations" [Ref.42]. This is more complicated than the reduction step, because site relations are expressed in terms of base relations and not vice versa. This query-rewriting problem is well known in the database literature as the problem of answering queries using views. Infomaster group has developed an algorithm for this problem, but, since it is out of scope of our thesis, we won't show this algorithm here.

c. *Summary*

Infomaster is an information integration tool that provides integrated access to various heterogeneous information sources. The core of the system is a facilitator that dynamically determines an efficient way to answer user's queries. Different data sources are connected to system by wrappers.

The essential of the system is the conversion of user queries and information source relations into base relations. Therefore new user interfaces and information sources can be added to system without affecting the current structure of the system.

C.   SUMMARY

In this chapter, we have presented related work done about the heterogeneous multidatabase systems. At the beginning, we have discussed the motivation behind the multidatabase systems and important issues in multidatabase design. In subsection A.3 we reviewed the early projects about multidatabase systems took place in late 1980's. Among these projects, we gave a special importance to MRSDM which was the first well prototyped multidatabase system. In Section B, we have discussed the current projects in this area. Among these projects, Tsimmis provides a framework and tools to assist users in their information processing and integration activities. Garlic supports integration of data not only in databases and record based files, but also in a wide variety of media-specific data repositories with specialized search facilities. DISCO projects provide a uniform and optimized access to the underlying data sources using a common declarative query language. And finally, Infomaster provides integrated access to multiple distributed heterogeneous information sources on the Internet. Infomaster handles both structural and content translation to resolve differences between multiple data sources and multiple applications for the collected data.

IV. SEMANTIC SIMILARITIES BETWEEN DATA OBJECTS IN MULTIPLE DATABASES

In any approach to interoperability of database systems, the fundamental issue is that of identifying objects in different databases that are semantically related, and then resolving the schematic differences among semantically related objects. In this chapter, we focus on the techniques and representational constructs used by the various practitioners in the field of multidatabases. We also try to explain techniques for representing uncertainty but only as an aspect of the semantic similarity between objects.

So far, many attempts have been made to capture the similarity of objects by using mathematical tools, such as value mappings between domains, and abstractions, such as generalization, aggregation, etc. However, it is argued that the RWS of an object cannot be captured using mathematical formalisms. We need to understand and represent more knowledge in order to capture the semantics of the relationships between the objects. The knowledge should be able to capture and the representation should be able to express the context of comparison of the objects, the abstraction relating the domains of the two objects, and the uncertainty in the relationship between the objects.

In Section A, we explore the basics of these three perspectives of semantics. In Section B, we discuss semantic proximity, and in Section C, the context building approach is explained. The context interchange approach is covered in Section D, and in Section E we discuss the common concept approach.

A.   SEMANTICS: PERSPECTIVES AND REPRESENTATION

Semantics is defined as "the scientific study of the relations between signs and symbols and what they denote or mean," [Ref.44]. These signs and symbols can be considered to be aspects of real-world semantics (RWS) of an object. As stated before, there are three main perspectives of semantics: context, abstraction and uncertainty.

1.   Context: The Semantic Component

The context in which the objects are being compared provides semantic support for identifying and representing the object similarities. Some critical developments of research in the field of heterogeneous databases related to context are as follows:

- *Semantic proximity* (Section B) [Ref.46] characterizes semantic similarity in which the context is the primary vehicle to capture the RWS.

- *Dynamic context building approach* (Section C) [Ref.47] provides for meaningful information exchange between various information systems.

- *Context Mediation* (Section D) Sciore and Siegel [Ref.48] propose an approach that uses metadata to represent context thereby achieving context mediation

- *Common concepts* [Ref.49] (Section E) characterizes similarities between attributes in multiple databases.

2. Abstractions/Mappings: The Structural Component

Abstraction in this chapter refers to the relation between the domains of two objects. However, since abstractions by themselves cannot capture the semantic similarity, they must be linked using either context or additional knowledge in order to capture the RWS. "Mapping between the domains of objects is the mathematical tool used to express the abstractions" [Ref.45]. Some of the approaches are as follows:

- Define abstractions in terms of value mappings between the domains of objects and associate them with the context as a part of the semantic proximity (Section B) [Ref.46].

- Define mappings between schema elements, which they term interschema correspondence assertions or ISCAs (Section C). A set of ISCAs under consideration defines the context for integration of the schemas. [Ref.47]

- Define mappings that they call conversion functions (Section D), which are associated with the meta-attributes that define the context. [Ref.48]

- Associate the attributes with "common concepts" (Section E). Thus the mappings (relationship) between the attributes are determined through the extra knowledge associated with the concepts. [Ref.49]

3. Modeling Uncertainty, Inconsistency, and Incompleteness

Understanding and representing semantic similarity between objects may involve "understanding and modeling uncertainty, inconsistency, and incompleteness of the information pertaining to the objects and the relationships between them as modeled in the database" [Ref.45]. Some approaches are to: (a) model uncertain information by

85

using the degrees of likelihood of the various intermediate contexts [Ref.47] (Section C); (b) use semantic proximity as a basis for representing the uncertainty of the information modeled at the database level [Ref.46] (Section B); and, (c) represent each attribute as a vector depending on the concepts associated with it. A similarity measure between two attributes is defined as a function of the vectors associated with the attributes [Ref.49].

B. SEMANTIC PROXIMITY: A MODEL FOR REPRESENTING SEMANTIC SIMILARITIES

There are two basic keys to the representation of semantics. The first focuses on identifying the real-world semantics of various entities and the relationships between them. The second is to represent all known knowledge about the domain at hand. Once these issues have been addressed, the relationships between various objects can then be determined on the basis of the encoded domain knowledge.

The concept of semantic proximity is introduced to characterize semantic similarities between objects and use it to provide a classification of semantic similarities between objects [Ref.46]. Given two objects O1 and O2, the *semantic proximity* (Figure 4.1) between them is defined by the 4-tuple given by:

semPro(O1, O2) = (Context, Abstraction, (D1, D2), (S1, S2))

where Di is domain of Oi, and Si is state of Oi.

1. Context(s) of the Two Objects: The Semantic Component

Each object has its own context. The term context in *semantic proximity* refers to the context in which a particular semantic similarity holds. This context may be related to or different from the contexts in which the objects were defined. It is possible for two objects to be semantically closer in one context than in another context. Thus, the respective contexts of the objects, and the abstraction used to map the domains of the objects, both help to capture the semantic aspect of the relationship between the two objects" [Ref.46]

Some of the alternatives for representing a context in an interoperable database system are as follows:

Figure 4.1. Semantic Proximity Between Two Objects.

- Context is defined as the knowledge that is needed to reason about another system, for the purpose of answering a query, and is specified as a set of assertions. [Ref.47]

- Context is defined as the meaning, content, organization, and properties of data and is modeled using metadata associated with the data. [Ref.49]

- A context may be linked with a database or a group of databases [Ref.45].

- The relationship in which an entity exists may determine the context of the entity [Ref.45].

- At a very elementary level, a context can be thought of as a named collection of the domains of the objects [Ref.45].

2. Issues of Representation and Reasoning

Various approaches have been proposed for representing semantic structures in the context of multidatabase systems. similar to context and for reasoning with the help of these representations. Kashyap and Sheth [Ref.50] use context to provide intentional descriptions of database objects. They represent contexts as a collection of *contextual coordinates* and *values*.

A contextual coordinate indicates an aspect of context that models a characteristic of the subject domain. The coordinate may be obtained from a domain specific ontology. Values can be a set of symbols, objects from a database, or concepts from a domain-specific ontology. Operations to compare the specificity of two contexts and to compute the greatest lower bound of two contexts are defined.

88

In another similar approach, a context is represented as a collection of meta-attributes and their values (Section D) [Ref.48]. This representation of context is at the level of data values and object instances. "Constraints are not able to be modeled at an intentional level (namely, cardinality constraints)" [Ref.48]. Context mediation is used for reasoning and is implemented using rules and predicates in a relational model.

3. The Vocabulary Problem

In constructing contexts and intentional descriptions for modeling semantics, the choice of terminology specific to the subject domain is critical. (WHY?) To address this critical need, traditional multidatabase approaches are now using ontologies for building semantic descriptions. An *ontology* may be defined as, "the specification of a representational vocabulary for a shared domain of discourse, which may include definitions of classes, relations, functions, and other objects" [Ref.51].

Relationships among the terms listed in ontologies enable representation of extra information in the contextual descriptions. Concept hierarchies have also been used in the common concepts approach [Ref.49] (Section E). Terminological relationships have been represented across ontologies to "handle cases where contexts or intensional descriptions may be constructed from different ontologies" [Ref.49]. An approach for resolving representational conflicts using terminological relationships is discussed in Chapter V (Remote-Exchange approach).

4.   The Structural Components

In this section, we discuss the three structural components of semantic proximity, *abstraction*, *domain*, and *states*.

a.   *Abstraction Used to Map the Objects*

The term "abstraction" is used to refer to a mechanism that maps the domains of the objects to each other, or to the domain of a common third object. An abstraction by itself cannot capture the semantic similarity. Some of the more useful and well-defined abstractions are listed here [Ref.45]:

- *Total 1-1 value mapping:* For every value in the domain of one object, there exists a value in the domain of the other object and vice versa.

- *Partial many-one mapping:* In this case, some values in the domain of one of the objects might remain unmapped, or a value in one domain might be associated with many values in another.

- *Generalization/specialization:* One domain can generalize or specialize the other, or domains of both the objects can be generalized/specialized to a third domain.

- *Aggregation:* One domain can be an aggregation, or collection, of other domains.

- *Functional dependencies:* The values of one domain might depend functionally on the other domain.

- *ANY:* This term is used to denote that any abstraction (such as the ones defined here) may be used to define a mapping between two objects.

- *NONE:* This term is used to denote that no mapping is defined between two semantically related objects.

b. *Domains of the Objects*

Domains refer to the sets from which the objects can take their values. When using an object-oriented model, the domains of objects can be considered as types, whereas the collections of objects might themselves be of as classes [Ref.45]. A domain can be either atomic (i.e., cannot be decomposed any further) or composed of other atomic or composite domains. The domain is, in other words, a subset of the product resulting from the crossing of the domains of the properties of the object [Ref.45]. Figure 4.2 shows a domain of an object and its attributes.

Figure 4.2. Domain of an Object and its Attributes.

An important distinction between a context and a domain should be noted. One of the ways to specify a context is as a *named collection of the domains of objects*, (i.e., it is associated with a group of objects.) A domain, on the other hand, is a property of an object and is associated with the description of that object.

c. States (Extensions) of the Objects

The state of an object can be thought of as an extension of an object recorded in a database or databases. Two objects having different extensions can have the same state real-world semantics (and hence be semantically equivalent).

5. A Semantic Classification of Object Similarities

Here, the emphasis is on identifying semantic similarity independent of the representation of the objects. The concept of semantic proximity defined earlier provides a means of qualitative measurement to classify the semantic similarities between objects.

a. The Role of Context in Semantic Classification

A partial context specification can be used by humans to decide whether context for modeling of two objects is the same or different, and whether the comparison of semantic similarity of objects is valid in *all* possible contexts or *specific* ones [Ref.45]. For semantic similarity in all possible contexts, the semantic proximity between the objects is defined with regard to all known and coherent comparison contexts. (Figure 4.3 shows the types of semantic proximity.) In this case, there should be coherence between

(a) the definition contexts of the objects being compared, and (b) between the definition contexts and the context of comparison.

For subcontexts with similar content, we might be interested in the semantic proximity between two objects in contexts that are more specific or more general with regard to the context of comparison. For dissimilar context labeled as *None*, no context exists in which a meaningful abstraction or mapping between the domains of the objects may be defined. This is the case when the definition contexts of the objects being compared are not coherent with each other.

When the semantic proximity between the objects is defined with regard to *some specific* context, this context may be constructed according to its *Greatest Lower Bound* (GLB) or *Least Upper Bound* (LUB). For the context of the two objects, we are interested in the GLB of the context of comparison and the definition context of the object. For the LUB, we are concerned with the definition contexts of the two objects when no abstraction/mapping exists between their domains in the context of comparison.

1. *Semantic Equivalence*: Two objects are defined as *semantically equivalent* when they represent same real-world entity or concept, and is the strongest measure of semantic proximity two objects can have. It means that given two objects O1 and O2, it should be possible to "define a total 1-1 value mapping between the domains of these two objects in any known and coherent context" [Ref.45]. The notion of equivalence, known more accurately as *domain semantic equivalence*, depends on the definition of the domains of the objects. A stronger notion of semantic equivalence between two objects is one that

93

incorporates the state of the databases to which the two objects belong. This equivalence is called *state semantic equivalence* and is defined as:

$$semPro(O1, O2) = (ALL, M, (D1, D2), (S1, S2))$$

where M is a total 1-1 value mapping between (D1, S1) and (D2, S2).Thus we can write it as:

$$semPro(O1, O2) = (ALL, total\ 1\text{-}1\ value\ mapping, (D1, D2), Don't\ care)$$

2. *Semantic Relationship*: Two objects are said to be *semantically related* when "there exists a partial many-one value mapping, a generalization, or aggregation abstraction between the domains of the two objects" [Ref.46]. This type of semantic similarity is weaker than semantic equivalence. Here we relax the requirement of a 1-1 mapping in a way that given an instance O1, we can identify an instance of O2 but not vice versa. The requirement that the mapping be definable in all the known coherent contexts is not relaxed [Ref.46]. Thus we define the semantic relationship as

$$semPro(O1, O2) = (ALL, M, (D1, D2), Don't\ care)$$

where M may be a partial many-one value mapping, generalization, or aggregation.

Figure 4.3. Semantic Classification of Object Similarities

3. *Semantic Relevance*: We consider two objects to be *semantically relevant* if they can be "related to each other using some abstraction in some context" [Ref.46]. Thus the notion of semantic relevance between two objects is context dependent, (i.e., two objects may be semantically relevant in one context, but not so in another.) Objects can be related to each other using any abstraction.

semPro(O1, O2) = (SOME, ANY, (D1, D2), Don't care)

4. *Semantic Resemblance*: The weakest measure of semantic proximity, semantic resemblance, might be useful in certain cases. Here, we consider the case in which the "domains of two objects cannot be related to each other by any abstraction in any context" [Ref.46]. Hence, the exact nature of semantic proximity between two objects is very difficult to specify. In this case, "the user may be presented with extensions of both the objects" [Ref.46]. In order to express this type of semantic similarity, an aspect of context, called role, is introduced, and semantic resemblance is expressed with its help [Ref.46].

$$semPro(O1, O2) = (SOME(LUB), NONE, (D1,D2), Don't care)$$

5. *Semantic Incompatibility*: While all the qualitative proximity measures defined so far describe semantic similarity, semantic incompatibility asserts semantic dissimilarity [Ref.46]. It should be noted that a lack of semantic similarity does not automatically imply that the objects are semantically incompatible. Establishing semantic incompatibility requires asserting that "the definition contexts of the two objects are incoherent with regard to each other and there exist no contexts associated with these objects such that they have the same role" [Ref.46].

$$semPro(O1, O2) = (NONE, NONE, (Di, D2),$$

C.   CONTEXT BUILDING APPROACH

This approach centers on the construction and maintenance of the context, within which meaningful information between heterogeneous information systems is proposed

for exchange. The context is defined as a set of *interschema correspondence assertions (ISCAs)*. Each ISCA consists of three dimensions: naming, abstraction, and level of heterogeneity. A classification of semantic conflicts is used as a basis for building and refining the context by discovering the ISCAs between corresponding elements of the component systems. The mathematical formalism used to model the structural similarity is expressed as the last two dimensions of the ISCA, abstraction and level of heterogeneity. The association between the abstraction and the context is achieved through the ISCAs that embody the correct context.

Context is described here as the "knowledge that is needed to reason about another system, for the purpose of answering a specific query. The context must provide an easily understood representation of how much is known and what is still needed in order to answer the query" [Ref.47]. The (partial) schemas and the subsequent query in the example [Ref.47] below are used to elaborate on the issues of context building.

EXAMPLE-1

Database: A database of a computer manufacturer

Data model: Relational

Schema: **COMPETITOR** (Name, Location, MarketShare, NetIncome)

Database: Financial information of companies in the US Community

Data model: Relational

Schema: **COMPANY** (Rank, Co-Name, Location, Industry)

Query: "What is the performance of my competitors in the US?"

1.   Context-Dependent Interpretation

Distinct contexts can lend different interpretations to a schematic conflict. [Ref.47]. The processes of detection of conflicts and integration help build a context dynamically, as the query itself "may be important in providing a context for the meaningful interpretation of schematic elements," [Ref.47]. Consider Example-2 [Ref.47], based on the schemas and query shown in Example-1, which illustrates the above (THE ABOVE WHAT?)

EXAMPLE-2

Hypothesis: COMPETITOR.Name and COMPANY.Co-Name appears to be related.

Context: Suppose the term "competitor" is the specialization of the term "company." The query specifies "competitor" and provides a context. *Interpretation*: We may conclude that COMPETITOR..Name and COMPANY. Co-Name are synonyms.

a.   *Organization of Context by Levels of Heterogeneity*

The object level is considered to be a higher level of heterogeneity than the attribute level, which is coarser than the instance level. In addition to structural schematic levels of heterogeneity, there are metadata levels of heterogeneity. These include (a) the differences requiring knowledge describing the objects and the attributes of the schema (also known as de*scriptive metadata*); (b) knowledge of the semantics that are inherent, implicit, and explicit in data models; and (c) general or domain-specific

98

knowledge about the database itself. Concepts at the metadata level could be objects, attributes, or instances (see Example-3 [Ref.47]).

EXAMPLE-3

The interschema correspondence assertion (ISCA) that COMPETITOR.Name and COMPANY.Co-Name are corresponding attributes, triggering an attempt to place them in the context of the respective objects they describe. This may result in the inference of an ISCA that COMPETITOR and COMPANY are corresponding objects. This is called *upward propagation* through the levels of heterogeneity. If there is enough evidence to refute the synonymy of COMPETITOR and COMPANY, this evidence would trigger a *downward propagation* to reclassify the relationship of COMPETITOR.Name and COMPANY.Co-Name as homonyms.

The semantic knowledge of another system, organized by levels of heterogeneity, provides a context for interpreting the view of the system" [Ref.47].

b.   *Classification and Representation of Semantic Conflicts*

The classification and representation of semantic conflicts is the founding principle behind dynamic context building. Conflicts are classified along the three dimensions of naming, abstraction, and level of heterogeneity. The semantic relationship between two elements of different databases is represented as an interschema correspondence assertion as follows [Ref.47]:

Assert[x, y] (naming, abstraction, heterogeneity)

The first two dimensions (naming and abstraction) include "what they view fundamental relationships between semantic concepts" [Ref.47]. The third dimension (level of heterogeneity) is needed to place the semantic relationship in the appropriate schematic context (see Example-4 [Ref.47]).

Naming conflicts refer to the relationship of the object, attribute, or instance names. These conflicts include synonyms and homonyms. "If a conflict cannot be categorized as either a synonym or a homonym, it is classified as unrelated" [Ref.47].

A conflict can involve objects that refer to the same class of objects, objects that represent similar semantic concepts at different levels of *abstraction* (generalization/specialization), an object that maps to a group of objects in another database (aggregation/part-of), or an incompatibility that occurs when one object can be mapped to another through a computed or derived function" [Ref.47].

EXAMPLE-4

The conflict between COMPANY.Co-Name and COMPETITOR.Name can be classified as:

Assert[COMPANY.Co-Name, COMPETITOR.Name]

(synonyms, generalization, (attribute, attribute))

"Assertions that have been inferred through upward propagation require reconciliation to fill the slots of the classification" [Ref.48].

Assert[COMPANY, COMPETITOR](?, ?, (object, object))

D. CONTEXT INTERCHANGE APPROACH

The context of data is essentially a collection of meta-attributes that model the associated metadata. When we consider the problem of transferring data from one environment to another and its manipulation in a new environment, we can use the context to cope with the meaning, content, organization, and properties of the data. We can also use context to model the environment to which the data belongs. [Ref.48].

The mathematical formalism used to model the structural similarity is expressed using conversion functions. These *conversion functions* are used to translate data from one context to another, thus enabling context interchange. Achieving an association between the mappings (conversion functions) and the context is accomplished by defining a conversion function for each meta-attribute that may be a part of the definition of a context.

1. Context and Metadata

A data source's metadata defines the *export context* and consists of stored information or rules describing the data provided by the source. A data receiver's metadata is called its *import context* and consists of predicates describing the properties of the data expected by the receiver [Ref.48].

EXAMPLE-5

A data source provides trade prices for stocks on the NYSE. The context of a trade price value might be the latest trade price in U.S. dollars. Data from this

environment may be moved to the environment of a Japanese company where the context might be the latest trade price in yen.

During the data interchange in Example-5 [Ref.48], the export context of the source is compared to the import context of the receiver to determine whether the data is meaningful to the receiver. Thus it is necessary to represent context using context specifications, and to compare those specifications to achieve context mediation. [Ref.48]. The idea is to represent all contexts from the data environment at the attribute level, using meta-attributes. "Meta-attributes are attributes that have a special relationship to the attributes whose context they define" [Ref.48]. For the financial data source discussed in Example-5, Currency and PriceStatus would be meta-attributes of the TradePrice attribute. Consider the schema in Example-6 [Ref.48] for the financial data source.

EXAMPLE-6

TRADES(InstrumentType, CompanyName, Exchange,

TradePrice(PriceStatus, Currency))

An example tuple can be given by

('equity', 'IBM', 'NYSE', 89.25('latestTradePrice', 'USDollars'))

2. Data Conversion (Conversion Functions)

A *conversion function* alters the values of a tuple's attributes and meta-attributes in a way that renders the meaning of the information unchanged. Each meta-attribute has a conversion function defined for it; by default, the conversion function for M is called cvtM. For example, suppose that a tuple in the TRADES relation has a TradePrice

value of 3 and a TradePrice.Currency value of 'USDollars'. The conversion function call cvtTradePrice (3, 'USDollars', 'Yen') returns the value 330 (assuming 1 U.S. dollar = 110 yen). The function cvtTradePrice is an example of a total and lossless conversion function. A *total function* is one that is defined for all arguments, and a *lossless function* is one for which every conversion has an inverse (AN INVERSE WHAT?)" [Ref.45]. Not all functions have these features.

a.   *Context Mediation*

The *context mediator* component examines each incoming tuple and attempts to build a new conversion function that will produce a tuple that conforms to the receiver's assumptions. [Ref.48]. The receiver's assumptions about the incoming tuple are described by the receiver's import context—"a predicate on the tuple's attribute values" [Ref.48]. The task of the context mediator is to return a conversion function that maps the input tuples to the receiver's import. context. This function is used to define a relational operator called cvtContext (see Example-7 [Ref.48]).

EXAMPLE-7

Suppose the TRADES table contains values for the attribute TradePrice with TradePrice.Currency = 'USDollars'

Suppose the receiver context is TradeCurrency = 'Yen'

The conversion function used will be:

cvtTradePrice(X, 'USDollars', 'Yen')

The relational operator can be defined using the above conversion function:

103

cvtContext(TRADES, TradePrice.Currency = 'Yen')

E.   COMMON CONCEPTS: AN APPROACH TO DETERMINE ATTRIBUTE SIMILARITIES

The context is implied in the representation of the functional definition of the concepts. The attributes are associated with the common concepts. Thus the mappings (relationships) between attributes are determined through "their association with the extra knowledge or the implicit context embodied in the common concepts" [Ref.49]. Uncertainty in the relationship between two attributes is modeled using similarity values. Each attribute is defined as a vector depending on the concepts associated with it. "The *similarity value* between two attributes is a function of e vectors associated with the attributes" [Ref.49].

1.   Representation of Attribute Semantics by Common Concepts

Each attribute can be characterized by a set of *common concepts*. Each concept represents a certain characteristic or a property that may be possessed by many objects (physical and/or abstract objects--see Example-8). These concepts are generic, (i.e., they are not application dependent.)

EXAMPLE-8

The common concepts horizontal-position *and* vertical-position *represent the horizontal and vertical position of a point, respectively.*

The common concept identification *can be associated with the attributes* sensor, mission, *and* platform *in the NASA databases, as these attribute have the property of identifying object.*

Concepts may have hierarchical relationships with other concepts. "Two relationships of particular significance are aggregate concept hierarchies an IS-A concept hierarchies" [Ref.48].

a. *Aggregate Concept Hierarchies*

The example of the concept *time-interval* is considered, which specifies a duration of time. It (WHAT?) could be modeled as an aggregate concept of the concept *begin-time* and *end-time*:

$$time\text{-}interval = aggregate(begin\text{-}time,\ end\text{-}time)$$

The attributes tpstart and tpstop can be characterized, respectively, by

time-interval.begin-time and *time-interval.end-time*

The two attributes are recognized to be an aggregate attributes and are characterized by the aggregate concept *time-interval.*

Consider the possible attribute similarity between (start-date, length) and (tpstart, tpstop). There are two levels of possible similarity [Ref.49]:

- *Individual similarity*: There is a similarity between start-date and tpstart as they are associated with the concept *time-interval.begin-time.*

- *Aggregate similarity*: There is a similarity between (start-date, length) and (tpstart, tpstop) as they are associated with the concept *time-interval.*

It should be noted that, a similarity with respect to aggregate attributes does not necessarily ensure that a similarity exists among individual attributes involved in the aggregate attributes. [Ref.49]. The attribute **start-date** is associated with *time-interval.begin-time*, but the attribute **length** is not associated with *time-interval.end-time*. For each aggregate concept in a hierarchy, the existence of a component labeled, "others" illustrates the lack of completion of any preexisting hierarchy, and enables the user to specify other component concepts.

b.   IS-A Concept Hierarchies

The IS-A concept hierarchy is the generalization specialization used in most semantic data models and object-oriented data models. [Ref.45]. If an object characterized by concept X is a subset of objects characterized by concept Y, then we call Y the generalization of X (equivalently, X is the specialization of Y). X is also called a *subconcept* of Y, and Y is called the *superconcept* of X. For example, the concept *square* is a subconcept of *rectangle*, which in turn is a subconcept of *quadrangle*.

F.   SUMMARY

In this chapter, first we present the important of knowledge and representation in order to capture the semantics of the relationships between the objects. The knowledge should be able capture and the representation should be able to express the context of comparison of the objects.

106

In Section A, we discuss the perspectives of semantics (context, abstraction, and uncertainty) and how the various researchers have dealt with these issues.

In Section B, the concept of *semantic proximity* is introduced to characterize semantic similarities between objects. Explanation is provided to demonstrate how semantic proximity is employed to provide a means of classification of semantic similarities between objects.

We have discussed the *context building approach* in Section C which centers on the construction and maintenance of the context, within which meaningful information between heterogeneous information systems is proposed for exchange. The context is a defined set of Interschema Correspondence Assertions (ISCAs).

The *context interchange approach* is presented in Section D. In this approach, data is transferred from one environment, and is then manipulated in a new environment. Conversion functions are used to change values of tuple's attributes and meta-attributes in a way that leaves the meaning of the information unchanged. Finally, we discuss the *common concepts approach*. In this approach, the mappings between data objects are determined through their association with the extra knowledge or the implicit context embodied in the common concepts.

In the next chapter, we will present the integration of information in different heterogeneous data sources.

THIS PAGE INTENTIONALLY LEFT BLANK

V. INTEGRATION OF INFORMATION IN DIFFERENT DATA SOURCES

In this chapter, we discuss the information integration techniques to ensure a homogeneous view of information sources and present our ideas about these techniques. Even though there are many approaches to multidatabase system designs, so far only the problems with database engineering have been addressed. We believe that a general methodology for information integration can be adapted to all these approaches. So in this chapter, first we examine the general principles of data integration.

As far as data integration is concerned, we believe that the type of information or data to be integrated is very important. Therefore, we make a new classification for the application areas of information integration. One is a domain specific application area where integration is required in a closed system such one consisting only of related data, such as medical, geographical, military etc. The other area is more general and is completely open to any kind of databases and information. The best example of the latter is the World Wide Web (WWW).

The related work and architecture mentioned thus far can be placed under one of these two classifications. The earliest projects developed (discussed in Chapters II and III) are clearly in the domain specific application area, because the motivation behind these projects was to establish a virtually homogeneous database used in specific organizations and companies. The projects defined as "current developments" can be classified in the second category, since the motivation was to use the current network capabilities more efficiently and to make the Web a virtual unique homogeneous information source.

It is apparent that a closed system is more tightly coupled and the number of member information sources is relatively very small, so the information integration problem is not as challenging as it is in an open system. The knowledge stemming from the classical database development era forced engineers to adopt the same techniques in classical databases. Among all the systems developed, the general approach was almost the same, to develop a stored global database definition (Meta-data) and a global database management system. The information integration problem came onto the scene when global meta-data was introduced. Then the problem was divided into two parts:

1. How the structural differences among the member databases can be eliminated; and,

2. How semantic heterogeneity at all levels (entity, attribute and extension) can be eliminated.

The general name of methodologies developed to address these problems is called *schema translation process*. We will examine this process in detail in subsequent sections, but suffice it to say at this point that this process is not fully automated and requires human interference at many points.

On the other hand, when an open system is concerned, it seems impossible to translate all local schemas into a unique global schema, because the number of member information sources can increase rapidly. This characteristic makes it impossible to interpret and combine different structures and semantics into a homogeneous structure, at least by human beings. So the shift in interest went towards another approach: wrapper-mediator architecture. The idea behind this approach was that, instead of

establishing a global schema of the whole system statically, (only allow changes when deemed necessary on a rare occasion), use only the integration of relevant information sources and make the integration at run time when needed.

Of course, this new architecture entailed new problems. The first problem is similar to the schema integration problem, but in this case the local schemas are more heterogeneous and the data may not be wholly structured (e.g., data might reside in a text file instead of a structured database). The most common approach to address this problem is to use a global data model. Second, the contents of member information sources are so irrelevant that the system, in the beginning, has no idea about the location of a specific data. So the relevant data must first be found among all distributed heterogeneous information sources. We examine this *location identification problem* in subsequent sections under the heading, "*semantic networks.*"

In the chapter layout, first we examine the resolution of representational diversity in multidatabase systems. After that we discuss the schema integration process that takes place between the local schemas and a global schema, or a global data model. And finally we discuss the semantic networks and the usage of an electronic lexicon "WordNet" to resolve the semantic heterogeneity problem.

A. RESOLUTION OF REPRESENTATIONAL DIVERSITY IN MULTIDATABASE SYSTEMS

The occurrence of representational differences that exist among related data objects in different local information sources is a major problem that frequently occurs

when attempting to support information sharing among autonomous heterogeneous database systems. In this context, a key to supporting sharing involves merging nonlocal data into the schema of an importing component as efficiently as possible. [Ref.52]. Support the resolution of semantic and representational (modeling) differences between the local and nonlocal perspectives is essential to achieving a graceful merger. Resolving representational differences indicates two steps:

1. Determine to as great an extent as possible the relationships between sharable objects in different components, and

2. Detect possible conflicts in their structural representations that will produce problems.

Especially during the early stages of multidatabase systems, many approaches were adopted and different methodologies were proposed for establishing relationships among data objects and conflict resolution. We describe these approaches in more detail in Chapter-IV.

1.   Heterogeneity in a Collaborative Sharing Environment

In order to support sharing of information among a collection of autonomous heterogeneous databases, many aspects of heterogeneity must be overcome (e.g., hardware heterogeneity, operating system heterogeneity), that make it difficult for components to cooperate. Heterogeneity is the natural consequence of the independent development and evolution of autonomous databases that serve fulfill the requirements of the application system they belong to. And, there is not total agreement with respect

to the clear definition and scope of the problem. [Ref.53]. In this section, we concentrate on a specific kind of heterogeneity that occurs at the database system level called *representational heterogeneity*. By this we mean variations in the semantics and structures of data in different components.

In this section, we examine the different kinds of heterogeneities that may occur as well as the causes of the representational diversities occurring in this heterogeneity.

a. *Types of Representational Heterogeneities*

Within the context of a multidatabase system with heterogeneous, autonomous components, we can a characterize a broad range of representational heterogeneity based on the following five levels of abstraction:

1. *Metadata language (Conceptual database model):* The components may use different collections of techniques for combining the structures, constraints, and operations used to describe data. Different data models provide different structural primitives such as records in the relational data model or objects in the functional data model, and operations for accessing and manipulating data such as QUEL and QSQL. Note that even when two database systems support the same data model, the differences in their data definition languages may still contribute to semantic heterogeneity. [Ref.52].

2. *Metadata specification (conceptual schema):* While the components share a common metadata language (conceptual database model) mentioned above, they may have varied conceptual schemas (specifications of their data). For

113

example, in two relational databases, the same data object may be represented as an entity in one database and as a relation in the other one.

3. *Object comparability*: The components may agree on a conceptual schema, however, there may be differences in the way information facts are represented [Ref.54]. This range of heterogeneity also encompasses the ways in which information objects are identified, and the interpretations of atomic data values as denotations of information modeled in a database. [Ref.52]. For example, a ship may be represented by Hull Number (e.g., S-351) in one database while it is being represented by the International Name (e.g., TBCK) in another one. Despite a difference in their type names, they represent the same data object. But note that, from the previous representation we can deduce that "this ship is a submarine," and this information is considered missing information in the other representation. Similarly, we can understand that this ship belongs to the Turkish Navy (i.e., *TB* represents Turkish Navy) from the latter type name, and this is also considered missing information in reference to the first database.

4. *Low-level data format*: Although the components agree at the model, schema, and object comparability levels, these same components may use different low-level representation techniques for atomic data values [Ref.52]. If we give an example similar to the previous one using ships, the conflicting difference in representation technique refers to the problem arising when values such as the length of a ship are represented using different units (meter vs. feet).

5. *Tool (database management system)*: The components may use different tools to manage and provide an interface to their data. This kind of heterogeneity may exist with or without the varieties described immediately above. This kind of heterogeneity is due to the fact that components may use a different DBMS.

We assume that the heterogeneities of type 1 and 5 are not as trivial as the 2, 3, and 4, because, type 1 (conceptual database model) heterogeneities can be eliminated by ensuring that the components utilize a common data model in a small federation. Or, a common data model can be adopted for larger systems with appropriate translations for local database models. When the common global data model is concerned, we propose that a universal standard must be ensured among the database project groups. But as far as we have seen in the literature, there is still a debate between different research groups about the kind of common data model. Although object-oriented data models take an advantage in this competition, there are still problems that prevent the full development of translation protocols between every type of data model.

But we don't consider this issue so important especially when the semantic heterogeneity problem is concerned, so further discussion on this issue is not presented in this section. Furthermore, no tool heterogeneity (type 5 in the spectrum) is not described in further detail as it is not particularly relevant to the discussion here. Instead, the remainder of this section will focus on resolving heterogeneities of type 2,3, and 4.

b. *Causes of Representational Diversities*

According to Batini et al. [Ref.55], there are three main causes of representational heterogeneity:

1.   *Different perspectives and needs:* This is a modeling problem occurring most frequently during the design phases of a database schema. Different user groups or designers adopt their own ways when modeling the same information, especially in this case of naming conventions as used in modeling. Referring back to our classic example, some may want to represent a ship as an object by hull number, and others may want to use an international name.

2.   *Equivalent constructs:* Data models, with their rich set of constructs, provide for a large number of modeling possibilities resulting in variations in the conceptual database structure. [Ref.54]. Generally, in conceptual models, several combinations of constructs can effectively model the real world domain. [Ref.52]. For example, a many-to-many relationship between two types can be modeled as several one-to-one relationships.

3.   *Incompatible design specifications:* Different design specifications result in different schemas. Cardinality constraints particularly stand out in design specification diversities. In one specification, an exact cardinality (1, 2, 3, ...) may be used, while a more general "many" (m) is being used in another.

Until this point, we have discussed the nature of semantic heterogeneity and identified the causes of such diversities. In the following sections, we present the details of a mechanism called *Remote-Exchange*, [Ref.56] for resolving semantic heterogeneity. Although other to resolving semantic heterogeneity shows small yet significant differences, we believe that the Remote-Exchange mechanism can encompass all the rationales behind these approaches as well as the potential they possess.

2. Remote-Exchange Architecture

Remote-Exchange is an architecture that supports the controlled sharing of information among a collection of autonomous, heterogeneous database systems. In Remote-Exchange, unification of remote objects with local objects plays a vital role.

The four major components of Remote-Exchange are the core object data model (CODM), the remote sharing language (RSL), the local lexicon, and the semantic dictionary. Figure 5.1 [Ref.56] provides a schematic overview of Remote-Exchange. In the following sections, we introduce each component separately. (A detailed description of the role of each component is given in Section A.4.)

a. Core Object Data Model (CODM)

Before any collaboration among the heterogeneous components of a federation is possible, a common model for describing the sharable data must be established. This model must be semantically expressive enough to capture the intended meanings of conceptual schemas that may reflect several or all of the kinds of

117

heterogeneity mentioned above. Further, beside the richness of this model, it must also be simple enough so that it can be easily understood and implemented. For these reasons, the core object data model (CODM) has been selected as the common data model which is considered to be very rich and relatively simple.

Figure 5.1. Remote-Exchange Architecture From [Ref.5].

CODM is a generic functional data model that supports the usual object-based constructs [Ref.56]. The model contains the basic features common to most semantic and object-oriented models. CODM supports complex objects (aggregation), type membership (classification), subtype to supertype relationships (generalization), inheritance of stored functions (attributes) from supertype to subtypes, and user-definable functions (methods) [Ref.56].

Some advantages of using an object-based common model include its ability to condense the utility of shared objects [Ref.57], its capacity for expansion, [Ref.58], and object uniformity. [Ref.59]. Object uniformity is especially important for the unification phase, in which questions about the equivalence of actual data values, types, and operations can be posed. [Ref.59]. It should be noted that using an object-based common data model does not rule out the use of components from other data models, (e.g. relational data model). As stated earlier, translation functions ensure the participation and from the perspective of semantic integration, we do not consider it a serious problem.

b.   *Remote Sharing Language (RSL)*

The remote sharing language (RSL) is part of CODM. and provides a standardized interface to the conceptual schemas of the participating components. RSL provides the capabilities to (1) pose queries to the metadata of selected databases in order to obtain structural information about type objects, (resolution) [Ref.56] and (2) enhance selected database schemas with remote objects, (unification) [Ref.56]. Table 6.1 [Ref.56] presents a preliminary list of RSL commands.

119

RSL Command	Description
ShowMeta(d:DB)	Returns a list of all sharable types in database d
HasProperties(d:DB, t:Type)	Returns a list of all functions defined on type t in d
HasInstances(d:DB, t:Type)	Returns a list of all instances of user-defined type t in d
HasValue(d:DB, i:Inst, f: Func)	Returns the value of function f on instance i in d
HasValueType(d:DB, f: Func)	Returns the value type for a function f in d
HasDirectSubtype(d:DB, t:Type)	Returns a list of all direct subtypes of type t in d
HasDirectSupertype(d:DB, t:Type)	Returns a list of all direct supertypes of type t in d
ImportMeta(d1:DB,t1:Type, d2:DB, t2:Type)	Integrates $t1$ from $d1$ into $d2$ at position $t2$ using relationship r
ImportInstances (d1:DB,t1:Type, d2:DB, t2:Type)	Copies all instances of type $t1$ into a local $t2$

Table 6.1. RSL Commands From [Ref.56].

c. *Local Lexicon*

A foundation of *remote-exchange* and other similar methodologies is the ability to maintain semantic information about all the sharable objects in each component beyond the information that is already provided by the underlying schema. For this purpose, a local lexicon augments each component database system where it defines all type objects which it is willing to share with the other components. "The common vocabulary in which shared knowledge is in a lexicon draws some ideas from declarative knowledge representation forms such as the Knowledge Representation Language (KRL), semantic networks, and the Cyc knowledge base" [Ref.56]. In the remote-exchange approach, knowledge is represented in a local lexicon as a static collection of facts by the following simple form:

<term> relationship-descriptor <term>

The term on the left side represents the unknown concept that is described by the term on the right side. The set of descriptors can be extended and specify the relationships that exist between the two terms. Table 6.2 [Ref.56] indicates a preliminary list of conceptual relationship descriptors used in the Remote-Exchange mechanism.

Relationship Descriptor	Meaning
Identical	Two types are the same
Equal	Two types are equivalent
Compatible	Two types are transformable
KindOf	Specialization of a type
Assoc	Positive association between two types
CollectionOf	Collection of related types
InstanceOf	Instance of a type
Common	Common characteristic of a collection
Feature	Descriptive feature of a type
Has	Property belonging to all instances of a type

Table 6.2. Relationship Descriptors From [Ref.56].

The list of commonly understood terms is called an ontology, and as stated in previous chapters, ontology varies from one application domain to another application domain. For example, an ontology for a multidatabase system consisting of cooperating branches of the Navy is different from the ontology used by collaborating medical societies. An "ontology package" that consists of a general–purpose ontology (GPO) is put into a multidatabase system initially. Later, several special purpose ontologies (SPOs) can be added according to need of application domain. The GPO is application independent, and comprises a minimal working set (OF WHAT?) for each multidatabase system. [Ref.56].

SPOs are application dependent and contain terms that cover specific topics. Both the GPO and SPOs are extremely dynamic, [Ref.56], meaning that their numbers of terms increase depending on vocabulary use by the participating components. The complete package is stored in the semantic dictionary. (We will present more on the ontologies in the next section when discussing semantic dictionaries.)

Because interoperability has meaning among components that model similar or related information, "it is reasonable to expect a common understanding of a minimal set of concepts taken from the application domain among all the participants" [Ref.56]. If we give an example from a military context, an ontology package could be as follows:

GPO = {Component, Name, Number, Item, Thing, Person, Vehicle, ... }

SPO (Military) = {War, Battle, Weapon, Ship, Cannon, Rocket, Missile, On duty...}

SPO (Navy) = {Submarine, Destroyer, Frigate, Hull, SAM, Torpedo, Deployment, Patrol, CO, XO, ...}

The *GPO* includes all those terms common to all English-speaking components. *SPO (Military)* is a special purpose ontology containing terms taken from the general military environment. *SPO(Navy)* contains terms used specifically in the Navy. As an example, consider a database about the Navy including the attributes, "ShipName, TorpedoType, PatroledBy, TaskGroup, Corvette" under different tables. A lexicon belonging to this database would be similar to the one in Table 6.3.

Term	Relationship Descriptor	Key Concept
ShipInternationalName	Equal	Name
TorpedoType	Identical	Torpedo
PatroledBy	Kind of	On Duty
TaskGroup	CollectionOf	Ship
Corvette	Compatible	Frigate
Corvette	Compatible	Destroyer

Table 6.3. Local Lexicon

The underlying principle of the lexicon is to define and introduce new terms so that they can be understood by other components in the collection. Thus, local lexicons portray the real-world meaning of sharable objects so that the structural representation complements the definitions given in the conceptual schema. [Ref.56].

d.   Semantic Dictionary

While local lexicons contain semantic descriptions about local, sharable data, they do not contain any knowledge about relationships among descriptions in different, independent lexicons. This information is collected in a global repository called a semantic dictionary. Like the local lexicon, the semantic dictionary is dynamic, meaning that its content is updated whenever new or additional information is made available, (e.g., after a relationship between two similar remote types has been established) [Ref.56].

124

Related types from different components that have been identified are grouped into a collection called a "concept". Furthermore, subconcepts can be identified within different subcollections, and this generates a concept hierarchy. In a sense, the semantic dictionary "represents a dynamic federated knowledge base about the different relationships existing among the sharable objects in a federation."[Ref.56].

In addition to the concept hierarchy of sharable objects, the semantic dictionary also contains the ontology package described earlier, as well as a list of relationship descriptors. These descriptors are used in local lexicons to describe relationships between unknown objects and terms from the ontologies. It merits consideration that the ontology package is in some ways collection of local ontologies. Only general-purpose ontology is not repeated in the semantic dictionary since it is common to all components. If we consider a MDBS used by all branches of the military, a semantic dictionary for this collaboration would be similar to the one in Table 6.4.

SPO	SPO Navy	SPO Army	SPO Air Force
Military	SPO	Training	
GPO			

Table 6.4. Semantic Dictionary.

As we said, the GPO in the package is static, but the SPOs are dynamic and can be updated over the lifetime of the federation. The GPO and SPOs taken

125

together provide the components with a vocabulary that describe the application areas of the database systems involved. As a result, the resolution mechanism in Remote-Exchange works best if all the components have a similar background, thus reducing the need for many additional SPOs, or SPOs with many terms. If we extend our sample MDBS to include other organizations such as some departments in the government, we should add their respective ontologies which means increasing the burden on the semantic dictionary.

3. Resolving Representational Heterogeneity in Remote-Exchange

The main reason for the relatively slow progress in the area of resolution is that heterogeneous databases provide a wide range of vocabulary and name usage that is inherently difficult for computers to "understand." The narrower the domain (i.e., the higher the degree of commonality in the vocabularies and the closer the relationships among objects), the higher the chances of successful resolution. Currently, the process of resolution of representational diversity is still only partially automated and requires human interaction at many points.

To determine the relationship between objects within a broader context, instead of a single method (e.g., schema resolution based on structural knowledge-see [Ref.60]), a combination of several different approaches taken together is offered in [Ref.52]. In [Ref.52], it is asserted that structural information alone will not be sufficient to resolve inter-object relationships. So, the authors have augmented the conceptual schemas of participating components with additional semantic information describing the usage of sharable objects in their application environment.

We now describe the remote-exchange approach for resolving semantic heterogeneity, namely, determining relative object equivalence. The meanings of concepts unknown to other components are determined using the RSL, in addition to a local dictionary or lexicon, as well as the semantic dictionary. [Ref.56]. RSL commands return structural information about an object (e.g. supertype, subtype(s), properties, etc.).

The local lexicon, which is created and updated by each component separately, contains the semantic description of every sharable type object in the database. In order to make the local lexicons usable through the entire federation, a common knowledge representation is used. Unknown terminology can be located and compared with known terminology. [Ref.56]. The semantic dictionary describes the relationships between terms in the local lexicons. Figure 5.2 [Ref.56] shows the interactions among these components.

As mentioned in previous chapters (in the architecture of federated databases), for components to protect their private data from the rest of the federation, sharable objects must be placed in a special section of the conceptual schema, namely *export schema*. Everything that is not part of the export schema is invisible to the rest of the federation.

Before discussing the details of the remote-exchange approach to resolve representational heterogeneity, we review the various relationships that can exist among object that model the real-world concepts in different participating components.

a. Common Concepts

As a result of different causes for schema diversity mentioned earlier, the concepts of an application domain may be modeled by different representations, say R1 and R2. If we remember our "ship" example, the ship object may be represented by its international name (Int-Name) in one data source while being represented by its hull number (HullNo). In addition to the obvious naming differences, both objects mirror closely related real-world information, but use different type constructs (string vs. umber) in their representations. Several types of semantic relationship can exist between two representations R1 and R2. They may be *identical, equivalent, compatible* or *incompatible.*

Figure 5.2. Representational Heterogeneity Resolving Mechanism From [Ref.56].

1.   *Identical:* R1 and R2 are the same. This situation occurs when the same modeling constructs and perceptions are used and applied, and when no extraneous information enters into the specification [Ref.55]. For example, in our sample lexicon table, "TorpedoType" and "Torpedo" are identical because they both represent the same real-world objects and type structures are the same (both use strings to model the information). This type of relationship between two representations R1 and R2 is expressed using the *Identical* relationship descriptor.

2.   *Equivalent:* R1 and R2 are not the same because different modeling constructs have been applied. For example, even though "HullNo" and "IntName" both represent the real-world object "ship", one uses alphanumeric characters and the other uses a domain of four letter strings. This kind of relationship between two representations R1 and R2 is expressed using the "Equal" relationship descriptor.

3.   *Compatible:* R1 and R2 are neither identical nor equivalent. However, their representation is not contradictory. For example, a corvette and destroyer actually represent different real-world objects, however there is a lower upper bound for them that both can be transformable to that bound. A "warship" or "ship" object may be *lub*. The relationship between these two representations R1 and R2 is expressed using the *Compatible* relationship descriptor.

4.   *Incompatible:* R1 and R2 are contradictory to each other because of the fundamental differences in the underlying information. They have neither a common context nor a common domain. As far as we see, in remote-exchange approach, all relationships except the ones defined by a relationship descriptor are accepted incompatible by default. However, we believe that there must have been a descriptor to indicate incompatible relationship. To conclude that two objects are incompatible, the lexicon must be scanned end-to-end, and this brings an extra burden to the system. At least a cache-like mechanism could hold this information and the lexicon would not be scanned completely every time when resolution is required.

b.   *Related Concepts*

In addition to common concepts, we can list the following most commonly used types of interschema (binary) relationships:

1.   *Generalization/specialization:* Generalization is achieved when two or more types are untied producing a higher level type. [Ref.52]. In our example, "warship" is the generalization of "destroyer" and "corvette". Specialization is the opposite of generalization. The relationship descriptor that represents generalization/specialization is called KindOf. We can say that specialization from a generic object produces compatible objects.

2.   *Positive association:* It is impossible to accurately classify every type of relationship capable of existing between objects. [Ref.56]. This category

130

includes concepts that are "synonyms" in some context, for example, "ship" and "boat". The relationship descriptor that represents positive associations is called Assoc.

c.   *Strategy for Resolving Object Relationships*

The basic function of the resolution mechanism is to return the relationship that exists between two given objects, a local and foreign one. Specifically, the resolution strategy of remote-exchange is based on (1) structural knowledge or the conceptual schema, and (2) the known relationships existing between keywords and the two objects in question. [Ref.56]. One characteristic of the approach is that the majority of user input is introduced before (as opposed to during,) the resolution step (i.e., when selecting the set of keywords and creating the local lexicon). [Ref.56]. So we can say that remote-exchange is highly static.

Now, let us refer to our Navy example. Assume we have two separate data sources, one of them keeps information about the destroyers (schema A) and the other keeps information about the submarines (schema B). In Figure 5.3, we gave a pictorial description of parts of two local lexicons belonging to Schema A and Schema B, respectively. In the same picture, we also gave a subpart of the concept hierarchy in the semantic dictionary.

Looking at A's lexicon, we can see that the property *Commanded-By* is equivalent to concept *"Commanding Officer"* in the ontology packet. Similarly, in Schema B, *"Captain"* and *"Name"* combination is equivalent to the concept, *"Commanding Officer."* Therefore, *"Commanded-By"* and the type property combination of *"Captain"* and *"Name"* must be equivalent.

Figure 5.3. Resolution of Representational Heterogeneity Between.

With respect to the resolution of representational heterogeneity, all information about the structure of a type object is provided through selected RSL

commands--an approach perceived as a variation to the typical model of behavior encapsulation in object-oriented programming languages. [Ref.56], Instead of encapsulating behavior, these commands encapsulate the structure of a type object. The advantage of this approach is that RSL commands are basically computed (or foreign functions) in the data model of remote-exchange architecture, and can be part of each component's schema without modifications to the underlying schema.

4.   Sharing

The Remote-Exchange architecture provides three services to its multidatabase components: an intelligent sharing advisor, resolution of representational heterogeneity, and sharing [Ref.52].   When a new component initially participate the federation, that component must first register by initiating the sharing advisor. [Ref.56].   The sharing advisor enters the data that the component is willing to share into the semantic dictionary so that it can be seen and used by other components.   After the component registers with the sharing advisor and defines its sharable type objects using the common terminology in the semantic dictionary, it is then ready to information exchange. "Sharing takes place on a component pair-wise basis when the sharing advisor has located the sources of relevant information.

The importing component selects those relevant foreign objects that it wants to integrate into its local framework. [Ref.56]   Given a foreign object, a related local object, and the relationship between the two, the sharing tool places the foreign object (including its instances and stored functions) into the appropriate place in the local

metadata framework (type hierarchy). At this point, the unification is complete, and the newly imported object can be used by the remote component.

B.   SCHEMA INTEGRATION

Even though we mentioned the approaches to database integration many times in previous chapters, we briefly summarize them here:

Two popular approaches to heterogeneous database integration are the global schema approach and the federated schema approach. In the global schema approach, schemas corresponding to each local database are combined into a single integrated schema. In the federated approach, each local database provides an export schema (a portion of its schema that it is willing to share with other databases).

The schema integration process, which means generating one or more integrated schemas from existing schemas, is the core of these approaches to provide heterogeneous database interoperability. These schemas represent the semantics of the component information sources that are being integrated, and are used as inputs to the integration process while the output of the process is one or more integrated schemas representing the semantics of the underlying databases. The output schemas are represented using a common data model, and are used to formulate queries that might need to span multiple databases. The schemas hide any heterogeneities that are result from (1) schematic differences in the underlying databases, or (2) differences in the data models which they are based upon. [Ref.52].

The term *schema integration* is used to refer to methodologies that facilitate the integration of schemas as defined above, and is typically used in the design of a new database schema. In addition, it is used to refer to view integration. However, the two processes differ in important ways:

- View integration is the process of generating a single integrated schema achieved by generating the integrated schema corresponding to the multiple user views, and then designing the database corresponding to that schema. [Ref.62]. So, we can say that *view integration* is used in top-down database design, whereas *schema integration* is a bottom-up process because it attempts to integrate existing databases.

- In view integration, users define views by using a unique single data model. But, in schema integration, since the underlying databases can be heterogeneous, the schemas to be integrated may be represented using multiple data models [Ref.62].

- User views do not reflect existing data in a database. However, in schema integration, the integrated schemas represent existing databases. We believe that this is an important distinction, because the schema generated after the schema integration process cannot violate the semantics of the existing databases. However, in view integration,

135

because the views don't represent the real objects, there may be more flexibility in the interpretation of their semantics.

Schema integration is a complicated and time-consuming task, primarily because most schematic representations cannot completely capture a database's intended semantics. [Ref.52]. Hence, the process of schema integration requires extensive interaction with database designers and administrators to understand the exact semantics of the databases. Otherwise, the semantics of the integrated schema may violate the semantics of the underlying databases. In [Ref.63], it is asserted that the necessity of this extensive interaction with the database designers and the administrators makes fully automated schema integration process impossible. We agree with this idea, because even a very developed expert system can not represent the ideas of a large group of peoples who design and administer information sources. Knowledge engineers, who interview the real experts in their own domains, design expert systems.

As we said in Chapter III, representing knowledge and establishing the knowledge base of an expert system is not a trivial matter. Since it is impossible to interview with all the experts in one domain of interest, all the knowledge can not be represented. In the schema integration process, this knowledge corresponds to the way of design of database designers, more precisely their intentions. This difficulty is evident when considering the natural language spoken by databases.

Database designers and administrators would like to use words, phrases, and abbreviations they use in their daily life. Even we assume, for instance a very rich thesaurus is used to cover all words, phrases etc. used in a natural language, this

thesaurus would be error free and cannot represent the possible errors that can be made by careless designers and administrators. However, as we will discuss in later sections in this chapter, tools can be developed that can reduce the amount of human interaction.

It should also be noted that schema integration is not a one-shot process. Since the integrated schema represents the real data (from underlying databases), changes to it may be needed for these reasons:

- Changes in the database structure that result in changes to the underlying schemas,

- Changes in the constraints specified on the underlying databases, and

- Changes to data values due to additions, modifications, or deletions in the underlying databases. [Ref.52]

As a result, we can say that a desirable property of any schema integration approach is that it should also be able to dynamically handle changes to the underlying databases.

In the rest of this section, we present a framework for schema integration, various steps in schema integration. We also discuss how the schema integration process can be automated and present several automated software tools.

137

1.   Schema Integration Framework

First we outline the steps of a typical methodology for schema integration, and then we present a classification of the various integration strategies that have been used.

a.   *Steps in Schema Integration*

A typical schema integration methodology can be divided into four phases [Ref.64]. The steps, shown in Figure 5.4, are as follows:

Figure 5.4. Steps in Schema Integration Methodology.

138

1. *Schema translation:* In this phase, schemas that correspond to the individual databases are translated into schemas using a common model. Generally, a semantic model (such as entity-relationship model) is used for this purpose.

Figure 5.5 [Ref.52] shows an example of a relational and a network database. The schemas are examples of databases that may exist in a bank such as Navy Federal Credit Union or Monterey Credit Union. The first database contains information about customers and their accounts in the bank. The second database keeps track of loans issued to borrowers. The translation of these schemas may be performed manually or with the aid of a translation tool. However, even with the use of a translation tool, it is likely that some manual interaction with the tool is needed. Any schema translation technique should have the following characteristics [Ref.64]:

a) The schema represented by the common model should include the complete semantics of the underlying database, and

b) A command on the translated schema must be translated into commands on the original schema. Figure 5.6 [Ref.64] shows the translated ER representations of the schemas in Figure 5.5.

```
(a)

CUSTOMER (SSNo, FName, LName, Address, PhNo, NYears, CR)

ACCOUNTS (AcctNo, Balance, Type, CustNo)


(b)

| IDNo | First | Last | Address | PhNo | CR | NYears | NLoans |
|------|-------|------|---------|------|-----|--------|--------|
|      |       |      |         |      |     |        |        |
```

Figure 5.5. Local Schemas (a) database 1 (relational) and (b) database 2 (network).

Figure 5.6. ER Representation of Translated Schemas (a) database 1   (b) database 2

140

2. *Schematic interschema relationship generation*: "The goals of this phase are to (1) identify objects that are potentially related in the underlying schemas (i.e., entities, attributes, and relationships), and (2) categorize the relationships among the related objects. [Ref.64]. This is done by examining the semantics of the objects in components and identifying relationships among them based on their semantics. The semantics of an object can be determined by analyzing schematic properties of entity classes, attributes, and relationships in the schema, but these properties can give only some partial semantics. For example, "integrity constraints, cardinality, and domains are properties of attributes that convey their partial semantics" [Ref.64].

To obtain the exact semantics, as stated before, interaction with the designers and administrators is required to exploit their knowledge and understanding of the application domain. The main goal of this step is to create a reliable set of relationships among database objects. [Ref.64]. The accuracy of the characterizations of these relationships is very important, because they are used as input to the integrated schema generation phase (the phase after this phase). For example, in Figure 5.6 [Ref.64], we would identify the two entities CUSTOMER in part (a) and BORROWER in part (b) as related to each other. In addition, we can classify the relationship as being a sub-sumption relationship [Ref.64]; (i.e., the set of borrowers in the second database is a subset of the set of customers identified in the first database.) Finally, we should identify attributes in the two entity classes that may be related.

3.   *Integrated schema generation:*   In this phase, the interschema relationships generated previously in the second phase are used to generate an integrated representation of the underlying schemas—a task that involves resolving different forms of heterogeneity that might exist between related objects. [Ref.64]. In their paper, Sheth and Kashyap [Ref.65] classify these heterogeneities into five major categories: domain definition, entity definition, data value, abstraction level, and schematic incompatibilities.

The integrated schema generation process resolves these forms of heterogeneities and generates an integrated schema that appears homogeneous to the user.   In our example, in the integrated schema in Figure 5.7, a sub-sumption relationship between CUSTOMER and BORROWER is generated to reflect the nature of the relationship among these entity classes.   Note that the attributes SSNo and IDNo have been integrated into a single attribute (CustNo) in the superclass. This type of integration assumes that these attributes have been identified as being equivalent to each other in the interschema relationship generation step.

Figure 5.7. Integrated Schema.

4.   *Schema mapping generation*: This step accompanies the integrated schema generation step and entails the storing of information about mappings between objects in the transformed (or integrated) schemas and the local schemas. [Ref.64].   Such mappings are important for query transformation.   For example, we would need to note that the attribute CustNo in the integrated schema (Figure 5.7) maps back to SSNo in Database 1 and IDNo in Database 2.

Figure 5.7. Integrated Schema.

b.   *Classification of Schema Integration Strategies*

Elmagarmid and Sheth [Ref.52] classify the primary properties that distinguish integration strategies in the literature as (1) the abstraction level at which integration is attempted, and (2) the semantics conveyed by the input schemas.

(1)   Classification Based on Abstraction Level: Integration methodologies presented so far in the literature can be classified as operating at one of three levels: the user view level, conceptual schema level, or data level.

Among these, the most common level is the user views level. The objective of view integration methodologies is to integrate several user schemas

144

(representing users' views of a database) into a single integrated schema [Ref.61]. Generally, users' views are represented by using a common data model. As a result, these methodologies will not likely require the schema translation step. [Ref.61]. Furthermore, most of the intended semantics are represented by the schema itself since the views do not represent an existing database. [Ref.52].

User views can be created dynamically as well as statically, but for this a multidatabase language is required. (While the multidatabase languages are not mentioned in this chapter, a brief informational summary can be found in Chapter II.) The integrated schema (from the views of users) could then be used as the stepping off point for designing a new database. Examples of view integration methodologies can be found in Batini et al. [Ref.55], Navathe and Gadgil [Ref.66], Batini and Lenzerini [Ref.67], Biskup and Convent [Ref.68], Navathe et al. [Ref.69], Shoval and Zohn [Ref.70], and Gotthard et al. [Ref.71].

Methodologies that operate at the conceptual *schema* level generate one or more integrated schemas from the schemas in the local databases. [Ref.52]. To achieve this objective, the methodologies must be able to manage the existence of both structural and semantic heterogeneity in the underlying databases." [Ref.64].

Methodologies at this level can be divided into two classes [Ref.64]:

a) *Schema restructuring methodologies:* Those methodologies that generate integrated schemas by applying schema restructuring operators to the underlying databases. Some examples of schema restructuring

145

methodologies include El-Masri et al. [Ref.72], Larson et al. [Ref.73], and Spaccapietra et al. [Ref.74].

b)   *View   generation   methodologies:*   Those methodologies that generate an integrated representation by developing views or defining queries on the local databases of interest. Examples of approaches using the view generation methodology can be found in Kaul et al. [Ref.75], Ahmed et al. [Ref.76], Bertino [Ref.77], and Kim and Seo [Ref.78].

Application of any of these methodologies to the schemas in Figure 5.6 (representing local schemas after translation) would result in an integrated schema similar to that shown in Figure 5.7. Thus, the primary difference between schema restructuring strategies and view integration strategies is the fact that "in schema restructuring methodologies, the schemas being integrated are derived from heterogeneous data models and represent one or more underlying databases" [Ref.52].

In our example, if we relax the restriction that all borrowers have an account with the bank, we would follow two steps to generate an entity class representing the set of all customers who are associated with the bank. First, we define a query on the CUSTOMER and BORROWER entity classes from the underlying databases, and second, create a supertype entity class called ALL-CUSTOMER. Figure 5.8 [Ref.64] shows the view and the query that can result in such a view being generated.

Figure 5.8. View Generation Using a Query.

It should be noted that if the SSNo and IDNo fields in the CUSTOMER and BORROWER entity classes are different, we may have to define a new attribute called CustNo and define a function that maps CustNo to IDNO and SSNO.

The primary difference lies in the static nature of schema restructuring and dynamic nature of view generation.[Ref.52]. An integrated schema generated using schema restructuring is a representation that reflects schema definitions at the time integration was performed. Therefore, any changes made to the underlying databases affecting the schemas will require that the integration process be repeated. [Ref.52]. The view generation approach is more dynamic because the integrated representation is generated by defining a view on the local schemas. As a result, if the

schemas change in such a way that the existing view is affected, the only thing which must be redefined, is the new view. [Ref.52]. For example, if the bank wanted to merge information about its money market account customers (maintained separately) with the rest of the databases, we would define a new view to include customers with regular, loan, and money market accounts, as shown in Figure 5.9.

Figure 5.9. New View Generation Using a Query.

The third class of integration methodologies operates at the *data level*. This class of methodologies relies on actual data values to achieve integration. [Ref.52]. Much of the work at this level has focused on integrating relational databases. We can find instance-level integration strategies in DeMichiel [Ref.79], Chatterjee and

148

Segev [Ref.80], and Prabhakar et al. [Ref.81] fall into this category. Data-level methodologies address two main problems [Ref.52]:

- *Entity identification*: How does one identify representations of the same real-world entity in different databases?

- *Attribute-value conflicts:* How does one deal with differences in data values among attributes that represent the same real-world entity? These differences may be caused by differing attribute domains as well as differences in the actual data values stored in the databases. [Ref.52].

For example, let us assume we are trying to generate a relation that represents the list of customers with outstanding loans. This list could be generated by defining an intersection of the two databases which are shown in Figure 5.10 [Ref.52]. The process of intersection is relatively easy if the two relations share a common key. Generating an integrated relation requires that we identify, for instance, that two tuples represent the same person.

However, consider the tuples shown in Figure 5.10. It is clear that tuple 1 in the CUSTOMER relation and tuple 2 in the BORROWER relation refer to the same entity. But because the SSNo and IDNo do not match, they cannot be used as the only source of identification for matching tuples. The combination of last and first names also cannot be used because more than one customer will have the same combination. Thus, "differences in data and the lack of a key can make identifying related tuples and performing a join difficult" [Ref.52].

Instance-level integration deals with resolving such incompatibilities. It should be noted that "changes to the data values may void any

149

integration performed previously" [Ref.52]. Thus, instance-level integration strategies are dynamic in nature.

SSNo	FName	LName	Address	PhNo
123123123	John	Doe	111 Brownell	5553333
213222111	Bob	Smith	21 Broadway	5552323
565656566	John	Doe	13 Moran	5559393

IDNo	First	Last	Address	PhNo
5008	John	Doe	13 Moran	9393
6005	John	Doe	111 Brownell	3333
1010	Jane	Smith	233 Leahy	3535
5009	Bob	Smith	21 Broadway	2323

Figure 5.10. Instance Level Integration Problem Between Two Relations.

Another technique that falls into the data-level category deals with semantic integrity constraints and their use in schema integration. Ramesh and Ram [Ref.82] presents a methodology that describes how integrity constraints from multiple databases can be combined to develop constraints at the global/federated schema level, and the use of these integrated constraints in semantic query processing.

(2)   Classification Based on Data Model of Input Schemas: Highly dependent on the semantics conveyed by the local schemas, strategies for schema integration are directly related to the type of data model used to represent the

150

local schemas. Four models have been used to represent the local schemas: relational models, semantic models, object-oriented models, and logic-based models.

a) *Approaches based relational models*: The earliest schema integration methodologies used relational models to represent the local schemas. The drawback of using a relational model is its limited power to express natural language, which results in an inadequate representation of semantics by the schemas. [Ref.52]. However, the commonality of relational databases and the simplicity of the relational model made the relational model and relational databases ideal starting points for new research. The examples of the prototypes and methodologies are described in Deen et al. [Ref.82], Templeton et al. [Ref.84], Chung [Ref.85], and Kim and Seo [Ref.78].

b) *Approaches based on semantic models:* Approaches based on semantic models use variants of the entity-relationship model to represent local schemas and the integrated schemas. Larson et al. [Ref.86], Shoval and Zohn [Ref.70], Spaccapietra et al. [Ref.74], and Sheth et al. [Ref.87] are examples of methodologies that belong to this category. The primary reason for using semantic models is that they, as compared to the relational models, can more richly express semantics. [Ref.52]. Since semantic models are most commonly used to represent views and conceptual schemas, most of these methodologies fall into the user view/conceptual schema-level categories of the previous classification.

c)   *Approaches based on object-oriented models*: Some of the methodologies in this category attempt to integrate methods along with schemas. They also deal with integration of complex attributes and object hierarchies. Most of these methodologies fall into the view integration/conceptual schema integration category presented above. Examples of methodologies in this category can be found in Bertino [Ref.77], Czejdo and Taylor [Ref.88], Kaul et al. [Ref.75], Geller et al. [Ref.89], Gotthard et al. [Ref.71], and Thieme and Siebes [Ref.90].

d)   *Approaches based on logic:* Among all these categories, the latest are the logic-based approaches, representing a natural step in the development of schema integration methodologies. First-order logic is shown to be capable of formally representing the semantics of relational databases. [Ref.52]. Using a logic-based approach also provides the capability of capturing more semantics than is possible using semantic models. For example, logic-based models enable us to express semantic integrity constraints, explicit user-defined integrity constraints deemed useful in query transformations. [Ref.91].

Ramesh and Ram [Ref.92] describe how semantic integrity constraints can be used to facilitate schema integration. Whang et al. [Ref.93] also note that it is easier to translate relational schemas to logic-based schemas than to semantic models.

2. Techniques for Interschema Relationship Identification (IRI) and Integrated Schema Generation (IGS)

In this section, we summarize the approaches to the interschema relationship identification (IRI), integrated schema generation (ISG), and schema mapping generation steps of the schema integration process.

We classify IRI techniques based on the abstraction level that defines the nature of the semantic knowledge available with regard to the databases and applications. [Ref.52]. Contrarily, the discussion on the ISG techniques center around the data model used, because the characteristics of the data model, its semantics, and the heterogeneity in the representation of the underlying databases are the primary factors affecting the ISC process. So, we focus on the methodologies belonging to the conceptual schema and data-based approaches.

Since schema mapping generation is relevant primarily in the context of approaches based on conceptual models, discussion on schema mapping generation focuses more closely on the differences in schema restructuring and view generation approaches.

a. Interschema Relationship Identification (IRI)

The objective of the IRI phase is to identify objects in the underlying schemas that may be related and to classify the relationships among them.

(1) Approaches Based on Conceptual Schemas: IRI techniques based on conceptual schemas use a two-phase process consisting of (a)

153

identifying objects that are related and, (b) classifying the relationships among these objects.

The first phase implies the extraction of the intended semantics of the objects in databases, and the identification of objects that are semantically related. [Ref.52]. After a potential set of related objects are identified, the second phase activates to classify these relationships into various categories. As we said before, the inability of existing data models to represent the true semantics of the real-world objects in databases causes this phase to require extensive interaction with designers or administrators of the database specific to application domain.

Approaches based on conceptual schemas use the knowledge represented in the schematic constructs to determine the relationships between objects. [Ref.52]. Primary schematic constructs are entity classes, attributes, and relationships. In Larson's paper [Ref.73], the various properties (uniqueness property, lower and upper cardinality constraints, domain, static and dynamic integrity constraints, security constraints etc.) of an attribute are used to establish relationships among attributes of two different entities from different databases. According to Larson [Ref.73], when the attributes of different entities are compared on these properties, they provide definitions for assessing the degree of equivalence of the attributes.

For example, entity classes could be compared on the basis of their names and the description of their roles in the database. Relationships could be compared on their names, cardinality, and the similarity of other participating entity classes. Thesaurus mechanisms and highly developed dictionaries could be used to

support the comparison of names, roles, etc. [Ref.73]. The process itself can be done manually, or it can be partially automated. (Tool kits that perform automated interschema relationship identification are introduced in subsequent sections.)

The aim of analyzing these schemas is to identify semantically related objects. However, these schemas must not only identify but also *classify* the relationships among related objects. [Ref.52]. The classification generated is dependent on the methodology used. Although different methodologies generate different classification, here we only show Larson's [Ref.73].

Larson [Ref.73] generates four types of equivalences between attributes. These are:

a EQUAL b,

a CONTAINS b,

a CONTAINED-IN b, and

a OVERLAP b.

He defines five types of relationships among entities and relationships:

A EQUAL B,

A CONTAINS B,

A CONTAINED-IN B,

A OVERLAP B, and

A DISJOINT B.

Users are asked to indicate one of these types of relationships for every entity/relationship that may have equivalent attribute relationships. [Ref.73].

(2)   Approaches Based on Data: The objective of most IRI techniques using the data-based approach is to distinguish instances of entity classes referring to the same real-world entity in different databases. [Ref.52]. The simplest approach assumes that relations from different databases have a common key. So it can be said that, the tuples that have a common key representing the real-world entity. In cases where there is no common key available, different probabilistic theories have been adopted. For example, some techniques try to evaluate the probability that two tuples refer to the same real-world entity only by comparing the key attributes, or by comparing all attribute values.

b.   *Integrated Schema Generation (ISG)*

Since the ISG phase is mainly affected by the data model used, we examine the approaches based on data models, namely semantic model approaches, object-oriented model approaches, and logic-based approaches.

(1)   Semantic Model Approaches: As previously mentioned, there are mainly two schema integrating methodologies: (1) schema reconstructing methodologies and (2) view generation methodologies. Since data language capabilities are the primary concern in view generation methodologies, we focus on ISG techniques used in schema restructuring methodologies.

The main problem is to generate an integrated representation of independent schemas that reflect the semantics of the underlying databases. The primary technique adopted for achieving the integrated representation is to create generalization-specialization relationships in the integrated schema. [Ref.52]. The schema in Figure 5.7 is an example of this integration.

Larson [Ref.73] presents an approach to schema integration that is based on the idea that any pair of objects can be integrated if their identifying attributes can be integrated. As we discussed in the previous section, Larson defines four types of relationships between attributes (i.e. EQUAL, CONTAINS, CONTAINED-IN, and OVERLAP). Entity class and relationship equivalence are then defined in terms of relationships between identifier attributes. Rules for integrating entity classes and relationships belonging to each category are presented along with rules for integrating attributes. Also, he presents general guidelines for transforming related objects into objects in the integrated schema. Since an object is typically linked with other objects in the schema, in order to generate a correct integrated schema, the transformation may require that changes be made to the links between objects. [Ref.73].

The rules for integration do not handle the naming conflicts. They must be handled differently. Naming conflicts present themselves in two forms: First, when two unrelated objects share the same name, one of the objects needs to be renamed. And second, equivalent objects have different names; in this case, a decision must be made as to which name should be used in the integrated schema.

Structural conflicts arise when either (a) two related objects are defined using different data model constructs or (b) two related objects use the same

157

construct with different properties. [Ref.52]. For example, in Figure 5.7, IDNo and SSNo represent related attributes, but they may have different properties.

Most methodologies for schema integration address the structural conflict problem. Spaccapietra [Ref.74] presents a methodology for the integration of any two types of objects. He views a schema as a graph with edges and nodes. Relationships between objects in the schema are specified using correspondence assertions (see Chapter V). A methodology for resolving structural conflicts is also presented in Bouzeghoub and Comyn-Wattiau's work [Ref.94]. They deal with another problem, integrating differing constraints, such as cardinality constraints and key and functional dependencies, during schema integration.

(2) *Object-Oriented Approaches:* Object-oriented approaches deal with all the issues relevant to approaches based on semantic models as well as some additional concerns. The general trend is to use object-oriented schemas as the input schemas to a view integration algorithm. Other issues dealt with by object-oriented techniques are summarized in two [Ref.52]:

• All OO approaches develop mechanisms for integrating class hierarchies that represent a set of classes exisiting in generalization/specialization relationships [Ref.90], (such as in our example *Borrower IS A Customer*). Recursive hierarchies (an attribute in one of the classes has another entity class as its domain) may further complicate such a class hierarchy. [Ref.52]).

158

Thieme and Siebes [Ref.90] present techniques for integrating class hierarchies on the basis of semantic and structural equivalence of the classes in the hierarchy.

• The methodologies have to deal with the integration of methods. Two resulting tasks are (1) new methods may need to be defined for the integrated view and (2) preexisting methods in the entity classes being integrated may need to be integrated.

Because existing methods may differ in name and parameters, techniques for resolving these differences need to be developed.

(3) *Logic-Based Approaches:* Work on using logic-based approaches for schema integration is still in its early stages. Logic-based approaches are desirable because of the way semantics can be conveyed using a logic-based representation. [Ref.52]. Also, it should be noted that the use of expert systems in semantic heterogeneity resolution rose in popularity after the advantages of logic-based representation became apparent. In subsequent paragraphs, we examine the first viable logic-based approaches in the literature.

Whang et al. [Ref.95] describe a rule-based approach to schema integration. Each of the local schemas being integrated is represented as a schema using first-order logic. These databases constitute the extensional databases (EDBs). The integrated schema is then established according to a set of first-order logic rules applicable to the EDBS, i.e., the integrated schema is a meaningful set of database (IDB) relations. [Ref.95].

Johannesson [Ref.95] describes the importance of schema transformations in view integration. He addresses the problem of merging semantically equivalent but structurally different concepts. He also asserts that standardizing the schemas to be integrated by applying schema transformations prior to integration yields better results.

c.   Schema Mapping Generation

Schema mapping generation is performed concurrently with bot! the schema translation and integrated schema generation steps of a schema integration methodology. The creation of mappings during schema translation is necessary to pose intelligible queries to the local databases. [Ref.52]. This mapping may be stored as a dictionary at each local database. The mapping generated during the integrated schema generation process maps an object in the integrated schema to objects in the local schemas being integrated. If the schema restructuring approach is applied, this mapping is generated concurrently with the integrated schema. The mapping is then stored in a global directory/dictionary. If the view generation technique is used, then the mapping is generally defined as part of the query statement used in creating the new view. [Ref.52].

3.   Automating Schema Integration

The methodologies presented in the previous sections describe the general principles that can be used to achieve schema integration. It is clear that schema integration is a very complex and time-consuming process, and automation is desirable.

However, as stated before, the schema integration process cannot be completely automated-- substantial interaction with designers and administrators is required during all the phases of the process. This is because schema integration tries to understand the semantics of component databases using knowledge representations that cannot completely capture the intended semantics of the data.

Furthermore, two identical schemas can be integrated differently according to their intended use. [Ref.97]. However, it is possible to automate schema integration to some extent so that tools take over ordinary, routine tasks, thus leaving most of the decision-making to designers and administrators, reducing the amount of user interaction. In this section, we present the integration tools that automate some portions of the schema integration process.

a.   *Schema Integration Toolkits*

One of the first automation efforts published was DeSouza's [Ref.98]. This work focuses on interschema relationship identification (IRI). The author presents an expert system designed to integrate conceptual schemas defined using the Abstract Conceptual Schema (ACS). He defined a set of functions (called *resemblance functions*) that can be used to compare objects in the schemas. These functions use both names and structure to estimate the resemblance between constructs.

Every resemblance function has a weight that is used to indicate the relative level of importance that the user determines. [Ref.98]. For example, if having similar attributes is the most important criterion, the weight associated with that function would be high. Objects whose computed values of similarity fall above a certain threshold are

presented to the user as being possibly similar. The methodology is specific to ACS (Abstract Conceptual Schemas), so its usage is limited. Also, this paper does not deal with the integrated schema generation step.

Sheth [Ref.99] presents a tool that leads users and designers through a five-step schema integration process: Schema Information Collection, Equivalence Class Creation and Deletion (Entities and Categories), Equivalence Class Creation and Deletion (Relationships), User Assertions (Entities and Categories), and User Assertions (Relationships).

In the schema information collection step, the schemas are input into the tool in the form of Entity-Category-Relationship schemas. [Ref.99]. The user is asked to specify relations among attributes of entities and relationships that may be related. After these equivalences are specified, they are used to generate an ordered list of object (entity and relationship) pairs. The order demonstrates the likelihood that an object pair may need to be integrated. [Ref.99]. Users are then required to analyze this ordered list and specify one of five types of relationships between the objects. These relationships are *equal, contained-in, contains, disjoint but integratable,* and *disjoint* and *nonintegratable.*

The toolkit presented in Sheth's paper is actually far from automation and requires a large amount of interaction with users and designers. Users can only make equivalence assertions among attributes, a condition that limits the amount of semantic information that can be encapsulated. [Ref.99].

Following the first introduction of toolkits comes the development of second-generation toolkits. One of them is BERDI [Ref.100], which provides for the

user's ability to define relationships among objects in a potentially related set of entities called *entity clusters*. [Ref.100]. The system allows users to assert three types of relationships among attributes: *equivalence, inclusion*, and *disjoint*. The system then provides mechanisms for generating attribute hierarchies based on these relationship assertions among attribute pairs. However, the user still is responsible for identifying entity clusters that may be related. BERDI contains several tools, such as access to dictionary information and graphical query and display facilities, to assist users in this process.

Another well prototyped toolkit is presented in Ramesh and Ram's paper [Ref.82]. The authors utilize knowledge about entity classes, attributes, and relationships to assign similarity values among entity class, attribute, and relationship pairs. well as among related objects represented using different constructs. [Ref.82]. Their approach to identifying interschema relationships measures the similarity or dissimilarity between entity classes, attributes, and relationships in the component schemas. They use two distinct measures: an index-of-similarity (IS) and an *index-of-dissimilarity* (ID) for this purpose. The IS can take on values between 0 and 1, and ID can take on values between 0 and -1. A high IS between objects indicates a high probability of a relationship between the objects. A high ID indicates that the objects in question may not be related.

Heuristics are used to reduce the search space of comparisons. If it is clear that the resulting value will not reach a certain desirable level, the primary heuristics may terminate the computation. [Ref.82]. Once the similarity values have been determined, an attempt is made to establish the type of relationship that could exist

163

among the various database objects. This is done partially using system heuristics and then confirmed by human integrators.

A key characteristic of the methodology in Ramesh and Ram [Ref.82] is that during schema integration, blackboard architecture is used to provide explicit support for the necessary human interaction. [Ref.82]. This is achieved by viewing human interaction as an additional knowledge source needed during schema integration. The system utilizes three types of knowledge sources: an interschema identification engine, an integrated schema generation engine, and the human integrators.

The four-level blackboard architecture is shown in Figure 5.11. When we follow the levels from bottom to top, the information in the levels closest to the goal state--or the generation of an integrated schema. The knowledge engines employ input information available at a lower level to deliver output information to higher levels. [Ref.82].

Goal Level	Integrated schema

Fact Level	User-confirmed interschema relationship

Assertion Level	System-generated interschema relationship

Data Level	Individual schemas to be integrated

Fig 5.11. Four-level Blackboard Architecture.

The data level of the blackboard stores a representation of each schema (using the common data model) that needs to be integrated. Hence, the schema transformation step is required before completing the integration process. This information represents the raw data that is going to be utilized by an interschema relationship identification (IRI) engine. By analyzing the schemas from the data level of the blackboard, the interschema identification engine generates temporary assertions about the similarity or dissimilarity between entity classes, attributes, and relationships in the schemas to be integrated. [Ref.52]. These assertions are placed on the assertion level of the blackboard.

The interschema identification engine consists of three distinct knowledge sources: entity class definition similarity, attribute similarity, and relationship similarity computation engines [Ref.82]. Information made available on the

blackboard activates these different knowledge sources. [Ref.82]. The interschema relationship identification scheduler (IRS) is responsible for monitoring the blackboard and triggering the appropriate knowledge sources as necessary. The output of IRI engines may not be always accurate, and needs confirmation from the user, so they are called *transient assertions*.

Humans use the information available at the assertion level of the blackboard during their role in the integration process. [Ref.82]. Their role is to transform the transient assertions generated by the interschema relationship identification process into facts by modifying or confirming them. These facts appear at the fact level of the blackboard. The human integrators interact with the blackboard through a graphical environment. This environment allows a group of designers to work simultaneously on modifying/confirming assertions placed on the blackboard. It also provides users with (computer) communication channels for resolving differing viewpoints among users during this process. Users can also use this environment to view other objects on the blackboard, such as individual schemas, the (partially) integrated schema, facts, and other assertions. [Ref.82]

The integrated schema generation process operates on the information available at the fact level. It consists of three components: the integrated schema generation scheduler (SGS), the preprocessor, and the integrator [Ref.82]. The SGS monitors the fact level of the blackboard and, when appropriate, activates the preprocessor and integrator components. The preprocessor inputs facts about similarity between entity classes (from the fact level) and attempts to generate possible equivalence relationships between them. The results of the computation are output to

the assertion level of the blackboard for confirmation by the human integrators. "The integrator utilizes information from the various areas of the fact level of the blackboard to integrate the individual schemas" [Ref.82]. Output from the integrator is placed on the goal level of the blackboard.

C. IDENTIFIYING SEMANTICALLY EQUIVALENT OR RELATED DATA ITEMS IN COMPONENT DATABASES.

To reiterate from previous chapters, an essential prerequisite to achieving interoperability in multidatabase systems is the ability to identify semantically equivalent or related data items in component databases. Another challenging task in multidatabase systems is to enable users to handle information from different databases that refer to the same real-world entity. In this section, we discuss an application of expert systems, *semantic networks,* that help to detect and resolve semantic heterogeneities among component databases. We will also discuss a semantic query language, SemQL, designed to capture the concepts users want to express. [Ref.101].

In this chapter, we have discussed the schematic integration techniques, but schema considerations alone are not enough to detect semantic heterogeneity; additional knowledge must be considered in order to gain semantic knowledge. As stated before, a completely automatic integration is impossible even with perfect expert systems due to the fact that such an expert system cannot represent the intentions of every database designers and administrators. However, it is possible to diminish the burden of humans by using some AI techniques and expert tools.

In this section, we present the usage of a linguistic knowledge in WordNet [Ref.102] for integrating information. WordNet is an on-line lexical dictionary and is organized by semantic relations such as synonymy, antonymy, hyponymy, and metonymy [Ref.103]. The noun portion of WordNet is designed around the concept of synset (synonym set) which is a set of closely related synonyms representing a word meaning.

Using the synsets in Wordnet and the descriptions of database objects, a semantic network can be constructed that provides semantic relations among database objects [Ref.101]. With this semantic network, semantic heterogeneities among component databases can be detected and resolved. On the other side, a semantic query language, such as SemQL (discuss in subsequent sections, can be used to capture the concepts about what users want. SemQL allows users to send queries to a large number of autonomous databases without any prior knowledge of their schemas. Figure 5.12 shows the outline of this approach.

1.   Semantic Networks

A multidatabase system can provide a uniform interface to a multitude of component databases. Consider the knowledge a multidatabase system would need to answer the following query:

"Find those ships whose length is over 100 meters."

To answer this question, the system must have several kinds of knowledge. It must know (a) where to find the relevant information on the component databases (*access knowledge*) and (b) which entities, attributes, or values in the component databases meet the semantics in the query (*semantic knowledge*).

Figure 5.12. Semantic Network Integration Approach.

To acquire this knowledge, semantic conflicts must be first be dealt with. Semantic networks, which specify the relations among entities, attributes, and value domains in component databases, can be used to deal with these conflicts [Ref.101]. The basic idea is to use synonym sets from a lexical system such as WordNet to provide a mapping mechanism. Once a semantic network is constructed, semantic

169

heterogeneity based on it can be detected and resolved. In the following subsection, we again classify the semantic heterogeneity based on schema conflicts and data conflicts.

a. *Classification of Semantic Heterogeneity*

As previously explained, semantic heterogeneities include differences in the way the real world objects is modeled in the databases, particularly in the schemas of the databases. Since a database is defined by its schema and data, semantic heterogeneities can be classified into schema conflicts and data conflicts. Schema conflicts mainly result from the use of different structures for the same information, and the use of different names for the same structures. Data conflicts are caused by inconsistent data in the absence of schema conflicts. [Ref.104]. Figure 5.13 shows an example illustrating semantic heterogeneities.

As the focus is only on the schema conflicts, we assume that data conflicts, such as different representation for the same data, are already conformed. Focusing on schema conflicts, we present the following below: entity versus entity structure conflicts, entity versus attribute structure conflicts, entity versus value structure conflicts, entity versus entity name conflicts, and attribute versus attribute name conflicts.

(1) Entity versus entity structure conflicts: These conflicts occur when component databases (CDBs) use different numbers of entities to represent the same information. For example, CDB1 uses two entities, **SurfaceShip** and

170

Submarine for general information on the ship, while CDB2 and CDB3 use only one

entity, **Ship**.

Component Database 1 (CDB1):

SurfaceShip (hull#, name, length, displacement, country, city, CO#)

Submarine (hull#, name, length, displacement, country, city, CO#)

SurfaceCO (CO#, name, rank, salary, age)

SubmarineCO (CO#, name, rank, salary, age)

Component Database 2 (CDB2):

Ship (hull#, nm, type, CO#)

Port (country, city)

Captain (CO#, nm, type, rank, salary, age)

Component Database 3 (CDB3):

Ship (hull#, name, type, port, CO#)

Captain (CO#, name, type, salary, rank, age)

Component Database 4 (CDB4):

Warship (hull#, nm, surface, submergible, CO#)

CO (CO#, nm, rank).

Figure 5.13. Example Database Schemas.

(2)   Entity versus attribute structure conflicts: This type of conflicts occurs if an attribute of some CDBs is represented as an entity in others. In the sample database schemas, CDB3 uses an attribute, **port,** in the entity **Ship** to represent the ship's port, while CDB2 represents the same information in the entity **Port.**

(3)   Entity versus value structure conflicts: These conflicts occur when the attribute values in some CDBs are semantically related to the entities in other CDBs. For example, CDB1 uses two entities, **SurfaceShip** and **Submarine**, for surface and submergible ships, respectively, while the same information for type is represented as values of an attribute **type** in CDB3.

(4)   Attribute versus attribute structure conflicts: These conflicts occur when CDBs use different numbers of attributes to represent the same information. For example, CDB1 uses two attributes, **country and city**, for information on port, while CDB3 uses one attribute, **port.**

(5)   Attribute versus value structure conflicts: These conflicts occur when the attribute values in some CDBs are semantically related to the attributes in others.   In the example database schemas, CDB4 uses two attributes, **surface** and **submergible,** for information on type, while the same information is represented as values of an attribute **type** in CDB3.

(6)   Entity versus entity name conflicts: These conflicts arise due to different names assigned to entities in different CDBs. For example, the entity for a war ship is called **Ship** in CDB3 and **Warship** in CDB4.

(7)   Attribute versus attribute name conflicts: Attribute name conflicts are similar to the entity name conflicts. An example of this conflict type occurs in the example database schemas where the attribute for ships's name is called **name** in CDB3 and **nm** in CDB4.

b.   *WordNet as an on-line Lexical Dictionary*

As the term "word" is used to refer to the physical utterance and the lexicalized concept used to express a form, the phrase "word form" is used for the former, and "word meaning" for the latter in order to reduce ambiguity. Hence, the starting point for lexical semantics can be declared to be the mapping between forms and meanings. [Ref.103].

WordNet [Ref.102] is the product of a research project at Princeton University that attempts to model the lexical knowledge of a native speaker of English. The system has the power of both an online thesaurus and an on-line dictionary, and much more. Information in WordNet is organized around logical groupings called synsets. Each synset consists of a list of synonymous word forms and semantic pointers that describe relationships between the current synset and other synsets. The semantic pointers can be of a number of different types including, Synonymy (X has the same

sense with Y), Antonymy (X has the opposite sense with Y) , Hyponymy (X is a kind of Y) and Metonymy (X is part of Y) [Ref.102].

Mappings between forms and meanings are many-to-many (some forms have several different meanings, and some meanings can be expressed by several different forms). Synonymy can be viewed as complementary aspects of this mapping. Polysemy and synonymy are problems that arise in the course of gaining access to information in multidatabase systems. Two difficult problems of information integration are polysemy and synonymy. Polysemy is the ambiguity of an individual word or phrase that can be used in different contexts to express two or more different meanings.

The case of "commanding officer (CO)" and "captain" is an example of synonymy, while the word form "captain'" has many different meanings, which is an example of polysemy. The following, Figure 5.14, shows several different meanings returned for *"captain"* in WordNet following a querying of the on-line site of WordNet about the definition of the word, *"captain"*.

c.   *Constructing Semantic Networks*

During the construction of a semantic network, as semantically similar entities and attributes might be abbreviated differently in different databases, the names of entities and attributes are generally not used. Instead, the words in an online lexicon (e.g. WordNet) that describe the concepts of the entities and attributes are used. A semantic network provides mappings between words in the lexicon and words provided by the local DBAs using synsets in Lexicon [Ref.101]. After we describe the construction steps we will show an example (Figure 5.15):

The overall process of constructing a semantic network is as follows.

STEP 1: At each CDB, the local DBAs must make descriptions of entities and attributes. As we said before, it is not possible for any system to capture semantics without human interaction, so creating a semantic network requires some initial input from the CDB administrators. For the sake of simplicity, most of the approaches allow the local DBAs to use only single *nouns, compound nouns,* or *noun phrases* in making the aforementioned descriptions [Ref.101].

WordNet 1.6 overview for "captain"

The **noun** "captain" has 7 senses in WordNet.

1. captain -- (an officer holding a rank below a major but above a lieutenant)
2. captain, skipper, commanding officer -- (the naval officer in command of a military ship)
3. captain, police captain, police chief -- (a policeman in charge of a precinct)
4. master, captain, sea captain, skipper -- (an officer who is licensed to command a merchant ship)
5. captain, chieftain -- (the leader of a group of people; "a captain of industry")
6. captain, senior pilot -- (the pilot ins charge of an airship)
7. captain, head waiter, maitre d'hotel -- (a diningroom attendant who is in charge of the waiters and the seating of customers)

Search for [   ] [▼] of senses [   ]
☑ Show glosses
☐ Show contextual help

Figure 5.14. Different Senses of "*Captain*" in the Wordnet Online Dictionary.

STEP 2: The descriptions produced in STEP 1 are decomposed into unit nouns, (i.e., single nouns, compound nouns, or noun phrases which must be found in the lexicon . For example the compound noun, *"commanding officer,"* can be found in WordNet, and is treated as a unit noun in constructing a semantic network.

STEP 3: After making a description table, the component database administrators must deal with synonymy and polysemy. Also, the noun portion of the lexicon must be designed in synonym sets (synsets), which are sets of closely related synonyms representing word meanings. To identify unit nouns related by synonymy *automatically*, the synsets in the lexicon are used [Ref.101]. However, to obtain the correct meaning of a unit noun, its polysemy must be manipulated by administrators. For example, when a local administrator inputs a unit noun "captain" into the lexicon, he must choose only one among many meanings. Actually, this procedure is a tedious task for the local DBAs, but the remaining steps in the creation and are automatic.

STEP 4: Given the sets of unit nouns in the two component databases (CDBs), say UcDBi and UcDBj, each unit noun in UcDBi is compared with the unit nouns in UcDBj, respectively, using synsets in the lexicon.

Now, to understand the steps mentioned above, we give an example from the Navy domain and construct a partial semantic network step-by-step by using the online lexicon WordNet. Our component databases will be used to show the semantic

heterogeneities (Figure 5.13). For simplicity we will use only first two databases (CDB1 and CDB2) which are given in Figure 5.15.

Component Database 1 (CDB1):

SurfaceShip (hull#, name, length, displacement, country, city, CO#)

Submarine (hull#, name, length, displacement, country, city, CO#)

SurfaceCO (CO#, name, rank, salary, age)

SubmarineCO (CO#, name, rank, salary, age)

Component Database 2 (CDB2):

Ship (hull#, nm, type, CO#)

Port (country, city)

Captain (CO#, nm, type, rank, salary, age)

Figure 5.15. Schemas of CDB1 and CDB2.

In our sample databases, although many entity and attribute names are straightforward, there are also some entity and attribute names that may not make sense to a civilian, for example "CO", "Hull#", and SubmarineCO. In Step 1, DBAs must describe these names. The following table (Table 6.5) shows the list of all entity and attribute names in their described form.

177

Entity/Attribute Name	Description	Respective number In Figure 5.16
SurfaceShip	Surface ship	-
Submarine	Submarine	-
SurfaceCO	Surface ship commanding officer	-
SubmarineCO	Submarine commanding officer	-
hull#	Hull number	1
Name	Name	2, 9
Length	Length	3
Displacement	Displacement	4
Country	Country	5
City	City	6
CO#	Commanding officer number	7, 8
Rank	Rank	10, 22
Salary	Salary	11, 23
Age	Age	12, 24
Ship	Ship	-
Port	Port	-
Captain	Captain	-
Nm	Name	20
Type	Type	21

Table 6.5. Description List.

After the entity and attributes names are described by the administrators, in Step 2, these descriptions are decomposed into unit nouns. (Recall that a unit noun is a single noun, a compound noun or a noun phrase existing in the lexicon.) This process can be done automatically, and this thesis proposes an algorithm developed for that algorithm purpose. The following simple pseudo-code describes the basic form of our algorithm. We assume that a description can have at most three single nouns. We extend this number easily by adding some extra lines:

178

DSet is a set of name descriptions in a database.

D is a description and D ∈ DSet.

D = [wi.wj....wt] where i ≤ k and 1 ≤ t ≤ x,   i, t, x ∈ Z

For ∃ D ∈Dset {

    // if the whole phrase exists in the lexicon

    if D = [wi] and [wi] is in Lexicon

        UnitName U = D; *// unit name will be whole phrase*

    // We decompose the description into two nouns

    Else if D = [wi.wj] AND wi and wj are in Lexicon

        UnitName U1= [wi] and U2 = [wj]
    // We decompose the description into three nouns

    Else if D = [wi.wj.wk] AND [wi.wj] and [wk] are in Lexicon

        UnitName U1= [wi.wj] and U2 = [wk]
    Else if D = [wi.wj.wk] AND [wi] and [wj.wk] are in Lexicon

        UnitName U1= [wi] and U2 = [wj.wk]
    Else if D = [wi.wj.wk] AND [wi], [wj], and [wk] are in Lexicon

        UnitName U1 = [wi], U2 = [wj], and U3 = [wk]

    }

Table 6.6 gives the results of this process.

Description	Unit Names (exist in Wordnet)
Surface ship	[surface ship]
Submarine	[submarine]
Surface ship commanding officer	[surface ship] [commanding officer]
Submarine commanding officer	[submarine] [commanding officer]
Hull number	[hull] [number]
Name	[name]
Length	[length]
Displacement	[displacement]
Country	[country]
City	[city]
Commanding officer number	[commanding officer] [number]
Rank	[rank]
Salary	[salary]
Age	[age]
Ship	[ship]
Port	[port]
Captain	[captain]
Name	[name]
Type	[type]

Table 6.6. Descriptions in Unit Noun Forms.

After the unit names list is obtained, the administrator should indicate the intended senses of unit names to cope with polysemy in Step 3. For example, "captain" must be selected from WordNet with the definition, "the naval officer in command of a military ship." Given the sets of unit nouns in the two component databases (CDBs), UCDB1 and UCDB2, each unit noun in UCDB1 is compared with the unit nouns in UCDB2, respectively, using synsets in the lexicon. Hence, the same unit names are used only once in the network. For example, we obtained three "commanding officers," and

we would not need to use three of them, just one would be enough in the semantic network.

Also another property of WordNet helps us to find the hypernym relationships (X is a kind of Y) between unit names. With this property the *lubs (lowest upper bound)* of unit names can be found. We believe that *lubs* can be used for many goals, some of which are explained in the following sections.

We can now establish our semantic network shown in Figure 5.16. We also add the *lubs* into our network.

2.   Semantic Query Processing

When we look at the process of database design from the semantic perspective, the scope spans from real world objects to data representation. As mentioned earlier, every designer develops his own conceptualization of the "real world" according to his habits and intentions, turning this conceptualization into a database design. As a result, different, generally incompatible schemas for the same information are developed. Therefore, users are faced with the problem of locating and integrating relevant information when they need to combine information from several databases.

The simplest solution to this problem is simply allowing the user to access the relevant databases. But how can the user know all the relevant databases? Assume he or she knows the locations, but how can he or she knows all of the schemas of the relevant databases. Therefore, this solution requires the user to learn all the schemas-- an unacceptable burden to the users.

Figure 5.16. Semantic Network.

182

A more efficient and effective approach is to allow users to issue queries to a large number of autonomous databases with his/her own terms. It frees users from learning independent schemas. A semantic query language such as SemQL can be used to issue queries not with schemas, but rather with *concepts* known by the users.

a.   *Semantic Query Languages*

In a relational database, semantic query languages are generally similar to SQL except that they have no FROM clause, and are basically formed of the two clauses SELECT and WHERE, and have the following form [Ref.101],

SELECT <concept list>

WHERE <condition>

Here <concept list> is a list of concepts whose values are to be retrieved by the query [Ref.101]. The <condition> is a conditional search expression that identifies the tuples to be retrieved by the query like in classic SQL.

<concept list> consists of concept expressions. A concept expression is a concatenation of entity and attributes names similar to SQL. However, these names are not real data object names, rather they are concept names. A basic concept expression is in the following form:

<concept expression> = <entity concept name>.<attribute concept name>.

The <condition> clause is similar to SQL's condition clauses, but again, instead of real object names, concept expressions are used. For example, when a user wants to learn the skipper of the USS Alabama, he will issue the following query:

SELECT skipper.name

WHERE submarine.skippername = skipper.name AND submarine.name = "TCG Doganay".

b. *Semantic Query Processing Procedure*

Although, there are some differences among the query processing capabilities of different languages, here we discuss only the SemQL to give a general idea of query processing.

The SemQL processor consists of *Query Parser, Resource Finder, Mapping Generator, Query Distributor,* and *Integrator.* The overall functionality of SemQL is completed in the following steps [Ref.101].

STEP 1: The users issue a semantic query (consisting of concept expressions mentioned above) with his/her own concepts to retrieve equivalent or related data items.

STEP 2: The *Query Parser* parses the query and extracts entity, attribute and value concepts from the query.

STEP 3: The *Resource Finder* identifies the relevant component databases in which the concepts exist by looking at the semantic network.

STEP 4: The *Mapping Generator* generates the necessary mappings between the original concepts in the query and representations in component databases.

STEP 5: The *Subquery Generator* reformulates the original query into multiple subqueries acceptable to the CDBs. The terms in the subqueries are those generated by the Mapping Generator. In this step, looking up the semantic network, Subquery Generator adds FROM clauses to the subquery.

STEP 6: The Query Distributor submits the sub-queries to the component databases.

STEP 7: The component databases receive the subqueries and execute them. After the execution, the results are returned to SemQL processor.

STEP 8: The integrator merges the intermediate results from the component databases and presents the integrated results to the users.

To better explain these steps, we provide an example from the previous databases (CDB1 and CDB2). For example, a user issues a query to learn the names of skippers who are under the age of 35. We assume that the user has no idea about the schemas of the databases and knows only the basic concepts. Such a query would be in following form in natural language.

QUERY: Find those skippers whose age is under 35.

The user issues the SemQL correspondence of this query as follows to the SemQL Processor (STEP 1):

SELECT skipper.name

WHERE skipper.age < 35.

The Query Parser parses the query and extracts concepts from the query (STEP 2). These extracted concepts are { skipper, name, age }. In Lee and Baik's approach [Ref.101], these concepts are compared with the unit nouns in the semantic network (on the left side of Figure 5.16). If one of these concepts can not be matched with a unit noun, the query fails and gives no result. We suggest that, the system also produces the hypernyms (X is a kind of Y) of the concepts.

As remembered, we already added the *lubs* of unit nouns into the semantic network. According to our approach, first the Resource Finder component of SemQL detects the component databases by looking up the semantic network and comparing the unit names and concept names. After that, it compares the *lubs* of unit names with the hypernyms of the concepts (STEP 3).

In our example, even though "name" and "age" exist in semantic network as a unit name, "skipper" does not exist. The first order hypernym *"skipper"* in WordNet is "officer". By following the chain downward starting from the word, "officer", we see that "skipper," "commanding officer," and "captain" are in the same synset (synonym set). So the resource finder detects CDB1 and CDB2 as relevant databases.

In Step 4, the Mapping Generator identifies the mappings between concepts in the original query and representation in CDB1 and CDB2. Figure 5.17 shows the result of the mapping procedure.

Figure 5.17. Mapping Procedure

Upon completion of the mapping procedure, the Subquery Generator generates subqueries (STEP 5) for CDB1:

SELECT name

FROM SurfaceCO

WHERE age < 35

UNION

SELECT name

FROM SubmarineCO

WHERE age < 35

And for CDB2;

SELECT nm

FROM Captain

WHERE age < 35.

And then the Query Distributor submits the two subqueries to CDB1 and CDB2 respectively (STEP 6).

The two component databases, CDB1 and CDB2, return the result tuples of the subqueries to the SemQL Processor (STEP 7).

The Integrator merges the results and presents the integrated results to the user (STEP 8).

D.   SUMMARY

In this chapter, we discuss the general information integration techniques. In Section A, we present the remote-exchange approach that represents the general mechanism for resolving representational heterogeneity in the context of a multidatabase system.   For these kinds of mechanisms to operate effectively, each participating component must agree to meet two principal conditions: First, a common data model must be supported at the federation interface, including a set of sharing functionalities

(like RSL in remote-exchange). Second, a structure-like local lexicon must be provided, wherein a component describes the meaning of the (type) objects it is willing to share with other components in federation. These objects must be described using the conceptual relationship descriptors.

In order to facilitate the establishment of a multidatabase system, we showed the importance of ontologies, which contain an initial set of terms that can be used to describe unknown concepts in the local lexicon of each component. A complete ontology package describes the general and specific information from the application domain and will evolve and grow by time to accommodate additional, more complex concepts within a given federation. Such packages can be provided for specific domains such as military environment or medical society.

In Section B, we emphasized the importance of schema integration for providing interoperability among heterogeneous databases. We describe the key problems and four-step process for schema integration. We also presented a framework for developing automated schema integration tools using blackboard architecture.

Finally, in Section C, we have discuss the semantic networks for data integration. Particularly, we used WordNet, an electronic online dictionary, as a background knowledge source to establish a semantic network. We show that semantic networks can be created from the word meanings from synonym sets in WordNet and descriptions of database objects. With the semantic relations presented in semantic networks, semantic heterogeneities among component databases can be detected and resolved. Also, we introduce a semantic query language, SemQL, to capture the concepts about

189

what users want. SemQL allows users to issue queries with the user's own concepts, which frees them from learning various component database schemas.

VI.   SUMMARY, CONCLUSIONS AND FUTURE WORK

A.   SUMMARY

Information needed by users may be supplied by two or more sources, each of which resides at a different physical location in a distributed system. Typically, information sources are designed from scratch, and are independent of each other. This has led to proliferation of information sources created according to different sets of requirements for modeling identical or similar real-world objects. As the amount of information on different platforms increases, the critical need to combine these data sources grows. The challenge is to give the user the sense that he or she can access a single information source that contains everything they need, that is, make the multi-source distributed nature of the system transparent to the user. However, any successful solution to the integration of preexisting information sources must address issues related to *autonomy* and *heterogeneity*.

To address these issues, different approaches have been proposed. Early research ignored the autonomy issue and the emphasis was put on finding solutions to heterogeneity; these are known as *global schema integration* approaches. All of the schemas from component information sources are integrated statically while the system is being built. The advantage of such an approach is that user has a fully consistent, uniform view of, and access to, data. However autonomy is completely sacrificed for the sake of the solution of the heterogeneity.

Also, creating a global schema is not a simple task and requires human interaction. Even if one assumes that the integration process was performed automatically by expert systems, since the *global schema* is the center of the system, the verification of the correctness of the schema is manually intensive task. As a result, the global schema approach is suitable only for small systems in which the number of information sources is very small, changes in component schemas are very rare, and the system is closed to new information sources.

The *multidatabase language* approach evolved as way to get around some of the drawbacks of a global schema integration approach. This approach is intended for users of a multidatabase system who do not use a predefined global schema. The goal of having a multidatabase language is to create constructs that perform queries involving several information sources at the same time. The main drawbacks of this approach are the lack of distribution and location transparency for users. Users not only have to find the right information in a potentially large network of information sources, but they are also responsible for understanding the schemas, detecting and resolving semantic conflicts, and performing view integration. Therefore, this approach is suitable for small simple systems.

In contrast, the *Federated Database Systems* (FDBS) approach removes the need for static global schema integration. The integration is performed dynamically when needed, and the amount of information delivered depends upon the needs of the users. Because there is no need for static and complete schema integration, the number of component information sources can be high, and theoretically can include all information

sources. The mediator-wrapper architecture adopted by most research groups is an example of subscribing to a FDBS approach.

Although FDBS approach appears to be the most convenient information integration approach, it is flawed when it comes to identifying objects in different, largely distributed information sources that are semantically related, and then resolving the schematic differences among semantically related objects. It is apparent that this is not a task a human can do. Therefore, at least some parts of integration process must be automated.

In this thesis, the focus is on the semantic heterogeneity issue in the integration process. In Chapter IV, different approaches are presented to represent the semantic relationship between real-world objects in different information sources. We have emphasized that the real-word semantics of data objects can not be captured using mathematical formalism. We need to represent more of the mostly human-based knowledge in order to capture the semantics of the relationships between objects. Database administrators must provide this knowledge when integration process begins. We try to lessen this burden for humans as much as possible by employing some knowledge sources.

Particularly, we have adopted the semantic network approach and WordNet as the knowledge source to identify the relevant information sources and resolve semantic conflicts. In similar semantic network approaches, researchers propose to use synonym sets in a lexicon to identify the semantically related objects. If the object names in different information source are in the same synonym set, then these objects are identified as related objects (equivalent or identical). In addition to this strategy, we have suggested

that the hypernym (X is a kind of Y) relationships between object names can also be used to identify other semantic relationships.

B. CONCLUSIONS

As a result of our study, we conclude that, a fully automated integration between independent information sources is difficult to accomplish due to the fact that every information source is created by individuals with their own conventions, needs, and intentions. Expert systems that serve as a knowledge base to resolve semantic differences can only help expedite the integration process. Otherwise, there is no such knowledge base capable of capturing the knowledge of all people who design and administer independent data sources.

When a natural language is concerned, humans tend to make incorrect matches between the meanings and utterances of words. This is significant as a perfect expert system can only reflect the abilities and knowledge of people who use natural languages. We cannot rule out the information sources designed and administered by people who err in the use of their own language. Therefore, future work in this area should concentrate on increasing the speed of integration by employing more knowledge and tools.

C. FUTURE WORK

In this thesis, we discuss only the integration of structured data sources, such as databases and record-based files, and the heterogeneity of information sources. Although we discuss the mediator-wrapper architecture to cope with semistructured and

unstructured data, the deployment of mediators and wrappers in various application domains was not mentioned, and may be a good future thesis topic. Like other researchers, we have overlooked the issues of autonomy and security, which have not been thoroughly addressed in the literature. Concerns about security and autonomy issues in multidatabases should be addressed in future works. We show the usage of semantic networks in the semantic resolution process, but this usage is only based on theory, and we did not test our approach to assess its effectiveness. The implementation of a semantic network to see how many information sources can be handled may also be a worthwhile thesis topic.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A: BACKGROUND OF EXPERT SYSTEMS

A. EXPERT SYSTEMS

1. The Nature of Expertise

An expert system is a computer program that employs knowledge of a specialist subject with the objectives of solving problems and making recommendations. The following conditions are necessary to call a computer program an expert:

First, the computer program should possess *knowledge*. For example, it is not sufficient for a program to rely solely on checklists or a list of algorithms. Secondly, this must be focused upon a *specific domain*. Miscellaneous collections of names, dates, places, and historical facts would not be considered a basis for expertise knowledge. Knowledge implies organization and integration in that different pieces of knowledge should interrelate in a logical and purposeful union. Finally, this knowledge must be capable of *solving problems* directly, implying an ability to accomplish tasks on demand. Merely *demonstrating* knowledge relevant to technical support should not be considered the equivalent to *solving* problems.

An expert system may completely fulfill a function that normally requires human expertise, or may assist a human decision maker. In other words, the client may interact with the program directly, or interact with a human expert who interacts with the program. In the latter case, an expert system may justify its existence by improving the productivity and increasing the quality of the decision maker's output. For example, the human collaborator may be someone who is capable of attaining expert levels of

performance given some technical assistance from the program, tantamount to achieving successful expert systems deployment.

Expert systems technology derives from the research discipline of *Artificial Intelligence* (AI) concerned with the design and implementation of programs that are capable of emulating human cognitive skills such as problem solving, visual perception, and language understanding. This technology has been successfully applied to a wide range of fields, including organic chemistry, mineral exploration, and internal medicine. Typical tasks for expert systems involve the following [Ref.105]:

- Interpretation of data, such as sonar signals;

- Diagnosis of malfunctions, such as equipment faults or human diseases;

- Structural analysis of complex objects, such as chemical compounds;

- Configuration of complex objects, such as computer systems; and

- Planning sequences of actions, such as might be performed by robots.

Although more conventional programs have been known to perform similar tasks in similar domains, we shall argue in the next section that expert systems are sufficiently different from such conventional programs so as to form a distinct class. There is no precise definition of an expert system that is guaranteed to satisfy everyone. However, certain features of critical importance should be possessed, to some degree, by any system classified as "expert."

2. The Characteristics of an Expert System

An expert system can be distinguished from more conventional applications programs in that:

- **It simulates human reasoning about a problem domain, rather than merely manipulating information about the domain itself.** This characteristic distinguishes expert systems from more familiar programs that involve mathematical modeling or computer animation. This is not to say that the program is equally capable as the human expert; rather, the ideal is to emulate an expert's problem solving abilities, enabling an expert system to perform relevant tasks as well as, or better than, the expert.

- **It applies reasoning over representations of human knowledge, in addition to doing numerical calculations or data retrieval [Ref.105].** The knowledge in the program is normally expressed in some special-purpose language and kept separate from the code that performs the reasoning. These distinct program modules are referred to as the *knowledge base* and the *inference engine,* respectively.

- **It solves problems by *heuristics* or *approximate methods* that, unlike algorithmic solutions, are not guaranteed to succeed [Ref.105].** A heuristic is essentially "a rule of thumb that encodes a piece of knowledge about how to solve problems in some domain" [Ref.105]. Such methods are approximate in the sense that (1) they do not require perfect data and,

(2) the solutions derived by the system may be proposed with varying degrees of certainty. [Ref.105]

An expert system also differs from other kinds of artificial intelligence programs in that:

- **It deals with subject matter of *realistic complexity* that normally requires a considerable amount of human expertise** [Ref.105]. Many AI programs are really research vehicles, and may therefore focus upon abstract mathematical problems or simplified versions of real problems in order to gain insights or refine techniques. Expert systems, on the other hand, solve problems that are of genuine scientific or commercial interest.

- **It must exhibit *high performance* in terms of speed and reliability in order to be a useful tool** [Ref.105]. AI research vehicles may not run very fast, and may likely contain bugs. They are programs, not supported software. But an expert system must propose solutions in a reasonable time and be right most of the time, (that is, at least as often as a human expert.)

- **It must be capable of *explaining and justifying solutions* or recommendations in order to convince the user that its reasoning is in fact, correct** [Ref.105]. Research programs are typically only run by their creators, or by other personnel in similar laboratories. An expert system will be run by a wider range of users, and should therefore be designed to ensure ease of operation for a wide variety of users.

The term *knowledge-based system* is sometimes used as a synonym for "expert system," although the former is more general. A knowledge-based system is any system that, instead of relying mostly on algorithmic or statistical methods, performs a task by applying an established set of rules to a symbolic representation of knowledge. Thus, a program capable of conversing about the weather would be a knowledge-based system, even if that program did not have any expertise in meteorology. But, an expert system in the domain of meteorology should also be able to provide us with weather forecasts.

In summary, expert systems encode the field specific knowledge of everyday practitioners in a particular field, which is then used to solve problems, instead of using universal problem solving techniques generated by computer science or mathematics. The process of constructing an expert system is often called *knowledge engineering,* and is considered to be applied artificial intelligence.

3. Fundamental topics in expert systems

It is not surprising that, given that expert systems research has grown out of more general concerns in artificial intelligence, this type of research maintains strong intellectual links with related topics in its parent discipline. Some of these links are outlined in the following sections.

Knowledge acquisition, or the transfer and transformation of problem-solving expertise from one knowledge source to a program, is usually a result of a series of complex interviews between a knowledge engineer, who is normally a computer specialist, and a domain expert. It is estimated that this knowledge acquisition produces between two and five units of knowledge per day, a rather low output that has led

researchers to regard knowledge acquisition as "the bottleneck problem" of expert systems applications [Ref.107].

There are a number of reasons why productivity is typically low. Specialist fields have their own lexicon, making it difficult for experts to communicate their knowledge in everyday language. Secondly, the facts and principles underlying many domains of interest cannot be adequately represented in terms of a mathematical theory or model whose properties are clearly defined. Statistical models may enable us to make rather general, long-term predictions, but generally do not provide short-term solutions.

Additionally, experts need to know more than the mere facts or principles of a domain in order to solve problems. For example, they usually know which information is relevant to which kinds of judgment, how reliable different information sources are, and how to make hard problems easier by breaking them down into more simple problems which can usually be solved independently. Such knowledge, which is normally based on personal experience rather than formal training, is much more difficult to acquire than basic facts or principles. Human expertise, even in a relatively narrowly defined field, is often set in a broader context that includes common sense and everyday knowledge.

Dissatisfaction with the interview method has led some researchers to try to automate the process of knowledge acquisition. One area of research concerns *automated knowledge elicitation,* in which there is a transfer of expert knowledge to a computer program. Other researchers have looked to the sub-field of AI known as *machine learning* for a solution to the bottleneck problem. The idea is that a computing system could perhaps learn to solve problems in much the same way that humans do. (Knowledge acquisition is described in greater detail in Section D.)

Knowledge representation (Section B) is itself a substantial sub-field, and is concerned with the ways in which information is stored and associated in the human brain, usually from a logical, rather than a biological, perspective. In other words, knowledge representation is not typically concerned with the physical details of how knowledge is encoded, but rather with what the overall conceptual scheme might look like [Ref.105].

In the 1970s, knowledge representation research attempted to address such questions as how human memory works, proposing theories of reminding, recognition, and recall. Some of the resultant theories led to computer programs which tried to simulate different ways in which concepts might be associated, so that a computer application would be able to, when attempting to solve a problem, find the right piece of knowledge at the right time.

In the world of expert systems, knowledge representation is mostly concerned with finding ways in which large bodies of useful information can be transcribed for the purpose of symbolic computation (See Section C for further explanation). *Formally described* means the material is written in unambiguous language or notation with well-defined syntactic and semantic systems revealing the meaning of expressions. *Symbolic computation* is the representation of non-numeric computations using symbols and symbol structures that define concepts and the relationships between them.

AI researchers have made a good deal of effort in constructing *representation languages,* that is, computer languages that are oriented towards organizing descriptions of objects and ideas, rather than stating sequences of instructions or storing simple data elements. The main criteria for assessing a representation of knowledge are logical

adequacy, heuristic power, and notational convenience, terms deserving some explanation.

Logical adequacy means that the representation should be capable of making all the distinctions that we want to make [Ref.105]. For example, it is not possible to represent the idea that every drug has some undesirable side effect unless we are able to distinguish between the designation of a particular drug and a particular side effect and the more general statement to the effect that: 'for any drug, there is an undesirable side effect associated with it.'

Heuristic power means that as well as having an expressive representation language, "there must be some way of using representations so constructed and interpreted to solve problems" [Ref.105]. It is often the case that the more expressive the language, in terms of the number of semantic distinctions that it can make, the more difficult it is to control the drawing of inferences during problem solving. Many of the formalisms that have found favor with practitioners may seem quite restricted in terms of their powers of expression when compared with English or even standard logic. Yet they frequently gain in heuristic power as a consequence; that is, it is relatively easy to bring the right knowledge to bear at the right time. Knowing which areas of knowledge are most relevant to which problems is one of the things that distinguishes the expert from the amateur or the merely well-read. [Ref.105]

Notational convenience is a virtue because most expert systems applications require the encoding of substantial amounts of knowledge, and this task will not be an enviable one if the conventions of the representation language are too complicated [Ref.105]. The resulting expressions should be relatively easy to write and to read, and it

should be possible to understand their meaning without knowing how the computer will actually interpret them. "The term *declarative* is often used to describe code which is essentially descriptive and can therefore be understood without knowing what states a real or virtual machine will go through at execution time" [Ref.105].

Several conventions for coding knowledge have been suggested, including *production rules* [Ref.108], *structured objects* [Ref.109], and *logic programs* [Ref.110]. Most expert systems use one or more of these formalisms, and their pros and cons are still a source of controversy among theoreticians.

Expert systems design requires close attention to the details of how knowledge is accessed and applied during the search for a solution. Knowing what one knows, and knowing when and how to use it, is *metaknowledge,* or knowledge about knowledge, and is implied by the concept of "expertise

Different strategies for bringing domain-specific knowledge to bear will have profound effects upon a program's performance characteristics. These strategies will determine the manner in which a program *searches* for a solution. The data given to a knowledge-based program will generally not be sufficient for the program to know exactly where it should look for answers.

B. KNOWLEDGE REPRESENTATION

1. Early Studies

Most knowledge representation formalisms can be employed under a variety of *control regimes*, and expert systems researchers have in the past and are continuing to experiment in this area. During the mid-1960s to the mid-1970s, computer programmers were very concerned with making machines understand natural language, especially stories and dialogue.

Winograd's (1972) SHRDLU system was arguably the climax of this epoch. His development was a program that was capable of understanding an impressive subset of English, accomplished by representing and reasoning a very restricted domain (a world consisting of children's toy blocks). The program exhibited an understanding of natural language by modifying its representations of the fictitious world in response to commands, and by responding to questions about both the configuration of and actions performed on the blocks. Thus it could answer questions like, "What is the color of the block supporting the red pyramid?" and derive plans for obeying commands such as, "Place the blue pyramid on the green block."

SHRDLU was said to understand sentences because it responded appropriately. The rationale for this view of understanding is called *procedural semantics*. The principle underlying the concept of procedural semantics is simply that if a program can respond to or answer a question appropriately, then the program is said to have "understood." Similarly, if a program can correctly carry out a command, then it has "understood" the language used in the process of issuing and responding to a command.

Another line of research attempted to achieve language comprehension in less artificial and more everyday contexts, such as visits to a doctor, or dining in a restaurant. A structure called *script* was used to represent the various elements that make up a real-world situation. Scripts can be thought of as formulas for the objectives, tasks, and customs that are associated with particular events. Thus, "a 'restaurant script' would be activated by an 'eating' goal, be satisfied by a 'meal' solution, and would assemble intermediate knowledge about seating, menus, checks, tips, and the like" [Ref.112].

Other researchers attempted to model human problem-solving behaviors on simple tasks, such as puzzles and word games. The goal was to make the knowledge and strategies used by the program resemble the knowledge and strategy of the human subject as closely as possible. The fundamental problem with such attempts is that there is no way to demonstrate that humans and AI programs are accomplishing the same tasks in the same ways. Thus, indirect arguments must be used to show that, for example, the program and human subject need the same amount of time to solve problems, or make the same types of error when presented with faulty data. Simply showing that the programs get the same answer is obviously not enough, because there are a multitude of different strategies and encodings of knowledge that will solve the same problem.

2. Knowledge representation schemes

The new emphasis on knowledge representation generated a scheme known as *production rules* which proved to be an extremely productive asset for computer science (see Section E). This scheme has since become a mainstay of expert systems design and

development, and can be seen as a direct precursor of some of the knowledge elicitation techniques that engineers use today.

Researchers have explored numerous possibilities for encoding both particular facts and general principles about the world in such a way that they could be applied by a computer program using goal directed reasoning. These possibilities involved using constructs such as:

- If-then rules, i.e., "*if these conditions hold, then apply this operator*";

- Various kinds of *networks,* where nodes stand for concepts and arcs for relationships between them; and

- *Logical formulas* for encoding facts and principles, including control information about when to draw what inferences.

Sometimes these constructs are used in combination. However, most of the programs produced at this time are essentially research vehicles, and few of them found their way into real applications.

The concept of "computer understanding" is entirely problematic, largely because the conditions under one is prepared to conclude that a machine understood anything are not clear. However, even if one is unsure about what would constitute adequate grounds for computer understanding, the following conditions would be essential.

One such condition is the ability to represent knowledge about the world, and reason using representations. Expert systems exhibit this ability in one sense, as they can (a) possess representations of knowledge about specialist fields, and (b) they can apply this knowledge in order to solve real problems.

Another sign of understanding is the ability to perceive equivalences or analogies between different representations of the same situations [Ref.105]. Expert systems perform badly here, since they expect their inputs to be in a certain form, namely one that corresponds to their stored knowledge. A deviation from a computer's expected patterns tends to result in breakdown or unpredictable behavior.

Finally, understanding implies an ability to learn in a dynamic process in which new information is linked to existing information in a logical and potentially productive system. Few expert systems have demonstrated this kind of facility, although some progress towards machine learning has been made in recent years. Also, progress has been made in the design of programs which elicit knowledge from experts via an interaction at the terminal and then compile that knowledge into an applications program.

Although current expert systems fall short on some of these criteria, it is arguable that they do not have to "understand" a domain in the way that a human can in order to solve problems. This claim is based on the theory that it is not necessary to establish a connection between a particular problem solving process and the solution itself. In other words, all we require is that an expert system gets more or less the same answer as an expert, or helps an expert get the right answer. We do not demand that the system go through the same steps of reasoning as a human would, or to organize its domain knowledge in exactly the same way.

3.   Knowledge is Power

During the modern period of AI, the belief that the measure of success of a program is its ability to quickly and efficiently access relevant knowledge for problem solving

purposes. Researchers have developed techniques for encoding human knowledge in modules which can be activated by patterns. Whereas early attempts to simulate human problem solving strove for uniformity and simplicity, modern attempts at knowledge encoding have allowed more flexibility. Unlike more conventional problem solving programs, expert systems are expected to offer the user some kind of explanation as to how the conclusions were arrived at.

In response to questions about how experts do their job, few provide a well-articulated sequence of steps guaranteed to be successful in all situations. Rather, the knowledge that experts possess must be elicited by asking what they would do in specific contexts, and then probing further to determine the course of action in other cases.

As the process of accessing knowledge piece by piece seemed to be closer to the way that human experts store and apply their knowledge, the advantages of representing human knowledge in pattern-directed modules (instead of encoding it into an algorithm) became clear. This method of programming allows for fast prototyping and incremental system development. If the system designer and programmer have done their jobs properly, the resulting program should be easy to modify so that errors and gaps in the knowledge can be remedied without necessitating major adjustments to the existing code.

Practitioners accept that, in order to be useful, a program does not need to solve the whole problem, or even be right one hundred percent of the time. An expert system can act as an intelligent assistant, presenting alternatives in the search for a solution, and eliminating some of the less promising. The system can effectively leave the final decision and some of the intermediate strategic decisions to the user.

However, it is also recognized that rule-based systems are not easy to build and debug. As the knowledge base grows, rules will tend to interact in unexpected ways, by competing to be applied to the problem.

The Modern Period of AI has seen the development of a number of systems that can claim a high level of performance on non-trivial tasks. A number of principles have emerged which distinguish these systems from both conventional programs and earlier work in AI. The most important of these are considered below.

The part of the program that contains the representation of domain-specific knowledge, the *knowledge base,* is generally separate from the part that performs the reasoning, the *inference engine.* This means that one can make at least some changes to either module without necessarily having to alter the other. Thus one might be able to add more knowledge to the knowledge base, or tune the inference engine for better performance, without having to modify code elsewhere.

Practitioners try to use as uniform a representation of knowledge as possible, making the knowledge easier to encode and understand, and helping to keep the inference engine simple. However, uniformity can become problematic if different kinds of unique knowledge are forced into the same categories for the sake of encoding. This dilemma creates a conflict resulting in the need to choose between simplicity and variety in the representation of knowledge.

4. Principles and Techniques

In the field of expert systems, knowledge representation implies that we have some systematic way of codifying what an expert knows about some domain. However, it is erroneous to assume that representation is the same thing as encoding. If one encodes a message by systematically translating its symbols, the resulting code would not accurately represent a message, even though the code would be machine-readable and easy to store in the computer's memory. For one thing, the code would preserve any lexical or structural *ambiguity* of natural language inherent in the message. Thus the message:

'Visiting aunts can be a nuisance'

is just as ambiguous as it is in natural language. Transcribing it into code does not alter the fact that it could mean either, *It is a nuisance having to visit one's aunt*, or, *It is a nuisance having one's aunt visit*.

Also, any technical communication will assume that the addressee will have some prior *knowledge*. Needless to say, a computer has no such prior knowledge, and so any representation of the technical expertise required for problem solving must be self-contained.

Finally, representation implies *organization*. Simply encoding knowledge in a machine-readable form will not achieve organization. Relevant pieces of knowledge should be evoked by the circumstances in which they are most likely to be used. Also, a knowledge base should be extensively indexed so that any program can control the way

in which different pieces of knowledge are activated without having to know exactly how they are stored.

Of course, whatever notational system is used, ultimately a computer must be able to store and process the corresponding codes. This need is not a very constraining requirement, however, as distinct representational schemes can be expressed so that they are formally equivalent.

A representation has been defined as 'a set of syntactic and semantic conventions that make it possible to describe things' [Ref.115]. In artificial intelligence, 'things' normally means the state of some problem domain; for example, the objects in that domain, their properties, and any relationships that hold between them. A *description* uses representations to describe some particular thing.

The semantics of a representation specifies how expressions should be interpreted, or how meaning can be derived from form. This specification is usually performed by assigning meanings to individual symbols, and finally attaching meaning to more complex expressions.

The syntax of a representation generates a set of rules for combining symbols to form expressions in the represented language. It should be possible to tell whether or not an expression is well formed, that is, whether or not it could have been generated by the rules. A common syntax used in artificial intelligence is a *predicate-argument* construction of the form:

<sentence> ::= <predicate> (<argument>, ... I <argument>)

in which a k-place predicate is followed by k arguments.

5. Object-Oriented Programming and Knowledge Representation

Knowledge representation and object-orientation are each based on a common perspective, namely that it is at least as important to model the *domain* of an application as it is to model the problem we want to solve. If we are working in a particular domain whether it is engineering, publishing, or command and control, the problems to be solved will change not only between projects, but also during the course of a single project, due to the redefinition and refinement of goals and concepts. What will remain relatively constant are the inhabitants of that domain, whether they are machines, processes, inanimate objects, or people. Fortunately, assuming that representations are constructed with reusability in mind, representations designed for use by machines are likely to find multiple applications in a variety of projects.

What distinguishes knowledge representation from object-orientation generally is that knowledge representation attempts to reflect a domain expert's *knowledge* in addition to other domain relevant entities. For example, different ways in which they might be viewed, ordered, categorized, and manipulated in the performance of various tasks. Seen in this light, the knowledge representation question then becomes: Is it possible to codify knowledge about a domain in a manner that supports multiple applications of that knowledge to different problems in the context of different projects?

The most common language used in Expert Systems is the CLIPS language [Ref.105], which combines the OO and rule-based programming methods. Building a rule-based system in such a language is a non-trivial exercise best left to experts.

CLIPS is chosen as the main vehicle for illustrating rule-based systems (Section E) for the following reasons:

214

- Is relatively cheap;

- Borrows features from other successful tools;

- Has a fairly standard (albeit LISP-like) syntax.

- Is reasonably efficient, so that programs run in a finite time;

- Is reasonably flexible, e.g., it allows foreign function calls;

- Provides some (limited) facilities for combining rules with objects. [Ref.105]

On the hand the philosophy and techniques of object-oriented programming have a great deal to offer the designers of expert systems:

- "The philosophy of representing our knowledge of the world in terms of interacting objects and agents provides an appropriate framework for many classes of problem, particularly those (like planning and scheduling) which have a strong simulation component". [Ref.105]

- The techniques of procedure and data abstraction encourage AI programmers to think about the kinds of object and behavior that are relevant to the problem, instead of "becoming engrossed in the implementation of functions and data objects at too early a stage in the design" [Ref.105].

- There is a growing literature on object-oriented analysis and design which is relevant to the design of expert systems modules.

215

It is interesting to note that the limitations of object-oriented technology have become apparent in recent years. Because objects are still primarily computational devices, literal interpretation of these objects as faithful representations of real-world objects can be both confusing and restricting. "Inheritance of behaviors can lead to implementation problems" [Ref.105], some new object-oriented schemes only allow inheritance of interfaces.

In a perfect world, one would be able to develop systems, especially expert systems, incrementally by simply adding code. But this is not always possible, even within the object oriented framework, because adding new modules of knowledge often has unanticipated consequences, such as rules in competition with each other or ambiguous patterns of inheritance.

Thus it is obvious that object-oriented methods do not solve all our problems. These methods still leave the expert system designer with many difficult decisions to make. However, once the design decisions are made, the object-centered approach makes design decision easier and facilitates implementation " [Ref.105].

C.   SYMBOLIC COMPUTATION

1.   Symbolic Representation

Symbol is the notion that forms the main link between artificial intelligence and formal systems of logic and mathematics. In its most simple definition, a symbol is something that stands for something else. This "something else" is usually called the *designation* of the symbol, and may be a physical object or a concept.

216

The idea behind symbolic computation is that we want to allow symbols to stand for anything. Programming languages based on this paradigm provide a number of primitive data structures for associating symbols with other symbols, as well as primitive operations for manipulating symbols and their associated structures. A programmer must then specify two types of rules, syntactic and transformation. *Syntactic rules* form symbol structures out of symbols in such a way that the meanings of the resulting structures are generated by their individual components. *Transformation rules* turn symbol structures into other symbol structures.

Typically a symbolic program takes as its input one or more symbol structures representing the initial state of some problem, and returns as its output a symbol structure, representing a terminal state or solution. This symbol structure should be well-formed according to the given the syntactic rules and is derived by the application of feasible transformations. Programs in such languages are themselves symbol structures. Thus, there is no reason why some programs cannot treat other programs as data, as uniformity in the representation of data and programs makes itself particularly useful in the context of artificial intelligence. Moreover we can do more than merely write down rules for manipulating symbols; we can embody these symbols, and the rules for manipulating them, in some physical device. This leads to the physical symbol system, a very simple but revolutionary idea.

2.   Physical Symbol Systems

A physical symbol system is described as a machine with the following components::

- A memory containing symbol structures, which can vary in number and content over time;

- A set of *operators* for manipulating symbol structures, for example reading, writing, and copying;

- A *control* for the continual interpretation of whatever symbol structure is currently active;

- An *input* from its environment via receptors, and an *output* to that environment via some motor component. [Ref.117]

A program in a physical symbol system is just "a symbol structure that is interpreted or *evaluated* in some manner that is a function of its constituent symbols and its symbolic input" [Ref.105]. The primitive programs correspond to operators for manipulating symbol structures; more complex programs describe processes composed of these operators. For the operators, control over the use symbol structures lies in the ability to distinguish the difference between data, which is just returned, and programs, which must be interpreted.

A physical symbol system resembles a general-purpose computer equipped with symbol processing software. We know that stored program computers are universal machines (roughly speaking, they can simulate the operation of any other machine), and therefore are capable of computing all general recursive functions (roughly, all functions computable by any machine). It is the apparent power of these physical realizations of symbol systems that has encouraged researchers to suppose that such systems are capable of intelligence.

D.   KNOWLEDGE ACQUISITION

In Section A, we cited Buchanan's definition of *knowledge acquisition as* the "transfer and transformation of potential problem solving expertise from some knowledge source to a program" [Ref.106]. Knowledge acquisition is a generic term, as it is neutral with respect to how the transfer of knowledge is achieved. For example, the transfer could be achieved by a computer program that, after processing a large number of case studies, learns to associate symptoms with diagnoses categories.

The term *knowledge elicitation,* on the other hand, often implies that the transfer is a result of a series of interviews between a domain expert and a knowledge engineer, who then writes a computer program representing the knowledge. The use of such programs is advantageous because it is less labor intensive, and can in one step accomplish the transfer of knowledge from the expert to a prototype. However, the term could also be applied to the interaction between an expert and program whose purpose is:

- Eliciting knowledge from experts in a systematic way (e.g., presenting them with sample problems and asking for solutions);

- Storing knowledge obtained from the interaction using an intermediate representation; and

- Compiling the knowledge from the intermediate representation into a runnable form, such as production rules .

In this section, Subsection 1 suggests ways in which knowledge acquisition can be broken down into different stages of activity or levels of analysis. Subsection 2

reviews some early work on automated knowledge elicitation, which focused on the syntax of rules.

1. Theoretical analyses of knowledge acquisition

We mentioned in Section A that knowledge elicitation interviews generate between two and five 'production rule equivalents' per day. The reasons why productivity is so poor include the following:

- The technical nature of specialist fields requires the non-specialist knowledge engineer to learn something about the domain before communication can be productive;

- The fact that experts tend to think less in terms of general principles and more in terms of typical objects and commonly occurring events [Ref.105]; and

- The challenge of developing effective notation for expressing domain knowledge, and a finding good framework for uniting all of the various pieces.

As with any difficult task, it is beneficial to try to break the process of knowledge acquisition down into subtasks that are easier to understand and simpler to carry out.

2. Stages of knowledge acquisition

[Ref.106] offers an analysis of knowledge acquisition in terms of a process model of how to construct an expert system (see Figure A.1), the summary of these stages is:

- ***Identification.*** Identify the class of problems the system will be expected
  to solve, including the data that the system will work with, and the criteria
  that solutions must meet. Identify the resources available for the project,
  in terms of expertise, manpower, time constraints, computing facilities and
  money.

Figure A.1: Stages of knowledge acquisition.

- ***Conceptualization.*** Uncover the key concepts and the relationships
  between them. This should include a characterization of the different
  kinds of data, the flow of information and the underlying structure of the
  domain, in terms of causal, spatio-temporal, or part-whole relationships,
  and so on [Ref.106].

- ***Formalization.*** Try to understand the nature of the underlying search
  space, and the character of the search to be conducted. Important issues

221

include the certainty and completeness of the information and other constraints upon the logical interpretation of the data, (i.e., time dependency, and the reliability and consistency of different data sources). [Ref.106].

- ***Implementation.*** Express rules in an executable form under a chosen control regime. Decisions must be made about the structures of data and the degree of independence between different modules of the program. In turning a formalization of knowledge into a runnable program, one is primarily concerned with the specification of control and the details of information flow.

- ***Testing.*** The evaluation of expert systems is not an exact science, but it is clear that achieving the task is facilitated by running the program on a large and representative sample of test cases. Common sources of error are rules (either missing, incomplete or incorrect), and competition between related rules that can cause unexpected bugs.

As Figure A.1 suggests, the primary consideration in designing an expert system is the class of problems that we want the system to solve. Beginning with either a particular conceptual analysis of the domain or with a particular organization of knowledge in mind is a mistake, because one the way in which we represent concepts to ourselves and organize our ideas depends largely upon our own needs and purposes.

3. Different Levels In the Analysis of Knowledge

The distinction drawn between identification, conceptualization, and formalization can be found in the work of Wielinga and Schneider [Ref.118] who have developed a modeling approach to knowledge engineering within a framework called KADS. The authors argue that "a knowledge-based system is not a container filled with knowledge extracted from an expert but an 'operational model' that exhibits some desired behavior and impacts real-world phenomena" [Ref.118]. Knowledge acquisition involves not just eliciting domain knowledge but also interpreting that data using a conceptual framework, and formalizing the conceptualizations so that a program can actually use the knowledge.

The KADS framework is founded on five basic principles, as follows:

1. "The introduction of multiple models as a means to cope with the complexity of the knowledge engineering process" [Ref.118].

2. The KADS four-layer framework for modeling the required expertise.

3. The reusability of generic model components as templates supporting top-down knowledge acquisition [Ref.118].

4. The process of differentiating simple models into more complex ones.

5. The importance of structure-preserving transformation of models of expertise into design and implementation.

The motivation behind the KADS framework is primarily the management of complexity [Ref.118]. Today's knowledge engineer is faced with a large space of methods, techniques and tools which could be used to build an expert system. However, he or she is also faced with three fixed issues, namely: (1) *Defining the problem* that the

expert system is meant to solve, (2) *Defining the function* that the expert system will fulfill with respect to that problem, and (3) *Defining the tasks* that must be performed in order to fulfill that function.

The first principle of KADS is that "a framework should provide multiple partial models to help answer these questions" [Ref.118], for example:

- *"An organizational model* of a socio-economic environment in which the system will operate, (e.g., financial services, health care)

- *An application model* of the problem to be solved and the task to be accomplished.

- A *task model* which shows how the task is completed by breaking the desired behaviors into component tasks, (e.g., gathering income data, and generating disease hypotheses).

The original KADS approach breaks the 'conceptualization stage' down into two parts: a *model of cooperation or communication* and a *model of expertise*. The former is responsible for decomposing the problem solving behavior into primitive tasks and then distributing these tasks across agents, whether human or mechanical. The latter corresponds to what is usually understood by the term "knowledge elicitation," primarily an analysis of the different kinds of knowledge that an expert brings to the problem solving process"

Finally, there is the *design model* that suggests computational techniques and representational mechanisms that could be used to realize the specification derived from the previous models.

4. Ontological analysis

Another knowledge-level analysis for expert problem solving is called *ontological analysis*. This approach describes systems in terms of entities, relations between them, and transformations between entities that occur during the performance of some task. Ontological analysis may seem rather abstract, but is valuable because of its tendencies to structure a rather unstructured task   There are three main categories for structuring domain knowledge:

1. *The static ontology,* consisting of domain entities and their properties and relations;

2. *The dynamic ontology,* defining the states that occur in problem solving, and the ways to transform one state into another;

3. *The epistemic ontology,* describing the knowledge that guides and constrains state transformations" [Ref.118].

Ontological analysis assumes that the problem in question can be reduced to a search problem, but unlike other types of analyses, ontological analysis does revolve around the method used to perform a search.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Hurson A., Bright M, and Pakzad H., *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, 1994.

2. Heimbigner D. and McLeod D., *A Federated Architecture for Information Systems*, 1985.

3. Elmagarmid A., Rusinkiewicz M., and Sheth A., *Management of Heterogeneous and Autonomous Database Systems*, 1999.

4. Sheth A. and Larson J., *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, 1990.

5. Spaccapietra S. and Parent C., *View Integration: A Step Forward in Solving Structural Conflicts*, 1994 .

6. Litwin W., *Multidatabase Systems*, 1994.

7. Breitbart Y., Litwin W., Roussopoulos N., and Wiederhold G., *Final report of the workshop on multidatabases and semantic interoperability*, 1990.

8. Chawathe S., Garcia-Molina H., Hammer J., Ireland K., and Widom J, *The Tsimmis Project: Integration of Heterogeneous Information Sources*, 1996.

9. Bobak A., *Distributed and Multidatabase Systems*, 1996.

10. Ozsu T. and Valduriez P., *Principles of Distributed Database Systems*, 1999.

11. Elmasri R. and Navathe S., *Fundamentals of Database Systems*, 1994.

12. Bright M. W., Hurson A. and Pakzad S., *A Taxonomy and Current Issues in Multidatabase Systems*, 1992.

13. Cardenas A., *Heterogeneous Distributed Database Management*, 1987.

14. Staniszkis W. et al., *Architecture of the Network Data Management Systems*, 1985.

15. Hammer M. and McLeod D., *On Database Management Architecture*, 1979.

16. Heimbigner D. and McLeod D., *A Federated Architecture for Information Management*, 1985.

17. Litwin W., *From Database Systems to Multidatabase Systems: Why and How*, 1998.

18. Heimbigner D., *A Federated Architecture for Software Management*, 1987.

19. Hurson A. and Bright M., *Multidatabase Systems: An Advanced Solution for Global Information Sharing*, 1994.

20. Breitbart Y., *Multidatabase Interoperability*, 1990.

21. Litwin W. and Boudenant J., *SIRIUS Systems for Distributed Data Management*, 1986.

22. Hammer M. and McLeod D., *On Database Management Architecture*, 1980.

23. Litwin W. et al., *SIRIUS Systems for Distributed Data Management*, 1982.

24. Litwin W. Abdellatif A., *Multidatabase Interoperability*, 1986.

25. Litwin W. et al., *MSQL: A Multidatabase Language*, 1987.

26. Templeton M. et al., *Mermaid: A Front-end to Distributed-Heterogeneous Databases*, 1987.

27. Bellcastro E., *DQS: Distributed Query System*, 1987.

28. Chawathe S. et al., *The TSIMMIS Project: Integration of Heterogeneous Information Sources*, 1988.

29. Wiederhold G., *Mediators in the Architecture of Future Information Systems*, 1992.

30. Garcia-Molina H. et al., *Capability Based Mediation in TSIMMIS*, 1997.

31. Carey. M. J. et al., *Towards Heterogeneous Multimedia Information Systems: The Garlic Approach*, 1998.

32. Haas L. M. et al., *Transforming Heterogeneous Data with Database Middleware: Beyond Integration*, 1998.

33. Roth M. and Schwarz P., *A Wrapper Architecture for Legacy Data Sources*, 1997.

34. Tomasic A. et al., *The Distributed Information Search Component DISCO and the World Wide Web*, 1997.

35. Tomasic A, Raschid L., and Valduriez P., *Scaling Heterogeneous Databases and the Design of DISCO*, 1997.

36. Krisnamurthy R., Litwin W., and Kent W., *Language Features for Interoperability of Databases with Schematic Discrepancies*, 1991.

37. Naacke H., Gardarin G., and Tomasic A., *Leveraging Mediator Cost Models with Heterogeneous Data Sources*, 1998.

38. Tomasic A., Raschid L., Member, IEEE, and Valduriez P., *Scaling Access to Heterogeneous Data Sources with DISCO*, 1998.

39. Graefe G., *Query Evaluation Techniques for Large Information Sources*, 1993.

40. Ozsu T., and Valduriez P., *Principles of Distributed Database Systems*, 1999.

41. Genesereth M., Keller A., and Duschka O., *Infomaster: An Information Integration System*, 1998.

42. Genesereth M. and Duschka O., *Infomaster: An Information Integration Tool*, 1999.

43. Geddis D., Genesereth M., and Keller A., *Infomaster and Information Integration*, 1999.

44. Wood W., *Foundations for Semantic Networks*, 1985.

45. Rusinkiewicz M. and Sheth A., *Management of Heterogeneous and Autonomous Database Systems*, 1999.

46. Kashyap V. and Sheth A., *So Far Semantically and So Near Semantically*, 1993.

47. Ouksel A. and Naiman C., *Coordination Context Building in Heterogeneous Information Systems*, 1993.

48. Sciore E. and Siegel M., *Context Interchange Using Meta-Attributes*, 1992.

49. Yu W., Sun W. and Dao S., *Determining Relationships Among Attributes for Interoperability of Multidatabase Systems*, 1991.

50. Sheth A. and Kashyap V., *Semantic and Schematic Similarities Between Database Objects*, 1994.

51. Gruber T., *A Translation Approach to Portable Ontology Specifications*, 1993.

52. Elmagarmid A, Rusinkiewicz M, and Sheth A., *Management of Heterogeneous and Autonomous Database Systems*, 1999.

53. Larson J., Navathe S., and El-Masri R., *A Theory of Attribute Equivalence and its Applications to Schema Integration*, 1989.

54. Kent W., *The Many Forms of a Single Fact*, 1989.

55. Batini C., Lenzerini M., and Navathe S., *A Comparative Analysis of Methodologies for Database Schema Integration*, 1986.

56. Fang D., Hammer J., and McLeod D., *Remote-Exchange: An approach to controlled sharing among autonomous, heterogeneous database systems*, 1991.

57. Bertino E., Pelagatti G. and Sbattella L., *An Object-oriented Approach to the Interconnection of Heterogeneous Databases*, 1989.

58. Linnemann V. et al., *Design and Implementation of an Extensible Database Management System Supporting User Defined Data Types and Functions*, 1988.

59. Connors T, and Lyngbeak P., *Providing Uniform Access to Heterogeneous Information Bases*, 1988.

60. Dayal U. and Hwang H., *View Definition and Generalization for Database Integration in Multidatabase Systems*, 1984.

61. Sheth A. and Larson J., *Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases*, 1990.

62. Papazoglou M., Laufman S., and Sellis T., *An Organizational Framework for Cooperating Intelligent Information Systems*, 1992.

63. Ram S. and Barkmeyer E., *The Unifying Semantic Model for Accessing Multiple Heterogeneous Databases in a Manufacturing Environment*, 1991.

64. Ramesh V. and Ram S., *A Methodology for Interschema Relationship Identification in Heterogeneous Databases*, 1995.

65. Sheth A. and Kashyap R., *So Far Semantically, Yet So Near Semantically*, 1993.

66. Navathe S. and Gadgil J., *A Methodology for View Integration in Logical Database Design*, 1982.

67. Batini C. and Lenzerini M., *A Methodology for Data Schema Integration in the Entity-Relationship Model*, 1984.

68. Biskup J., and Convent B., *A Formal View Integration Method*, 1986.

69. Navathe S., El-Masri R., and Larson J., *Integrating User Views in Database Design*, 1986.

70. Shoval P. and Zohn S., *Binary-Relationship Integration Methodology*, 1991.

232

71. Gotthard W., Lockemann P., and Nuefeld A., *System-guided View Integration for Object-oriented Databases*, 1992.

72. El-Masri R., Larson J., and Navathe S., *Schema Integration Algorithms for Federated Databases and Logical Database Design*, 1986.

73. El-Masri R., Larson J., and Navathe S., *A Theory of Attribute Equivalence and its Applications to Schema Integration*, 1989.

74. Spaccapietra S. and Parent C., *View Integration*, 1994.

75. Kaul M. and Drosten K., *Viewsystem: Integrating Heterogeneous Information Bases by Object-oriented Views*, 1990.

76. Ahmed R., De Smedth P., and Du W., *The Pegasus Heterogeneous Multidatabase System*, 1991.

77. Bertino E., *Integration of Heterogeneous Data Repositories by Using Object-oriented Views*, 1991.

78. Kim W. and Seo J., *Classifying Schematic and Data Heterogeneity in Multidatabase Systems*, 1991.

79. DeMichiel L., *An Approach to Performing Relational Operations over Mismatched Domains*, 1989.

80. Chatterjee A. and Segev A., *A Probabilistic Approach to Information Retrieval in Heterogeneous Databases*, 1991.

81. Prabhakar S. et al., *Instance-level Integration in Federated Autonomous Databases*, 1993.

82. Ramesh V. and Ram S., *A Methodology for Interschema Relationship Identification*, 1995.

83. Deen A., Amin R. , and Taylor M., *Implementation for a Prototype for Preci**, 1987.

84. Templeton M., Brill D., and Dao K., *A Front-end to Distributed Heterogeneous Databases*, 1987.

85. Chung C., *Dataplex*, 1990.

86. El-Masri R., Larson J., and Navathe S., *A Theory of Attribute Equivalence and its Applications to Schema Integration*, 1989.

87. Sheth A. and Gala S., *On Automatic Reasoning for Schema Integration*, 1993.

88. Czejdo B. and Taylor M., *Integration of Database Systems Using OO Approach*, 1991.

89. Geller et al., *Algorithms for Structural Schema Integration*, 1992.

90. Thieme C. and Siebes A., *Schema Integration in OO Database*, 1993.

91. Shekhar S. et al., *Learning Transformation Rules for Semantic Query Optimization*, 1993.

92. Ramesh V. and Ram S., *Integrity Constraints Integration in Heterogeneous Databases*, 1997.

93. Whang W. and Navathe S., *Logic-based Approach for Realizing a Federated Information System*, 1991.

94. Bouzeghoub M. and Wattiau I., *View Integration by Semantic Unification and Transformation of Data Structures*, 1990.

95. Whang W. and Navathe S., *Logic Based Approach for Realizing a Federated Information System*, 1991.

96. Johannesson P., *Schema Transformation as a Aid in View Integration*, 1993.

97. Sheth A., *Issues in Schema Integration*, 1991.

98. DeSouza J., *A Schema Integration System*, 1986.

99. Sheth A. and Larson J., *A Tool for Integrating Conceptual Schemata and User Views*, 1988.

100. Sheth A. and Marcus H., *Schema Analysis and Integration: Methodology, Techniques and Prototype Toolkit*, 1992.

101. Lee J. and Baik D., *SemQL: A Semantic Query Language for Multidatabase Systems*, 1999.

102. Miller G., *WordNet - a Lexical Database for English*, 1998.

103. Miller G. and Beckwith R., *Five Papers on WordNet*, 1996.

104. Garcia-Solaco M. and Saltor F., *Semantic Heterogeneity in Multidatabase Systems*, 1996.

105. Jackson P., *Expert Systems*, 1999.

106. Buchanan B. G., Barstow D., *Constructing an Expert System*, 1983.

107. Feigenbaum E.A., *The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering*, 1977.

108. Davis R., King J., *An Overview of Production Systems*, 1977.

109. Findler N. V., *Associative Networks*, 1979.

110. Kowalski R.A., *Logic for Problem Solving*, 1979.

111. Davis R., *Meta-rules: Reasoning about Control*, 1980.

112. Schank R.C., Colby K., *Computer Models of Thought and Language*, 1973.

113. Anderson J. R., *Language, Memory and Thought*, 1976.

114. Newel A., Simon H.A., *Human Problem Solving*, 1972.

115. Winston P.H., *Artificial Intelligence*, 1984.

116. Winston P.H., *Artificial Intelligence*, 1992.

117. Newel A., *Knowledge Level*, 1982.

118. Wielinga B. J. Schreiber A. Th., *KADS: A Modeling Approach to Knowledge Engineering*, 1992.

119. Wielinga B. J., Breuker A. J., *Models of Expertise*, 1986.

120. Alexander J. H. Freiling M. J., *Knowledge Level Engineering: Ontological Analysis*, 1986.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

No. copies

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Naval Postgraduate School
   Monterey, CA 93943-5101

4. Prof. James Bret Michael 1
   Computer Science Department Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

5. Prof. Thomas Wu 1
   Computer Science Department Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5000

6. Deniz Kuvvetleri Komutanligi..1
   Personel Daire Baskanligi
   Bakanliklar
   Ankara, TURKEY