1986

Heterogeneous Distributed Data Base Management Systems

Stephen M. Deal

Rochester Institute of Technology

School of Computer Science and Technology

Heterogeneous Distributed Data Base Management Systems

by

Stephen M. Deal

A thesis, submitted to
the Faculty of the School of Computer Science and Technology,
in partial fulfillment of the requirements for the degree
of
Master of Science in Computer Science

Approved by: 9/9/86

Professor Jeffrey A. Lasky, committee chairman

9/9/86

Professor James E. Heliotis

9/10/86

Professor Chris Comte

I wish to be contacted whenever reproductions of this document are requested. I can be contacted at the following address.

11/25/86

Stephen M. Deal Date

17 Authors Avenue
Henrietta, New York 14467

ABSTRACT

This work analyzes the design and implementation issues of Heterogeneous Distributed Data Base Management Systems (HD-DBMS). To date, HD-DBMS research projects and implementations have been limited. The few such systems which have been constructed provide valuable insight into the nature of problems faced due to heterogeneity. Some of these systems (SIRIUS-DELTA, MULTIBASE, AIDA), are presented in order to examine their solutions to the problems.

The major issues described in the thesis are: the architecture of the distributed system; query translation; schema mapping; and integration of the schemata within the heterogeneous distributed database. In seeking solutions to these issues, an architecture for a HD-DBMS is proposed.

DEDICATION

For my wife, Julia Deal,
her companionship, encouragement and love
sustained me.

ACKNOWLEDGEMENTS

I am indebted to my family, who accepted my absenteeism, supported my efforts and urged me to reach my goal. I want to thank my thesis advisor, Jeff Lasky, and my thesis committee for their contributions. I also wish to thank the Eastman Kodak Company for their support.

Lastly I want to thank the two felines, Wandou and Hairy, whose friendship, affection and indifference put things in their proper perspective and provided a balance to it all.

Table of Contents

2. Evolution of the Technology

3. Heterogeneous Issues

APPENDICES

List of Figures

CHAPTER 1

OVERVIEW OF HETEROGENEOUS DISTRIBUTED DBMS

1.1. Introduction

Databases are proliferating within corporations, research laboratories and other organizations. These databases are used for many different applications including those in the financial, business, scientific and engineering fields. Integration of information resources permits activities to be coordinated more efficiently and precisely. One of the tools required to support this integration is a Heterogeneous Distributed Data Base Management System (HD-DBMS).

By a distributed database, we mean

> A distributed database is a collection of data which are distributed over different computers of a computer network. Each site of the network has autonomous processing capability and can perform local applications. Each site also participates in the execution of at least one global application, which requires accessing data at several sites using a communication subsystem.[1]

[1][CERI80] Page 6.

Creation of databases without regard to data integration and advances in data base management technology have produced dissimilar, yet related, information resources. Most of the existing Data Base Management Systems (DBMSs) were installed with little thought given toward future integration. They were designed and implemented for a specific purpose, usually at a single physical location. Older data models (hierarchical or network) are common in many organizations. In recent years the relational model has become more visible and has gained acceptance in the information management area.

At one point in time an organization may have a homogeneous information management system, but given the evolution of information technology, the system may eventually become heterogeneous. As use of databases continues to grow, organizations are feeling the need to eliminate the boundaries between databases and DBMSs. The desire of corporations to develop a Computer Integrated Manufacturing (CIM) environment is a current example [APPL85].

While software systems for distributed databases do exist, only a few have been constructed that support pre-existing databases and permit the combining of different data models. Integrating existing DBMSs that are dissimilar in schema and/or data model to form a single global database produces a Heterogeneous Distributed Database.

1.2. Description of the Research

1.2.1. Research Objectives and Approach

The objectives of this thesis are to examine and discuss several key issues related to Heterogeneous Distributed Data Base Management Systems. The research has been partitioned into the following areas:

(1) Definition of some of the important issues involved in the design and implementation of Distributed DBMS. The issues chosen are those that identify significant differences between Homogeneous and Heterogeneous Distributed DBMS.

(2) Presentation of past or present distributed data base management systems that have been constructed. Discussed are system capabilities, degree of functionality and how they have addressed the critical issues.

(3) Examination of the discriminating issues specific to Heterogeneous Distributed DBMS. Solutions to these problems will be presented and analyzed.

(4) Based on the thesis research, an architecture for a comprehensive HD-DBMS will be proposed. It will be designed to incorporate pre-existing DBMSs and provide the integration necessary to support a Heterogeneous

Distributed Data Base Management System.

1.2.2. Limitations of the Project

This thesis discusses issues specific to Heterogeneous Distributed databases that support pre-existing databases and DBMSs. Emphasis is placed on query processing and integration of the schemata.

Areas outside the scope of this project are those problems that are common to both homogeneous and heterogeneous distributed data base management systems. These include the following: Update Propagation of replicated data, Concurrency Control methods, Distributed Recovery and Network Communications.

1.3. Background

The reader should be familiar with six key areas: System Architecture, Data Transparency and the Global Data Model, Data Dictionary Management, Query Processing, Distributed Transaction Management and Integration of the Schemata. An understanding of these is necessary to appreciate the design and functional scope of a HD-DBMS.

There are many terms and acronyms in this document that may be unfamiliar to the reader. In these instances the glossary should be referenced to clarify their meaning.

1.3.1. System Architecture

The structure of a distributed system determines the functionality and the capabilities that system will have. The Global system architecture of a distributed database may be centralized or distributed.

Centralized

A Centralized system would have a single node that supports the Distributed DBMS. This node would be accessed by the remote nodes using a communications network for submission of, and response to, the global queries. All distributed transactions would be processed at this single site. This node would manage the data dictionary, control transactions, translate the queries, map between the schemas and present the results to the users.

A disadvantage of this type of configuration is the vulnerability of the distributed system. If the central node is unavailable to the remote nodes, distributed database services cannot be provided to any of the nodes. This condition can occur when the central node has crashed or when the

network links are unavailable. In the latter case, the central node may be up and available for local users, but no distributed transactions can occur because the remote sites cannot be reached.

Distributed

In a Distributed system structure, each node is capable of processing global queries. Global queries are decomposed at the node of origin and subqueries are submitted directly to the appropriate nodes. Although the network may be partioned and nodes may be unavailable for remote queries, the global query may not require information from the unavailable sites or may be satisfied using replicated data at reachable sites.

The Distributed system structure is a more flexible and robust design for a D-DBMS. It permits the system to operate no matter how partitioned the network may become. As long as the data required by a query can be obtained, a distributed database system will still be supported.

A disadvantage of this type of design is its complexity. Permitting all of the nodes in the distributed system to perform the processing required means redundancy of code and of data. Each node must be capable of managing the transactions for the subqueries it generates and receives. It must also be able to translate the global query and optimize its

execution. This design also increases the problems associated with concurrency control because of the decentralized query management. Updates to replicated data must be propagated throughout the distributed database.

For the purpose of this paper, it will be assumed that a distributed structure will be used to implement the distributed database.

1.3.2. Data Transparency and the Global Data Model

When data is distributed between separate systems and stored in dissimilar database schemas, it is necessary to design an architecture that makes these differences transparent to the user and application programmer.

The architecture would be layered to provide support for the global schema of the D-DBMS, the logical fragmentation of the global schema and the allocation of the fragments to each site. The architecture must also permit mapping between the different schemas and data models contained within the distributed DBMS. Among the three basic data models, Hierarchical, Network (CODASYL) and Relational, the Relational Model [CODD70] has been used for this global structure most frequently.

An architecture that supports data transparency is a critical issue for a HD-DBMS. One type of structure might have each site's DBMS support distributed operations using it's local Data Manipulation Language (DML). This begins to become complex for a variety of reasons. First of all, each site would have to be aware of all the other sites in the distributed database. This implies that a fully replicated data dictionary must be maintained at each site. It would also mean that each site must be capable of translating queries to and from every other site in either the global or a local DML. At this point, modifications to local DBMSs would become necessary, violating the rule of site autonomy. As this architecture is defined further, it becomes obvious that the expandability and maintenance of the system would become difficult.

To support a heterogeneous distributed DBMS, it is advantageous to represent the data using a global database. This database would use a common, or global, data model to represent the global schema of the distributed database. The global data model and its corresponding global data manipulation language, provide a unifying mechanism for the heterogeneous sites. A single image system for users to work with, provides the degree of transparency necessary for a HD-DBMS.

The global data model and its DML must be sufficiently flexible to support both the distributed system and its

heterogeneous characteristics. Data representations and data manipulation commands from each of the local DBMSs are mapped to the global database. These mappings and translations only occur between the local databases and the global database to minimize the quantity and type of translations necessary to support the HD-DBMS. In order to provide this type of support, the data model should have two basic qualities. One is the ability to support simple data types to represent any type of global data value. The other is the capacity to represent the global, fragmentation and allocation schema. Providing simple data structures, for example binary, permits the construction of representations of more complex data structures.

As mentioned earlier, the structure of a distributed database would be layered. The top layer of a distributed database architecture is the global database schema. This schema is the basis for a users view of the distributed data. It effectively describes all of the data contained within the D-DBMS in a way similar to a centralized database. That is, the distribution and storage of the data is transparent to the user. The global schema consists of relations that represent subsets of the distributed database. At this layer the distribution of the data is most transparent to the users. Users and application programmers are able to work with the distributed database as if it were a non-distributed.

In the next layer, the relations defined in the global schema are broken down further into subsets called partitions or fragments. These fragments are disjoint sets of the data contained in one global relation. In many instances it is logical to fragment the relations by the location of the data. For example, all of the data for a particular branch office of a company may be one fragment of a relation. Fragmentation of the relations requires that a mapping exist between the global relations and their fragments. This mapping is called the fragmentation schema and is always a one to many relationship (e.g. one global relation has many fragments).

The physical location of the fragments is defined by the allocation schema. It is this layer of the architecture that specifies where redundant data will exist in the HD-DBMS. When the same fragment is allocated to more than one site, it is said to be replicated. This layer of the global architecture is dependent on the local sites. It entails mapping the fragmentation schema to the local DBMS schema. This is a specific area where a heterogeneous D-DBMS differs from a homogeneous one.

In a HD-DBMS, each site will require a local mapping schema. This schema performs two mapping functions. The first is between the global and local schemas. The second is between the global and local data models. It is the latter of these that makes the job difficult.

The selection of the common data model and data manipulation language is very important. Some of the questions that need to be resolved are:

Which data model will best support a HD-DBMS?

What data types are to be supported?

What capabilities should the global DML provide?

How much transparency is required from the Global DBMS?

1.3.3. Global Data Dictionary Management

The Global Data Dictionary (GDD) stores information that describes the distributed database. Contained in the Data Dictionary are the descriptions of the global, fragmentation and allocation schemas and mappings to local DBMS names. Access methods, statistical information and integrity constraints for the global and local systems are also stored here.

This information is used by the system to perform the various translations between schemas and data models. Some of these functions include:

(1) Translation - Using the data mappings defined between the application and the physical data to translate between global and local references.

(2) Optimization - by examining the fragmentation allocation of the data, the most efficient accessing method or path can be determined.

(3) Transaction Management - verifying that the optimal plan is valid, the user has adequate privileges and sites are capable of performing the transaction.

How the Data Dictionary (or Catalog) for the distributed system is structured is be determined by the architecture of the system and the patterns of usage.

The three basic forms the global data dictionary can take are Centralized, Partitioned and Replicated. The data dictionary could also consist of combinations of these forms.

Centralized A single site would manage the data dictionary. This simplifies the management of the dictionary and guarantees its accuracy. When a centralized system architecture exists, the same site would be used to manage the data dictionary. In this instance the global database most closely resembles a non-distributed database in structure.

Partitioned When the data dictionary is partitioned, each site in the distributed database maintains the portion of the data dictionary specific to that site. The entire dictionary is not maintained at any one site, yet could be produced by combining the partitions.

Each site might also contain information from the global data dictionary. For example, which database contains what information, each site's processing capabilities and other global database information.

Replicated With a replicated data dictionary, every node contained in the distributed database would retain a full copy of the entire global data dictionary. This forces the data dictionary to be maintained at all sites in the distributed database. When duplication of the data dictionary occurs, modification of the dictionary becomes more complex. Changes and modifications made to the data definition must be propagated throughout the distributed database system.

Combinations Each site would maintain the data dictionary for its own local data as well as that of data at remote sites. If frequent queries are made to certain sites, it is advantageous to store the data dictionary information locally. Queries can be formulated based on this information in advance of communicating with the remote site. Another combination uses a partitioned dictionary for each site and has a central site maintain the entire dictionary. Permitting combinations of data may be the best solution as far as flexibility of the system is concerned.

1.3.4. Query Processing

1.3.4.1. Introduction

Distributed database systems handle queries in a manner similar to that of centralized systems. Two basic operations, decomposition and optimization, occur during query processing in both centralized and distributed databases. An additional operation takes place with HD-DBMS, this is the translation of the query.

Query decomposition involves breaking the global query into subqueries that single sites are capable of processing. Query optimization is the process of developing an efficient access method in order to obtain the desired results of the query. Translation of the query is necessary to convert the query in global DML to the appropriate local DML.

In most implementations, a common global data manipulation language is defined to provide users with a consistent interface to the global schema. In a homogeneous D-DBMS this may be the data manipulation language native to all of the local DBMSs. For example the R* system from IBM uses an extended version of the SQL query language.

1.3.4.2. Global Query Decomposition

The global query is broken down to define subqueries for each of the sites. As in a centralized database, the global query must be analyzed to determine the best method of decomposition for optimum processing. Decomposing the query requires that the query be clearly defined to permit combining of common subexpressions and translation of the query from a global view down to a more site specific level.

After a global query is submitted, it is examined to determine what global relations are required. Then a binary tree is constructed to represent the query in algebraic form. This tree, using canonical expressions, defines the basic operators and operands required to produce the desired result. It may be modified or reduced using Relational Algebra equivalence transformations. These transformations primarily support small algebraic expressions.

Once the query has been reduced and common subexpressions have been combined, the global query is then transformed into fragment queries. A canonical expression of the global query is produced that operates on the fragments of the global relation. This expression, while usable as a query, is not an optimal execution procedure. It is at this point that query optimization methods are used to improve the efficiency of the query.

1.3.4.3. Global Query Optimization

Factors such as data location, site processing capabilities, site processing costs and network communication costs can all affect the efficiency of query processing. Various optimizing algorithms weigh these and other factors differently. Some will try to maximize parallelism for better response time while other methods strive to minimize network communications.

There are several problems to be considered when choosing an optimization strategy. Included are:

> The Materialization - this is the determination of the set of fragments the query uses.
>
> The Sequence - the order that the operations are executed on the materialization.
>
> The Method - the combining of operations at a site, this is heavily dependent on the capabilities of the local DBMS.

These three problems are all interrelated, but quite often optimization is simplified by solving each of them separately.

The strategies of optimization are different depending on the configuration of the system. Long haul networks such as ARPANET have limited bandwidth and communication costs are considered high, LANs on the other hand are usually privately owned and have a much larger bandwidth, keeping communication costs low. When communication costs are

reduced, site processing costs and capabilities are of greater concern. Yet another factor that must be taken into account is the communication delay between sites in a distributed system.

In either one of the configurations mentioned above, site processing capability is a major consideration. The processing speed, number of I/O operations required and local DBMS functionality all affect the decisions rendered during query optimization. Depending on the query strategy chosen, join and semi-join relational operations are used at strategic points to minimize transmission costs by reducing the amount of data transferred between sites. In a HD-DBMS the strategy used for query processing becomes dependent on the capabilities of each site. It may be desirable to use some sites to combine intermediate results, but not possible.

1.3.4.4. Query Translation

An operation that occurs during global query processing in a HD-DBMS, is the conversion from the global DML into the local DML. When and where this translation occurs affects the efficiency of the query. Conversion of the global query into a local DBMS (site) query may take place before or after the query is globally optimized. In some instances global optimization may be unnecessary if data from remote nodes is not required. Determination of when translation occurs depends on the query strategy used, the architecture of the HD-DBMS and the global data model.

Some of the issues involving query processing that need to be addressed are:

- How does a global data model support different schemas from different data models?
- When converting data between schemas, how can this be accomplished in an efficient and timely manner?
- When are queries or application programs translated between data manipulation languages?
- How are differences in the data manipulation languages resolved (i.e. joins aren't supported on IMS)?

These issues are discussed in greater detail in chapter 3.

1.3.5. Transaction Management

Transaction Management deals with several interrelated issues such as Concurrency Control, Integrity and Reliability of the query.

Controlling transactions on a distributed database system requires a distributed transaction manager. Its job is to guarantee that distributed transactions are executed reliably, efficiently and can occur concurrently. This implies the use of commit protocols, recovery methods and procedures, optimized access plans and concurrency control mechanisms. Since this occurs in a distributed environment, the transaction manager must be able to coordinate these interrelated tasks and possibly work with different DBMS systems.

Similar to transactions in a centralized database system, distributed transactions must be atomic. This means that in the case of a site or network crash, the transaction will either fully commit or abort. To do this requires that the manager provide transaction recovery support for distributed transactions. This helps ensure the integrity of the database when a transaction uses more than one resource.

One of the goals in terms of transaction management is to minimize the cost of CPU processing and response time to the

user and optimize the execution of the transactions. While query optimization will produce strategies based on the fragmentation and allocation schemas, the optimizer may not be aware of dynamic changes occurring at the sites or in the network. By using the optimization strategies produced by the optimizer and knowing the current status of the distributed system, the manager will determine a plan of execution.

In a multiuser distributed database system, several transactions will be executing concurrently. In many cases they will be using the same databases and will require access to the same data. Concurrency Control guarantees that when multiple transactions access the data, the results will be the same as if all the transactions had been executed serially by a single user.

The global transaction manager in a distributed system must contend with the local DBMSs when it wants to manage concurrency in the distributed database. In order to maintain site autonomy as stated in section 1.1, the global transaction manager may not acquire locks directly on the local DBMSs. Instead, each site has a process that acts as a remote transaction manager that operates similar to a local transaction. Its task is to communicate with the global transaction manager and perform local functions for the distributed transaction. In this manner, the global transac-

tion manager can coordinate the actions of the remote transaction managers without being concerned with the site dependent details. Global transactions that require synchronization of the local sites can be managed effectively in this manner.

1.3.6. Integration of Schema

In order to present a consolidated view of the information to the user, the data must be integrated. This requires that the individual representations of the information in all of the local schemata be merged to form a single global schema.

In a heterogeneous distributed database, chances are that the databases to be integrated are pre-existing. It is also likely in many organizations that the separate databases will contain related, if not overlapping data sets.

For example, there are two databases, engineering and purchasing, that are in the distributed database, and each uses a different data model. Both of these databases contain information about parts. The overlap occurs when the same part is referenced in both local databases. The engineering database describes part number, quantity and part material. The purchasing database describes the same three attributes,

but uses item number, quantity and vendor part number. Here we run into conflicts between the two schemata. The part number and item number are synonyms, two names for the same fact. The quantity attributes are homonyms, the same name is used for different purposes. Engineering quantity defines number of parts in the assembly, whereas purchasing quantity defines the number in an order. The units of measure in each of these definitions could also be in conflict. Lastly, the part material and vendor part number fields are both referencing the same attribute, but in a different manner. An example would be for a part material of "T6-6063 aluminum", the vendor part number is "Al6063-T6".

In some instances the local databases can be modified to agree in definition, but this violates the site autonomy rule. Instead an auxiliary database could be used to store attribute name or scale conversions, for mediation of structural and abstraction conflicts. MULTIBASE, is an implementation that uses an auxiliary database [LAND82]. Another implementation, AIDA [TEMP86], uses a process local to each site to perform the same function without using an auxiliary database.

In order to integrate the schemata in the HD-DBMS, these differences must be resolved. The conflicts can be classified as follows:

(1) Name conflicts, involving synonyms and homonyms.

(2) Scale conflict, different units for the same data.

(3) Structural conflicts, an entity in one schema is an attribute in another.

(4) Abstraction conflicts, different levels of detail for the same fact.

(5) Replication conflicts, inconsistencies between redundant data.

How these conflicts are resolved depends on the structure of the Heterogeneous Distributed Data Base Management System. Resolution of these conflicts by the HD-DBMS provides support for schema integration.

1.4. Thesis Outline

Up to this point the reader has been introduced to the subject and presented with a description of the area to be researched. The background information has defined some of the important issues and focused on those specific to Heterogeneous Distributed Data Base Management Systems. The remainder of this thesis will be structured as follows.

Chapter 2 will present project systems that were built to investigate the issues and problems involved in supporting a HD-DBMS. Each of the projects presented represents a different approach to solving the problems and a different degree of functionality.

Chapter 3 examines the issues that differentiate a heterogeneous D-DBMS from a homogeneous one. These include problems associated with schema mapping, query processing and integrating the schemata.

An architecture for a Heterogeneous Distributed Data Base Management System is proposed in Chapter 4. This system is intended to be comprehensive and will be designed to incorporate pre-existing databases into the global database. It will support data transparency and provide integration of the local database schemata for the global user.

Chapter 5 concludes the thesis by summarizing some of the

conclusions drawn and lessons that were learned. The feasibility of the proposed and existing systems is also discussed. Some speculation is made as to the future trends in the area of Heterogeneous Distributed Data Base Management Systems.

CHAPTER 2

EVOLUTION OF THE TECHNOLOGY

2.1. Introduction

Several projects have been undertaken to research and develop HD-DBMS. In many cases they have been borne from research environments and funded by government sponsors.

Distributed processing and data management has been a topic of interest for many years. As computer networks have become more developed, the feasibility of distributed DBMS has grown. Hardware and/or software vendors have either created an entirely new product or expanded an existing product to support a distributed DBMS. The few systems that have been produced commercially are in most instances homogeneous. Most of the following projects have taken a few years to yield results. They have usually produced prototype models in a research environment. Few, if any actual heterogeneous systems are available as commercial products.

2.2. SDD-1, An Early System

The Computer Corporation of America (CCA), Cambridge Massachusetts, has worked on a few projects involving D-DBMS. One of the most notable, SDD-1 (System for Distributed Databases) [ROTH80], was designed in 1976 and its development continued through 1980.

This was the first general-purpose distributed DBMS ever developed. Its purpose was to provide, functionally, the same capabilities of a non-distributed system. Upon completion of the prototype, the system supported distributed query processing, concurrency control and reliable writing.

SDD-1 made use of an existing DBMS, Datacomputer [CCA78], and its corresponding query language called Datalanguage, for global data management and manipulation. The system supports a relational data model and its language is most commonly embedded into host application programs similar to SQL or QUEL. The language can also be used interactively by end-users.

While SDD-1 did demonstrate distributed database capabilities, it was not designed to be implemented on top of pre-existing DBMS and did not address the heterogeneous issues.. The system had however defined some of the major issues involved in distributed systems and led the way for future

projects. Query optimization techniques developed in SDD-1 were later applied to the MULTIBASE and AIDA systems.

2.3. The SIRIUS Project

A French nationwide project, SIRIUS, was initiated by the government of France in June of 1976. Funded by the French Ministry of Industry and the Institut National de Recherche d'Informatique et d'Automatique (INRIA), the goal of SIRIUS was the research and development of distributed database management systems in a heterogeneous environment.

Some of the objectives for the project were to:

(1) Research the problems and propose solutions to the issues involved in distributed data management.

(2) Build prototypes to test the ideas and evaluate methods.

(3) Transfer the results of the research to industry for practical application (banking).

The first phase of this project involved development of theories and methods necessary to support distributed databases in different environments. Basic research was conducted and limited prototypes were constructed to gain knowledge in specific areas and to prove out ideas. In the

second phase, the SIRIUS-DELTA project, more complex prototypes were constructed in an effort to integrate results obtained from the first phase.

2.3.1. POLYPHEME

POLYPHEME, a joint project between the University of Grenoble and the CII-Honeywell Bull Scientific Center was one of these limited prototypes. Theoretical work had been done on data modeling, query processing and system architecture, but not all these factors were incorporated in this prototype.

The main features to be implemented were:

A design as a network of abstract relational machines.

A D-DBMS defined as a union of all the local relations.

An algebraic relational language was provided for on-line data definition and manipulation.

Support of distributed query decomposition and optimization.

A distributed execution monitor for asynchronous remote procedure activation (not a procedure call) over the network.

The design of POLYPHEME did not include mechanisms for distributed recovery in the event of site or network failure. Concurrency control was another issues that was not addressed during this phase of the project. Banking applications were used for testing its capabilities. The initial prototype ran on IRIS 80 computers, and were connected to

the CYCLADES network.

2.3.2. SIRIUS-DELTA

SIRIUS-DELTA was designed and built to test heterogeneity in distributed data base management systems. Its design began in 1977 and the final version of the prototype became operational in 1982. The prototype used not only different DBMSs, but different computers and operating systems as well.

SIRIUS-DELTA was designed to be an extremely flexible system. There are no site dependent activities outside of local DBMS functions. Each site is capable of processing a query and the system is fully distributed. The system uses a data flow approach and strives toward parallelism.

System Architecture The SIRIUS-DELTA system uses a two level architecture, global and local, to support the distributed system (Figure 2.1). The global level represents the distributed database and provides the appearance of a single unified database to a user manipulating global data. The local level is a centralized DBMS which processes both global and local queries and works directly with the physical data.

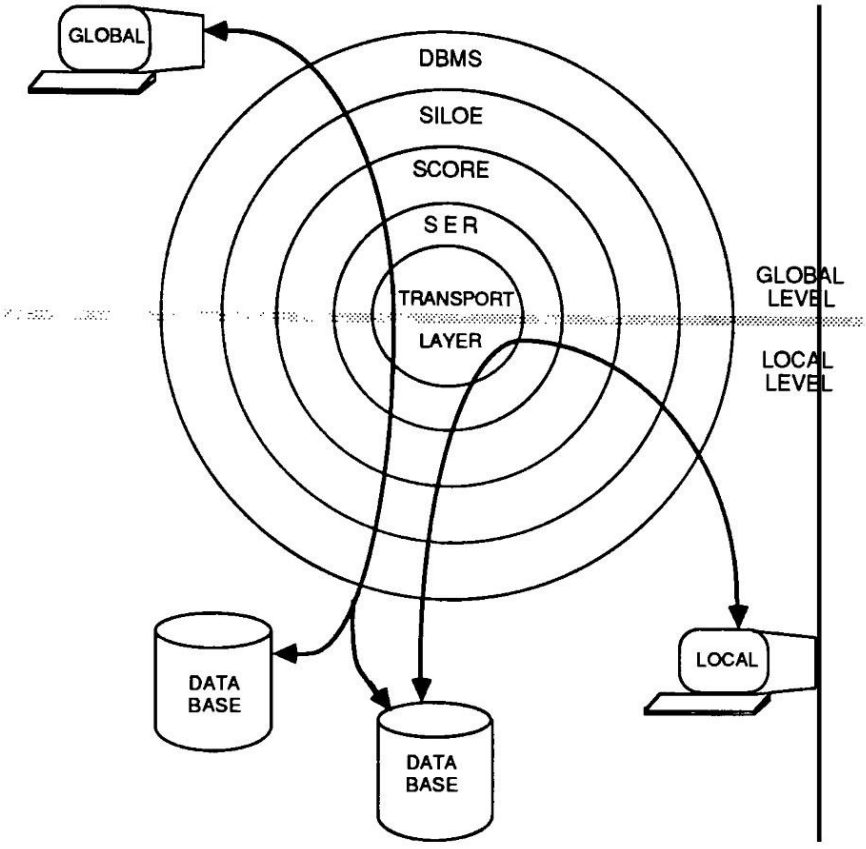


Figure 2.1 Architecture of the Components

The global and local levels use five layers that provide basic distributed capabilities.

These layers are as follows:

(1) Conventional (centralized) Data Management functions.

(2) Distributed Data Management functions.

(3) Distributed Control functions.

(4) Distributed Execution functions.

(5) Network Communications (transport).

SIRIUS-DELTA makes use of existing products to provide support for the conventional data management and transport layers. Modules were built to support the layers which are specific to distributed support.

The three layers are named as follows:

SILOE - for Distributed Data Management

SCORE - for Distributed Control

SER - for Distributed Execution

The first of these, SILOE, manages data distribution and query optimization. It exists on both the global and the local levels. On the global level it performs two func-

tions:

(1) Global internal schema management to define mappings between the global and local levels.

(2) Data materialization and generation of the distributed execution program.

The local module of SILOE is the interface from the local external schema to the global internal schema. It manages translation of the global subquery and supports local DBMS functions required by the global level.

The task for the SCORE subsystem is to control concurrency and reliability in the distributed database. This includes providing recovery facilities in the event of failure of a system component. This subsystem consists of three basic mechanisms:

(1) The Circulating Sequencer which is a decentralized synchronization mechanism.

(2) Protocols for locking and unlocking.

(3) The Two-Phase Commit protocol for operations which require writing.

The SER layer provides transaction management and communications support for distributed execution. The basic

functions it provides are:

(1) Remote process activation

(2) Process scheduling

(3) Process to process data transfers

(4) Use of parallelism as much as possible

(5) Failure and error detection of processes

These three subsystems provide facilities not available in conventional DBMSs. They are layered onto existing systems so as not to interfere with site autonomy. It is the addition of these subsystems which allows a local host DBMS to become part of the distributed database.

Data Transparency A global data model, called the PIVOT model, was chosen to support the distributed database. It is relational in nature and supports the External, Conceptual and Internal schemata at the global level. The architecture of the schemata in the SIRIUS-DELTA system is shown in Figure 2.2.

The External schema presents the distributed data to the user much like a centralized database. Entity and attribute names, relationships and other characteristics specific to a local database are not visible to the global user.

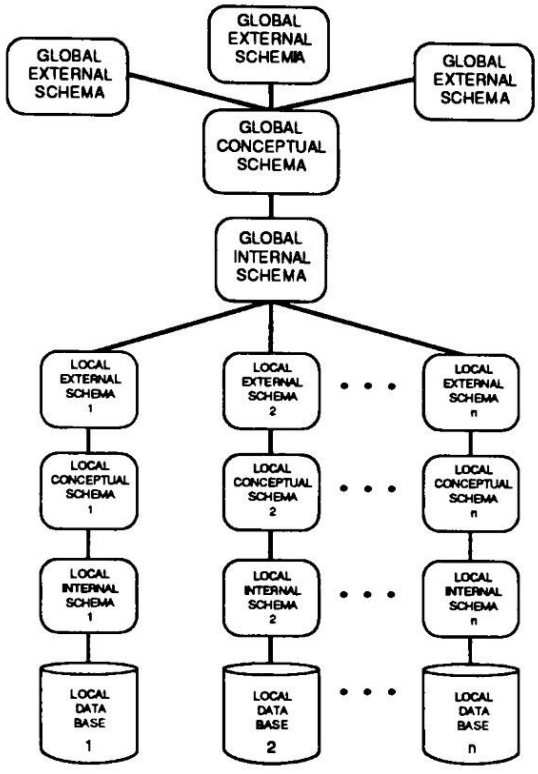


Figure 2.2 Architecture of the Schemata

The Conceptual schema defines the entire distributed database. It describes the entities (relations) their attributes and relationships. It could be represented by any data model, (hierarchical, CODASYL, relational).

The Internal Schema defines the data mappings of the global relations to the local external schemata. This schema is for construction of the global data which is dependent on the local data. It describes, for each site, the local data

set which is contained in the distributed (global) database.

The local level in the architecture is where the data actually resides. Users conducting local operations would interact directly with the local DBMS in the usual manner. Global operations are handled as if submitted locally. The three schemas, External, Conceptual and Internal, are also present at the local level of the architecture.

The Local External Schema provides the local user with a view of the local data, without concerning them with the data model structure or physical data location. The distributed database is not necessarily comprised of the entire database from each local site. Instead, the Local External Schema supports the specific data set required by the global database. This schema is used by the global level for mapping local data to the global level.

The Local Conceptual schema, using any of the aforementioned data models, defines the entire local database. The Internal schema defines the actual physical location of the data. These both function normally as if they were used only for the local DBMS.

Query Processing A global external language (the PIVOT language), called PETAL, was specified to perform global transactions with the PIVOT model. In following with the relational PIVOT model, PETAL uses a relational algebraic

language to manipulate sets of data. The algebraic language decomposes into operations on sets and relations easily and facilitates the interfacing to local DBMSs.

Queries directed toward the global database are submitted using the PETAL language. The global query is analyzed and decomposed into subqueries for each of the local sites. The subqueries are then translated, by local SILOE, from the PETAL language into the local external language. The local DBMS will then process the query in its normal manner.

Integration of the schema occurs at the global level. It is there that the local schema are mapped to the global schema and a distributed view of the data exists. Conflicts between the schema are defined and resolved at the global level of SIRIUS-DELTA.

The SIRIUS-DELTA project provided valuable insight into many of the problems faced in implementing a HD-DBMS. It presented researchers with practical experience in heterogeneous distributed systems. Areas that require trade-offs were identified and understood for future applications of a HD-DBMS.

2.4. MULTIBASE

In the early 1980's, using the knowledge gained from SDD-1, CCA began development on another project, MULTIBASE [LAND82]. This software system was designed to support user interaction with a heterogeneous distributed database. Updates of the data or synchronization of read operations are not supported by MULTIBASE.

Generality, compatibility and extensibility are three major objectives of the MULTIBASE system. To maintain the generality required, it has been designed to be a tool for integrated access to dissimilar DBMSs regardless of the application. This permits physically distributed data to be retrieved, logically related and presented to the user as a single view of the data. It has been designed to make data location, DBMS specific query languages and type of data model transparent to the user.

Compatibility is a requirement for interfacing with pre-existing DBMSs. This is to guarantee that no modifications will be needed to the existing databases or their application programs. Autonomy of the local systems is upheld and allows them to manage their own databases.

Adding future DBMSs to the distributed database requires that the system be extensible. It will support the three

known data models (hierarchical, network and relational) and has been designed in modules to permit easy adaptation.

System Architecture MULTIBASE uses a modular architecture to isolate those processes that deal with the distributed database as a whole, from the parts of the system that interact with the local host DBMSs (Figure 2.3). There are two basic modules used for this function, the Global Data Manager (GDM), and the Local Database Interface (LDI).

The Global Data Manager handles all of the global aspects of a query. To support global queries the GDM uses an Internal DBMS. This DBMS is a tool for the global DBMS. It stores intermediate results returned from the LDIs and supports an Auxiliary database. The Auxiliary database is used for schema integration and resolution of data incompatibilities.

Data Transparency MULTIBASE uses DAPLEX [SHIP81], a data definition and manipulation language, to define the schemata and to provide a uniform user interface for queries to the distributed database. It is with this language that the Global, Auxiliary and Local Schemata are defined.

While the entire distributed database is defined by the Global Schema, each local DBMS is described to MULTIBASE as a Local Schema. This local schema is the equivalent of the schema defined at the site. These definitions may be modi-

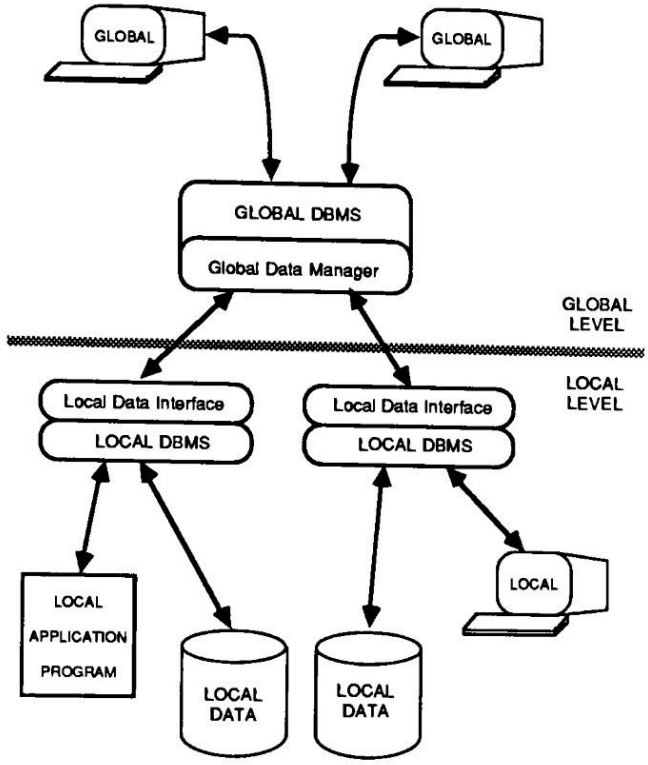


Figure 2.3 Architecture of the Components

fied to accommodate changes to the databases without restructuring the entire system. An Auxiliary schema is used by the GDM to structure the information used for resolving conflicts between schemata. Figure 2.4 shows the structure of the schemata in the MULTIBASE system.

Query Processing Global queries are submitted using the DAPLEX query language. The GDM performs a series of operations on the query before passing it to the LDI. First, the

Figure 2.4 Architecture of the Schemata

query is transformed, taking the global references and changing them into local and auxiliary schemata references. The query is then globally optimized to begin determining the query strategy. Once the optimizer has defined segments of the query, they are passed to the Decomposer. The decomposer separates the query by site producing DAPLEX single-site queries. When queries reference the auxiliary DBMS supported by MULTIBASE or require combining data from separate sites, they are collected for processing by the Internal DBMS. After decomposition, the queries are

filtered to remove operations that the local DBMSs cannot support. The definition of these operations is saved, the query rephrased and passed to the optimizer once more. When the optimizer finally releases the reduced DAPLEX query to the Monitor, it is in a form that can be processed by the local DBMS site. The Monitor is the global transaction manager. It directs the single-site queries to the proper site and activates the LDI to process the query. It may also send a final query to the Internal DBMS for processing the results returned from the LDIs.

At this point in query processing, the Local Database Interface is assigned control. Since the LDI is working with a single site, its task is not as complex as that of a GDM. Once it receives the DAPLEX query from the GDM, it first optimizes it. Since DAPLEX uses a "set-at-a-time" query language, local DBMSs that support this type of query can optimize it in a fairly straight-forward manner. If the local DBMS uses "record-at-a-time" processing, the query must be reformulated and this can become extremely complex. The query is then translated into the local DBMS query language and processed in the standard manner. Results returned from the query are formatted into a standard representation and passed back to the GDM.

A breadboard version of MULTIBASE has been designed and implemented. Some of the functions it provides are:

Uniform query access

Logical schema integration for non-overlapping data

Techniques for handling incompatible data

Global and Local Query optimizers

It demonstrated the ability to provide integrated access between hierarchical and CODASYL data models. A prototype version is intended to support standalone test and evaluation experiments for different applications. It would be implemented in ADA in hopes to keep the system portable.

2.5. AIDA

AIDA (Architecture for Integrated Data Access) [TEMP86] is a system designed to permit users to access one or more databases as a single (distributed) database utilizing a common query language. The system is being designed and implemented at System Development Corporation (SDC), Santa Monica, California.

The purpose of the research project that spawned the AIDA system is to explore the issues of distributed data management. One of the project goals has been to develop a prototype system that utilizes new technology developments in data management, networks, security and artificial

intelligence. A long term research goal is to develop a front end to distributed data, text, image, voice and knowledge objects.

The AIDA system has been designed as a front end to heterogeneous databases. It is an application system built to integrate pre-existing DBMSs and provide users with a single language to manipulate the resulting collection of distributed data. While AIDA is not a DBMS, it is capable of presenting data to a user in an integrated and transparent manner. The problem of translation between relational and CODASYL schemas has not been addressed by the system. Its development is directed primarily toward integrating more current DBMS technology (relational views) rather than older DBMS technology (hierarchical).

In order to support interaction with a database, AIDA requires that the DBMS provide a "relational" view of its data. This allows temporary relations to be created by AIDA within the database. Although updates will eventually be permitted, these will usually be small scale updates (a few tuples) and are not currently supported.

A goal of the system is to maintain autonomy of existing systems. For the majority of cases the local databases will be responsible for their own updates and control access to their data.

System Architecture The system was designed with a modular architecture to permit flexibility and facilitate combinations of workstations, processors and database machines (Figure 2.5).

A module in AIDA called Nexus, provides the user interface necessary to support either ARIEL (defined by SDC) or SQL as query languages. Nexus calls the Query and Schema Translator (QUEST) module to parse and translate the user query into the Distributed Intermediate Language (DIL). This processed query is then passed to the Procedural Interface which communicates it to a portion of the system known as

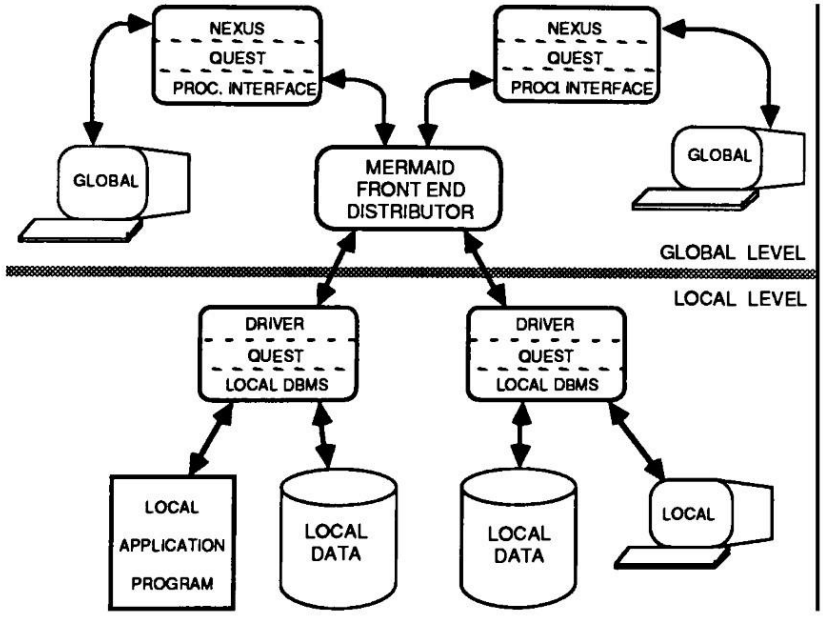


Figure 2.5 Architecture of the Components

Mermaid [TEMP83].

Query Processing It is the task of the Mermaid system to control the optimization and decomposition of the query. The subqueries are then transmitted to the appropriate DBMS driver and the Mermaid distributor waits for responses. Each DBMS site has a driver that processes the DIL subquery for the local system. Using the QUEST translaton, the DIL is transformed into the local DBMS query language and submitted. The response may be translated back into DIL and returned to the distributor, or forwarded to another DBMS driver. When data is transferred between sites it is translated into a common representation (ASCII) and then into the representation of the receiving site. Once the distributor receives the information back from the DBMS drivers, it must assemble the results of the subqueries. The unformatted results are sent to the Nexus user interface for formatting and display. Communication between sites uses a Local Area Network (LAN) that supports the Transmission Control and Internet Protocols (TCP/IP).

The Mermaid system was initially implemented on a single VAX and all databases were supported by a single database machine (1983). It was expanded to become the AIDA system in 1984 and currently runs on four computers. One VAX 11/780 w/Britton-Lee IDM database machine, two Sun Microsystems (120 and 170) running Ingres and a Sun 120 running Mistress.

They are all using UNIX and connected by Ethernet.

The AIDA prototype is being upgraded to become a product with a scheduled delivery date of 1987. Some of the modifications necessary for this to occur are:

(1) Support for updates.

(2) Data downloads into new relations at a user's workstation or local database.

(3) A query optimizer to combine the semijoin and replicate algorithm.

(4) Improved user interface features.

(5) Development of a schema design tool for users.

More research will occur in the areas of expert systems, secure systems and object management systems.

CHAPTER 3

HETEROGENEOUS ISSUES

3.1. Introduction

The focus of this thesis is on the key issues introduced by the heterogeneous dimension of a D-DBMS. These issues have been separated into the following four areas.

(1) The definition of the data model and system structure that is necessary to support dissimilar databases using different data models.

(2) The translation between the Data Manipulation Languages of the different DBMSs.

(3) Mapping equivalencies between the schemata of the different data models.

(4) Integration of the schemata to define the HD-DBMS at a global level.

The following sections discuss these issues and present possible solutions.

3.2. The Architecture of the Distributed System

3.2.1. The Global Components

Several components are necessary to support the distributed processing that occurs in a HD-DBMS. Providing a global view of the data requires collecting the distributed data from separate sites, and merging it into a single view. Depending on the architecture of the system, varying degrees of functionality are provided.

The structure of the components that make up the system will determine its functionality, resiliency and the amount of global system maintenance required. We first define the basic components needed in a HD-DBMS.

3.2.1.1. Description of the Components

Global Data Manager

The primary component, termed the Global Data Manager (GDM), coordinates the global operations that occur in a distributed database. The GDM uses multiple modules to perform a variety of tasks including query translation, query optimization, transaction management, schema mapping and user interfacing.

In one implementation (section 2.5, AIDA), a GDM is not used for actual processing of the query per se. Instead, it uses the capabilities of each of the local sites to manipulate the data and perform postprocessing. An example would be two local sites retrieving the data, and one of the two performing the final JOIN on the two data sets. In other implementations (e.g. SIRIUS-DELTA, MULTIBASE) the GDM itself supports processing of global transactions.

Global Data Dictionary

The GDD is the second critical component in a distributed database. It stores information about the global databases and the distributed system as a whole. It is especially important in a HD-DBMS where this information would describe each of the local databases, their capabilities, and the data they manage that is used by the Global Data Manager. A point of clarification may be useful here. Even though a local database may be a subset of a global database, not all of the data in the local database would be part of a global database. The information held in the GDD would define mappings between the data models and schemata, query translation and optimization methods, and provide the data necessary to resolve incompatibilities between local databases.

User Interface

A component necessary for end-user interaction with the global database is the user interface. In many cases, a global query language is defined for this purpose. It is used by HD-DBMS to provide a site-independent neutral language format, with which queries may be submitted, decomposed and optimized. Global queries are decomposed into subqueries, which are then transmitted to their appropriate local DBMS for processing. Translation of the subquery, from the global into the local query language, may occur before or after transmission to the local site.

Local Data Manager Interface

In order for the GDM to operate in a distributed manner, it must interact with the local sites. This interaction should not interfere with the autonomy of the local DBMSs. A component, the local data manager interface, exists at each of the local sites managing their portion of a global operation. This component appears to the local DBMS as if it was a local application program submitting operations for a local transaction. This interface is responsible for receiving a subquery and processing it locally. The results are either passed back to the Global Data Manager or forwarded to another site for further processing.

3.2.1.2. Structure of the Components

There are two general architectural configurations that these components can assume, Centralized and Distributed.

Centralized

The Centralized architecture is the more straightforward of the two. As its name implies, a single central site is used to manage the distributed system.

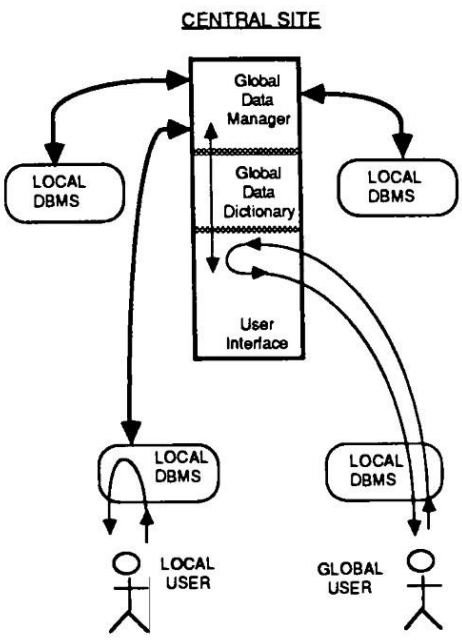
The global data manager, data dictionary and user interface components all reside on a single host machine (Figure 3.1a). All interaction with the distributed database is through this site. This configuration can substantially simplify the HD-DBMS, because it has properties similar to a centralized (non-distributed) DBMS. One of the more difficult aspects of a distributed system, concurrency control, is more easily managed when a single global data manager is controlling all of the global transactions.

Distributed

The Distributed architecture is a more complex implementation of a HD-DBMS. While providing the same functionality as a centralized system, all of the global components do not exist on a single host machine (Figure 3.2b).

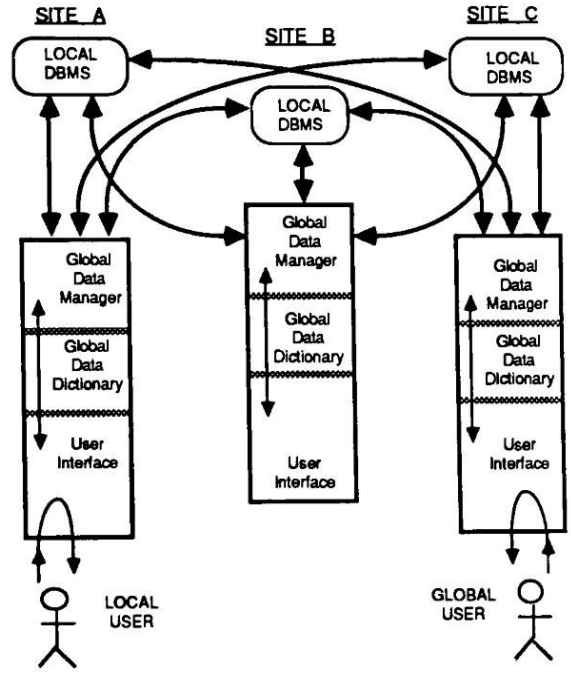
One or many of the components may exist at the local sites permitting each site to manage distributed operations. For example a user interface module could operate on each of the local sites and communicate to a central global data manager. The extreme of the distributed architecture, would be for all of the components to exist at each local site. In this instance, each site acts similar to a centralized DBMS architecture, yet all of the sites must cooperate with each other.

In comparison, the two architectures, Centralized and Distributed, are similar in how they handle a query. Both will use a single site to manage a single global query. However, in the distributed architecture, the complexities become more pronounced. Each site is now capable of managing distributed transactions. With multiple sites initiating global operations, concurrency conflicts between global data managers will result. The global data dictionary is replicated on each site requiring dictionary updates to be



Centralized

Figure 3.1a



Distributed

Figure 3.1b

propagated throughout the system.

A major disadvantage to the centralized architecture is the global data manager residing on a single site. If the global data manager is unavailable either due to site or network failure, distributed transactions cannot occur. A centralized architecture also forces all global transactions through a single site, with an attendant concentration of network traffic.

3.2.2. Global Data Model

The global data model is perhaps the most critical attribute of the HD-DBMS. It is the common descriptive model that is used to represent the heterogeneous distributed database. In a homogeneous system all the local sites are using the same data model. Mappings are still required; however, they are limited to those between different database definitions, not data models. The global data model used for the HD-DBMS must be capable of mapping between both the data models and the database definitions at each of the local sites.

In considering the three predominant data models in use today, hierarchical, network (CODASYL) and relational, the relational model provides the most flexibility. Other models that have been proposed or implemented include the Entity-Relationship model [CHEN80] and the DAPLEX functional data model [SHIP81]. In the majority of existing HD-DBMSs, the relational model was most commonly employed as the global data model.

3.2.3. Global Data Manipulation Language

Associated with the global data model is the global data manipulation language. A user interface provides an environment for interaction with the HD-DBMS, and needs a common language to conduct global operations. Defining the global conceptual schema, global databases, and queries for

global processing would all occur using this language.

In a homogeneous system, a single Data Manipulation Language is common to all of the local DBMSs. In a heterogeneous system, the capability must exist to support dissimilar DMLs used by the various local DBMSs. Translators are used to convert commands from one DML to those supported by the DML of another DBMS. Use of a common DML provides the distributed system and global user with a single neutral language, minimizing the number of translators required.

The two basic types of query languages used by DBMSs are procedural and non-procedural. A procedural language navigates through the database by describing the methods of data access. This usually produces a record-at-a-time style of output and can be code intensive even in a high-level language. A non-procedural language specifies the results desired from the query and returns a data set as output. While both types of languages can produce identical output from the same query, the non-procedural language is a more straightforward approach for a casual user. In a database environment, users usually know what results they desire, but are not experienced or knowledgeable enough to formulate a procedural query.

Once a query has been submitted to the global data manager, it must be processed at the global level. This involves optimization of the global query and its decomposition into

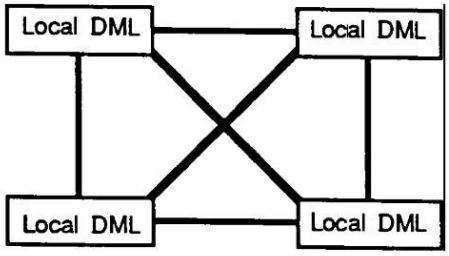
subqueries. This operation is common to both homogeneous and heterogeneous D-DBMS. Translation of the query from the global DML to the local DML is where the differences appear.

3.3. Query Translation

When a query is submitted using a global query language, it must be translated into the data manipulation language of the local DBMS. This query translation includes both mapping of names (table, record, field, etc.) and the translation of the operators used by the query languages.

The global data manager communicates with each of the local sites when managing a transaction. Using a global DML as a standard language minimizes the number of translators required by the HD-DBMS. Without a common language, n(n-1)/2 translators are necessary to support n dissimilar DMLs (Figure 3.2a). When all dissimilar DMLs are translated into a common DML, only n translators are required (Figure 3.2b).

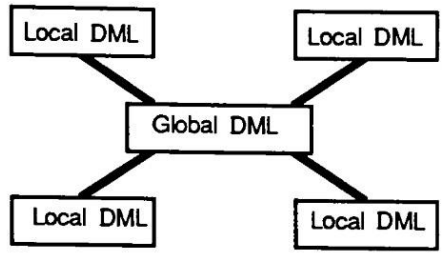
In some cases it may be inefficient to translate a query from a global DML to a local DML. For example, if the site where the query originates is the only site required to process the query, it would be advantageous to submit the query using the local DML directly. However, users may be unaware



n = 4 dissimilar sites
T = 6 translators

No Global DML

Figure 3.2a



n = 4 dissimilar sites
T = 4 translators

Global DML

Figure 3.2b

of the location of the data and it is not usually feasible to train the user in two different systems. Providing a single uniform interface to the data, whether it is global or local data, is a desirable goal.

Once a global query has been decomposed, the subqueries are transmitted to their respective sites for local processing. At some point, the subqueries are translated from the global DML into the local DML. This translation between DMLs is unique to HD-DBMSs.

Translation occurs most commonly at the site of the local DBMS because the translators are DBMS dependent. The translator must handle many variables when converting the subquery into a local query. A one-to-one translation converts the global DML syntax and references into local DML syntax and references. However it must resolve the inherent

processing operator differences found in the DMLs that support different data models. In the case of the global data model being relational and a local site using a hierarchical DBMS, the set of operators of the DMLs differ greatly.

As an example, consider the case when a local non-relational DBMS is required to process the relational JOIN operator. During translation into the local DML, the statement for that operation would be changed. The modification involves translating the unsupported JOIN operation into an equivalent group of supported DML statements. This equivalence must be strong enough so that the results returned by either of the forms would be identical. Section 3.5 discusses this key issue in detail.

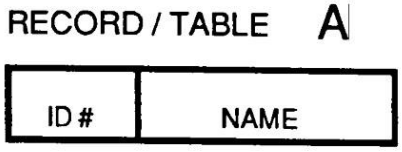
Conversion of relational (non-procedural) operators into sequences of navigational (procedural) operators is fairly straightforward. Described below, are translations of three basic relational operators into a set of procedural steps.

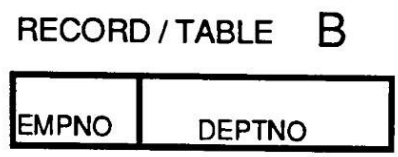
SELECT In a relational DBMS, this operation selects rows or records from a table. Selection is based on boolean operations applied to the values of the data in the table, (e.g. Find all employees where Lastname = "Doe"). Rows that meet the condition(s) are placed into a result table. This command, translated into a procedural language, would require a loop that would access a record. It would then check the values of the data and if they meet the

conditions, store them in a buffer. The loop then continues and accesses the next record.

PROJECT This operation processes an entire table, producing as its result, values from specified fields for each row in the table. Its equivalence in procedural statements would also be a loop. The loop accesses each record selecting from it the values of the desired fields. These are placed into a buffer and the loop continues.

JOIN This is one of the more complex relational operations. Two tables (A & B) are JOINed by comparing the values between the JOIN field(s) in each of the tables, the results form a third table. In a procedural language this would correspond to two nested loops. The outer loop would access a record from data set A_n then the inner would process all of the records in data set B. For each match found between the two data sets, the data values for each of the records are copied to a buffer. The matching field(s) are only copied once into the resultant table. Once the inner loop completes processing data set B_n the outer loop accesses the next record in data set A. This processing continues until data set A has been fully processed. See Figure 3.3 for an example of a JOIN, SELECT and PROJECT, using both procedural and non-procedural languages.






```
retrieve A.NAME, B.DEPTNO
    where A.ID# = B.EMPNO
```

Figure 3.3a Non-procedural

```
while not end of A
{
  while not end of B
  {
    if (A.ID# = B.EMPNO)
      buffer A.NAME, B.DEPTNO
    next B.EMPNO
  }
  next A.ID#
}
```

Figure 3.3b Procedural

Figure 3.3

JOINs, SELECTs and PROJECTs destined for the same DBMS may be combined by the global query optimizer before transmittal to the local site. This will minimize network communications by passing the subquery in a form that uses fewer statements. As mentioned above, translation from the global DML to the local DML commonly occurs at the local site. However the results of the translation are not usually optimal for local DBMS processing of the query. Since many of the factors that affect processing of the subquery are site dependent, the local DBMS will perform local query optimization after DML translation has occurred.

The translated subquery will be processed by the local site

and the results returned to the local data manager component that is acting remotely for the global data manager. Results are commonly formatted into a standard form as in MULTIBASE (section 2.4) or in AIDA (section 2.5). In this manner, the results of the query may be applied to another site for continued processing, or formatted into a report for the user interface.

3.4. Schema Mapping

3.4.1. Definition of Schema Mapping

A database uses a schema to describe its entities (data objects) and their relationships. In a heterogeneous distributed database, local DBMSs have different data models and schemata that define their databases. Mappings among these schemata are necessary to establish logical equivalences between the definitions.

When the global database is heterogeneous, the mapping operations are more complex because they involve both the schemata and the data models. The mappings must have the capability to translate the data objects, the relationships among the objects and be able to convey the integrity constraints that exist between the objects.

3.4.2. Determining Schemata Equivalency

Upon creation of a heterogeneous distributed database, a schema is defined for the global database. Portions of the schemata from the local databases are mapped to the global schema to provide the distributed database with the required logical equivalencies. In comparing the schemata, there are three degrees of equivalency that can be demonstrated. These are classified as Static, Dynamic and Semantic.

Static Equivalency

Static equivalence involves the translation of a specific state of one database, to a specific state in another database. The schemata are said to be statically equivalent if, all of the facts represented by one database, are also represented by the other database [MCGE74], and vice versa. Another name for this is Instance Level equivalency. There are two weaknesses in this type of equivalency. Transitions made in one database are not mapped to the other database and the meaning of the data is not mapped.

The lack of transition mapping means that there is no guarantee that a modification to the data in one database, can be represented in another database. The before and after states can be represented, but the actual transition between the states cannot. Transition mapping can appear at

two levels in a DBMS. On the data model level it could be a function supported by one data model but not another (JOIN). On the database level it would involve integrity constraints specific to one database schema that could not be mapped to the other.

The second weakness in static equivalency it that the meaning of the data would not be mapped when conversions between data types and field names occur. Figure 3.4 shows two schemas (1 & 2) that represent the same facts. The field names can be mapped, NAME <==> ID, DISTANCE <==> MILEAGE and GAL <==> VOL. The NAME and GAL fields can be mapped directly, however the DISTANCE <==> MILEAGE mapping requires a conversion based on its meaning (DISTANCE/GAL <==> MILEAGE).

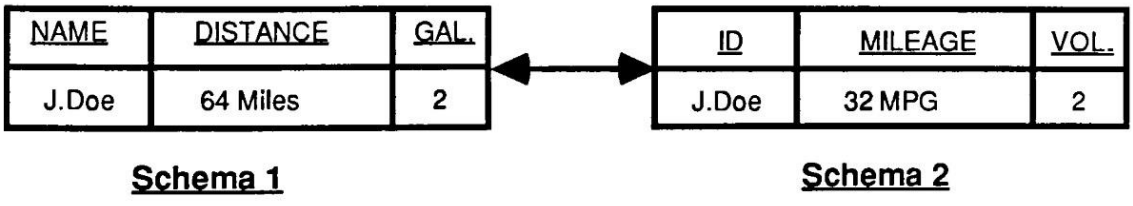


Figure 3.4 Mapping between Dissimilar Schemata

Dynamic Equivalency

Dynamic equivalence provides for transitions in the databases and assures that the schemata are equivalent when changes are made. Its definition is as follows. For two database schemas, X & Y, to be dynamically equivalent, two conditions must be met:

(1) For all possible states of database X, database Y will be statically equivalent and vice versa.

(2) For any transition allowed by the integrity constraints of database X, database Y will permit the same transition, and vice versa.

Here it is shown that dynamic equivalency includes static equivalency as part of its definition and then extends it further to map the transitions that occur within the databases. This insures that both schemata describe the same set of entities and relationships. Instance level equivalencies, Figure 3.4, are still not guaranteed to be equivalent even with dynamic equivalence.

Semantic Equivalency

Semantic equivalence of database schema involves equating the meaning of the schemata as well as the structure and integrity constraints. An example would be, a field containing a numeric value that represents quantity. It is

unknown to the DBMS whether it is the quantity on hand, or the quantity ordered. This represents a difference in the semantics of the quantity field. In order for an accurate mapping to occur between schemata, the semantics of the schemata must be translated as well. In order to integrate schemata from dissimilar databases, equivalencies must be defined statically, dynamically and semantically.

3.4.3. Implementation of Schema Mapping

With more than one data model being used in a heterogeneous distributed DBMS, schema mapping will occur between dissimilar data models (e.g. network to relational).

The three schema architecture was defined by the 1976 ANSI/SPARC report [ANSI76]. It defines the three schema levels to be External, Conceptual and Internal. An External schema is a users view of the data, of which there can be many. The Conceptual schema is the logical description of the entities in a database, their attributes, data types and relationships to other entities. One will exist for each of the local site databases as well as for the global distributed database. The Internal schema describes how the data is physically stored on devices at various locations.

When equating an external view to the physically stored data, the external view is mapped to the Conceptual schema.

This mapping provides logical data independence. The view portion of the Conceptual schema is then mapped to the Physical Storage device(s) providing physical data independence. The three schema architecture has been implemented in centralized DBMSs, most notably the relational DBMS model.

This architecture is used in a similar manner for distributed DBMSs. Mappings occur between the internal schema of the global DBMS and the external schemata of the local DBMSs (Figure 2.2). A global query is submitted to the HD-DBMS through the global external schema. The global conceptual schema is used to reference global data for manipulation. When converting these global references to those in a local database, a mapping occurs between the schemata. Instead of the global internal schema being mapped to a physical storage device, it is mapped to a local external schema at the necessary sites. The global internal schema may also describe the location and replication rules for the data.

Translation of the Schemata

The schema for the global database is defined by the various local external schemata. This definition, and subsequent mapping(s), require that the local database schemata are translated into the same data model as the global DBMS.

As stated previously (section 3.4.2), equivalencies among

the database schemata must occur on a semantic level to fully integrate the schemata. Facts that describe a real world state will define the instances represented in a database, and vice versa. A real world state cannot be formalized so an abstract state is introduced to formally model the real world state, including its semantics.

Two formal languages, Logical Data Language (LDL) and Logical Data Definition Language (LDDL), have been defined [BILL78] to represent the abstract state and its corresponding abstract model respectively. The LDDL is a semantic metalanguage used to define the semantics of a database schema. When the schemata of two databases can be translated into the same LDDL schema, they are said to be equivalent [BILL79].

It is necessary for the LDDL to be a complete language to cover all of the possible definitions in local data manipulation languages. It must be capable of defining any logical data structure as well as any integrity constraint defined by a DML. This implies that the global semantic model should support the most fundamental data structures. Mapping of a database schema to the semantic model is then reduced to a decomposition of complex data structure declarations into elementary data structures.

Translation of a schema into an intermediate schema demonstrates:

(1) The need for the semantics of the schema, to accurately map between schemata (and data models).

(2) It is advantageous to use a formal semantic metalanguage for this purpose.

(3) Use of a metalanguage helps to correctly translate the integrity constraints.

(4) An intermediate step will simplify proof of the schema equivalence.

(5) A direct mapping can be derived from the two mappings.

Below is an example of the schema definitions that would result when a CODASYL schema is mapped to a Relational schema. The intermediate LDDL schema is shown as a step in the mapping process.

CODASYL to LDDL

Defined in Figure 3.5 is the CODASYL schema for the Curriculum database. It has three entities in it, PROFESSOR, STUDENT and COURSE. They are used to describe the relationships between Professors teaching Students in a particular Course. The integrity constraints for this database are:

-a PROFESSOR must teach at least one CLASS.
-a PROFESSOR may teach no more than three CLASSES.

The LDDL schema is a formal semantic description of the CODASYL schema, for the Curriculum database (Figure 3.5). It is used for mapping between the CODASYL definitions and the LDDL definitions. The LDDL declarations (Figure 3.6a), define the data types used in the LDDL schema. The LDDL

```
Schema Curriculum-Database
  area PROFESSOR
  area COURSE

  Record PROFESSOR
    identifier PID
      PID: picture 9              {PROFESSOR ID}
      PNAME: picture A(20)
      NUMB-OF-CLASS: picture 9
      CLASS: picture A(20); occurs NUMB-OF-CLASS

  Record COURSE
    identifier CID
      CID: picture 9              {COURSE ID}
      CNAME: picture A(20)

  Record STUDENT
    identifier SID
      SID: picture 9              {STUDENT ID}
      SNAME: picture A(20)

    Set PROF-COURSE
      owner PROFESSOR
      member COURSE
      insertion automatic retention mandatory
      structural PID of COURSE equal PID of PROFESSOR

    Set STUDENT-COURSE
      owner STUDENT
      member COURSE
      insertion automatic retention mandatory
      structural SID of COURSE equal SID of STUDENT
```

Figure 3.5 CODASYL Schema for Curriculum database

schema of the Curriculum database describes the relations (Figure 3.6b).

There are corresponding definitions for the PROFESSOR, COURSE and STUDENT entities in the two schemas (e.g. Record STUDENT <=> Relation HAS SID). The CODASYL schema also defines two sets, PROF-COURSE and STUDENT-COURSE. These sets represent the links from the PROFESSOR and STUDENT records, to the COURSE record. The corresponding relation in the LDDL schema is LECTURES, it represents: PROFESSOR LECTURES STUDENT in COURSE. This entity supports the many-to-many relationship between the Professors and the

```
basic integer
basic string

type PID     subset of integer
type PNAME   subset of string

type CID     subset of integer
type CNAME   subset of string

type SID     subset of integer
type SNAME   subset of string

type PROFESSOR
    characteristics: HAS PID
        identified by {HAS PID}

type STUDENT
    characteristics: HAS SID
        identified by {HAS SID}
```

Figure 3.6a LDDL Declarations for the Curriculum database

Students. The integrity constraints, (number of classes taught by professor) are also mapped to the LDDL schema. This mapping occurs with a formal statement of the constraints, and in the form of the TEACHES A COURSE and TEACHES MANY COURSES entities.

To properly map between the schema is a critical operation. The LDDL schema must be defined by a data base administrator knowledgeable in the semantics of the source (CODASYL) schema. Once defined, it requires maintenance when the CODASYL schema is modified.

LDDL to Relational

After the CODASYL schema is mapped into the LDDL schema, it is translated from the LDDL schema into the relational schema. To reproduce the correct representation of the CODASYL schema, the relational schema must represent all of the data types, entities, relationships and consistency conditions described in the LDDL schema.

The relational schema requires type declarations for each of the domains (Figure 3.7a) and definitions of the relations themselves (Figure 3.7b). The key fields in the PROFESSOR and STUDENT relations are the PID and SID respectively. In the TEACHES relation the PID, SID and CID domains make up the composite key.

```
relation HAS PID
    domains equal to PROFESSOR, PNAME
        functionally dependent on PROFESSOR

relation HAS CID
    domains equal to COURSE, PNAME
        functionally dependent on COURSE

relation HAS SID
    domains equal to STUDENT, SNAME
        functionally dependent on STUDENT

relation LECTURES          {Professor lectures Students}
    domains equal to PROFESSOR, STUDENT, CID
        functionally dependent on PROFESSOR, STUDENT

                           {Professor teaches a Course}
relation TEACHES A COURSE
    domains equal to PROFESSOR, CID, CNAME
        functionally dependent on PROFESSOR, CID
        restricted by PROFESSOR, CID min 1 max 3;

                           {Professor teaches many Courses}
relation TEACHES MANY COURSES
    domains equal to PROFESSOR,
                           INTEGER where (0 <= n and n <= 3)
        functionally dependent on PROFESSOR

Integrity Constraints:
  There exists  n  where:
    - n is an element in  TEACHES A COURSE.PROFESSOR
      such that:
         the number of occurrences of  TEACHES A COURSE,
                    is  <=  TEACHES MANY COURSES.INTEGER
```

Figure 3.6b LDDL Schema for the Curriculum database

Mapping between the LDDL schema and the relational schema is fairly straightforward. The HAS PID, HAS CID and HAS SID entities in the LDDL schema, correspond to the PROFESSOR, COURSE and STUDENT relations The LECTURES entity maps to both the TEACHES and ATTENDS relations to form two distinct

```
domain PID
    description: integer

domain CID
    description: integer

domain SID
    description: integer

domain PNAME
    description: string where size = 20

domain CNAME
    description: string where size = 20

domain SNAME
    description: string where size = 20
```

Figure 3.7a Domain definitions for Curriculum database.

```
PROFESSOR (PID, PNAME)

COURSE (CID, CNAME)

STUDENT (SID, SNAME)

TEACHES (PID, CID)

ATTENDS (SID, CID)

Consistency Constraints:
    PID must teach at least 1,
            but no more than 3 courses.
```

Figure 3.7b Relation definitions for Curriculum database

relations in the Relational schema. To complete the mapping, the integrity constraints described by the TEACHES A COURSE and TEACHES MANY COURSES entities and defined

formally in the LDDL schema are created in the relational schema. This produces the formal integrity constraints that are translated into the relational DML. Integrity constraints such as these (minimum/maximum), are usually managed by the relational DBMS. These mappings are again defined by a data base administrator who is knowledgeable about the semantic meaning of the schemata.

Schema mapping is but a translation of a database definition from one data definition language and data model to another. To create an accurate representation of the facts in the target schema, requires far more than a static or dynamic mapping. The semantics of the data must be mapped as well.

Mapping the data definition precisely requires a translator. As shown in this section, the schema mapping is a complex operation. Equivalencies between data models, data definition languages and schemata are manually defined by the GDBA. Once the methodology and rules for a mapping have been determined, a translator can be constructed. Information needed to map could be coded into the software to support fast mapping operations. However this information is subject to change when either the global or local schema are modified. An alternative method is to store the schema mapping information in a database. This provides flexibility in the mapping translators, but may reduce their performance.

Using either method, once a specific translator has been implemented, it can be used in many locations if necessary. That is to say, once a Schema A to Schema B translator has been built, it can be used whenever that mapping occurs.

3.5. Integration of Schemata

3.5.1. Introduction

Schema Integration is the act of combining local schemata into a global schema, that represents the heterogeneous distributed database. Local schemata are integrated into the global schema when they are mapped to the global data model.

Even when all of the local databases belong to the same organization, there is no guarantee that their schemata are structured the same. In the previous section it was shown how mappings occur between dissimilar schema and data models. These translations are but the first step taken in the integration process.

3.5.2. The Process of Schema Integration

Integration of the local schemata, requires the generation of three essential structures.

(1) Definition of the global schema required by the HD-DBMS.

(2) Development of the schemata mapping translators.

(3) Definition of the Integration Data Base (IDB) for use in resolving conflicts among schemata.

These are used to support the mapping among the local schemata and provide the basis for their integration.

To clarify the first point, the global schema requires a separate declaration since it is not the union of all local schemata. The global schema consists of whatever entities, attributes and relationships the GDBA defines. This individual must be knowledgeable in the relationships between the local schemata, and understand which entities are necessary to represent the global database. The global schema may use the entire schema of one local database and a subset of another database. The definition of the global schema provides the cornerstone for the integration of the local schemata.

Once the global schema has been defined, the translators which map between the local and global schemata can be designed and implemented. As described in section 3.4, these translators may cope with dissimilar data models as well as schemata.

3.5.3. Schemata Conflicts

In conjunction with the translations, functions may be applied during mapping to resolve conflicts between the schemata. These functions modify the mappings for integration into the global schema. The conflicts to resolve are due to the dissimilarity among the local schemata and can exist in the following forms:

(1) Name - a name used by an individual schema may be a homonym or synonym to a name in another schema.

(2) Scale - the units used to measure a value may differ between two schemata.

(3) Structural - what is a relation or record in one schema, may be an attribute in another.

(4) Level of Abstraction - a single attribute in one schema may be represented by many attributes in another schema.

(5) Inconsistencies amongst Replicated Data - there may be disagreement in the value of data that is stored by more than one database.

In order to resolve these conflicts, there exists the IDB. This database, used by the HD-DBMS, stores information about

the local schemata. The information is used to construct the functions which convert a local schema entity (or attribute) into a global schema entity. The complexity of these functions is dependent on the type of conflict they resolve. Each of these conflicts is discussed in detail below.

3.5.3.1. Name Conflict

Name conflicts appear in two forms, Homonyms and Synonyms. The name could be that of entities, attributes, sets or even databases.

Homonyms occur when a name used in one database schema, is used in another database schema but with a different meaning. An example is shown in Figure 3.8 where the attribute NAME, represents the name of the ID field (SSN) in Schema A. The NAME field in Schema B is the name of the person which corresponds to the NUMBER.

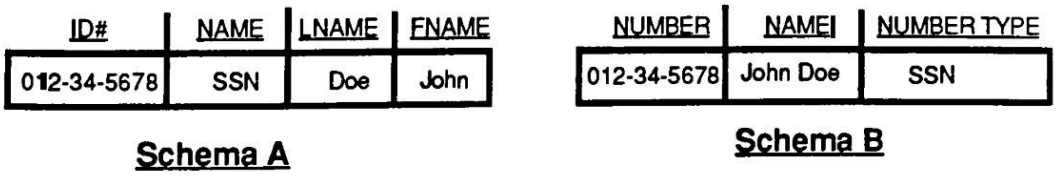


Figure 3.8 Name Conflict

Synonyms exist when there are different names which have the same meaning. In Figure 3.8, the fields ID# and NUMBER both reference the same piece of information, yet use different names.

The IDB stores the information which defines the relationships between the local schema and global schema names. A function is used to convert from one name to another. The function applied to these conversions could be a simple lookup table.

The following is an example of name conversions. In Schema A, the ID# and NAME fields are equated to the NUMBER and NUMBER_TYPE fields in Schema B respectively. To complete the conversion, the FNAME and LNAME fields would be concatenated and mapped to the NAME field in Schema B. This particular conversion displays Level of Abstraction conflicts and are discussed later.

3.5.3.2. Scale Conflict

Scale Conflicts occur when different units of measure are used for the same data. For example, there may be a measurement of temperature in a database. One schema may store the value in degrees Fahrenheit, another schema may use degrees Centigrade. Both units of measure are equally valid, however when they are mapped to the global schema, different functions must be applied to them.

This type of conversion is calculable, (e.g. degrees Fahrenheit = degrees Centigrade * (9/5) + 32), so the functions are mathematical in nature. If another unit of measure is given, say an enumerated one such as Cold, Cool, Warm and Hot. Then the function may involve a lookup table in the IDB to determine what numeric value should be used.

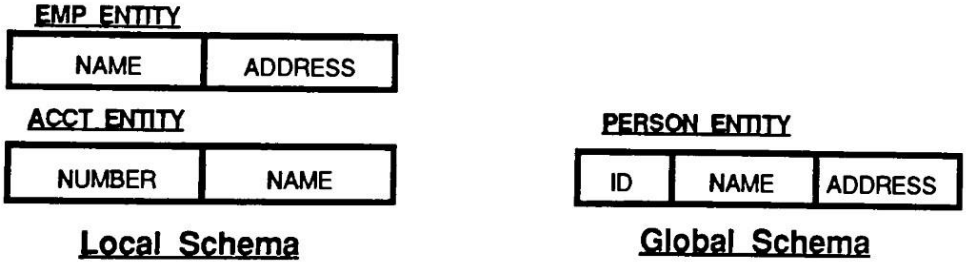
3.5.3.3. Structural Conflict

This type of conflict is common in a HD-DBMS. A structural conflict occurs when an entity in a local schema is an attribute or relationship in the global schema. This type of mapping can occur in either direction, entity to attribute or attribute to entity.

When a local entity is mapped, as an attribute, to a global entity, additional information is needed. It must be determined which global entity is being mapped to, possibly using a lookup table. More specifically, what are the rules or guidelines which must be followed in determining to which occurrence of the global entity, the local entity data will be mapped. This information would be found in the IDB.

In the example shown in Figure 3.9, the local schema consists of the EMP and ACCT entities. The global schema defines the PERSON entity which contains the EMP entity data as attribute data.

To map the EMP entity data accurately, the ACCT.NUMBER value must be used. The NAME fields in the EMP and ACCT entities are "JOINED". This allows the ACCT.NUMBER field to be used



Entity to Attribute Mapping

```
map:                EMP to PERSON.NAME, PERSON.ADDRESS
(using results of)      using ACCT.NUMBER
(link NAME to NUMBER)   where EMP.NAME = ACCT.NAME
```

Figure 3.9 Structural Conflict

to link the PERSON.ID field to the respective NAME and ADDRESS fields from the EMP entity. In this example the ACCT_NUMBER field is not mapped to the PERSON.ID field. It has been used only to reference the local entity data to its appropriate global entity as attribute data.

In defining these mapping rules, the Global Data Base Administrator must be able to determine how a local entity relates to the attributes of a global entity. Being able to specify which occurrence of a global entity to map the attribute data to, is of prime importance.

3.5.3.4. Levels of Abstraction Conflict

Similar to Structural conflicts are conflicts among the Levels of Abstraction. This pertains to the degree of detail in which data is defined in the individual schemata.

As shown in Figure 3.8, Schema A stores a person's name in two fields, LNAME for their last name and FNAME for their first name. Schema B stores the person's full name, in the single field NAME. To integrate these two definitions, the fields in Schema A may be concatenated to form a single field. Alternately the single NAME field in Schema B may be parsed to form two fields.

Another example of conflicts in Levels of Abstraction is

shown in Figure 3.10. Schema X defines two address fields (HOMEADDR and OFFICEADDR) for a person. Schema Y also defines an address, simply titled ADDRESS which is the person's home address. This fact is known by the Local DBA for Schema Y. It is critical for the Global DBA to be aware of this definition in order to map the ADDRESS attribute to the HOMEADDR attribute. In this instance the OFFICEADDR field would either be filled in from another source, or left null.

3.5.3.5. Inconsistent Replicated Data

In some HD-DBMS the local data may be duplicated on more than one system. This provides a redundant system for the access and manipulation of data.

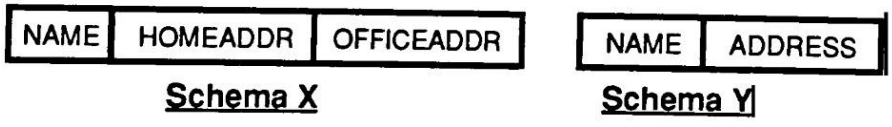


Figure 3.10 Levels of Abstraction Conflict

Updates to this data may not support the degree of accuracy desired. The values of the data may vary from site to site depending on update methods and propagation time. This could cause different results to be returned for the same query, depending on which sites are available and/or used for the query. This type of conflict, one of inconsistent data, may be resolved in a few ways.

One solution is to use the aggregate of the values to determine the approximate value. As long as the user is aware that the results were determined by aggregation and may not be accurate, a "best approximation" type of answer has been provided. This method may be adequate for many of the queries posed to the system. It can, however take time to compile a result if data is replicated at many sites.

When more accurate results are required, the system may search based on the last date/time modified value for the data. The most accurate value might be found at any of the sites that store replicated data. A drawback to this would be if the site with the most recent modification were unavailable, then the user may still receive "old" information. A subset of this type of search would be for data which represents an accumulating value (i.e. count of vehicles through a toll booth). In this case the highest value would be the most accurate. Again, if not all sites which retain this data are available, the retrieved data may not be as accurate as desired.

The Integration Data Base plays an important part in the integration of the local schemata. It is used for simple conflicts such as name mappings or conversion of units of measure. It can also resolve conflicts involving structural or abstract definitions of the schemata. The IDB may coexist with the global data dictionary. In this manner, the IDB may be centralized or distributed across the local sites.

CHAPTER 4

PROPOSAL FOR A HD-DBMS ARCHITECTURE

4.1. Introduction

This chapter proposes an architecture for a HD-DBMS. It is discussed in two sections. The first section describes the general concepts of the system architecture. It defines the components in the system, their physical relationships to each other and their functions.

The second section goes into more detail on each of the components. Their design is discussed and the functions of each are described. Problems associated with the components and their structures are presented and possible solutions proposed.

4.2. Conceptual Architecture

In order to define an architecture, the basic requirements of the system are first listed. It should provide:

-Management of the Global Schema and its associated data.

-Management of Distributed Transactions.

-Mechanisms for integrating dissimilar databases.

-Transparent views of distributed data.

-A consistent interface for HD-DBMS users.

-A robust HD-DBMS that provides for graceful degradation.

-Autonomy for local DBMSs.

The architecture and functionality of the proposed system meets these basic requirements by specifying a HD-DBMS that a) is truly distributed in its component and schema structure, and b) will support fragmented and replicated data and c) provides distributed access and computation to all global users. In order to support these capabilities, two core modules are required.

(1) A Global Data Base Module (GDBM) which supports all global interaction with the global database schema. This module (Figure 4.1) would consist of the Global Data Manager, Global Data Dictionary, Integration Data Base, Network Interface (NI) and User Interface (UI). To support full distribution of the HD-DBMS, a GDBM

will exist at each (primary) site requiring global database capabilities. Other (secondary) sites would be data resources for the heterogeneous distributed database, but would not require a GDBM. The components in this module would provide the user with a consistent interface to the global, and if desired, the local database.

(2) Each secondary site supporting the HD-DBMS requires a Local Interface Module (LIM). This module serves the GDMs at the various sites. It consists of three components, the Local Subquery Manager (LSM), Remote Network Interface (RNI) and Transaction Monitor (TM) (Figure 4.2). The LIM acts as an agent for the GDMs,

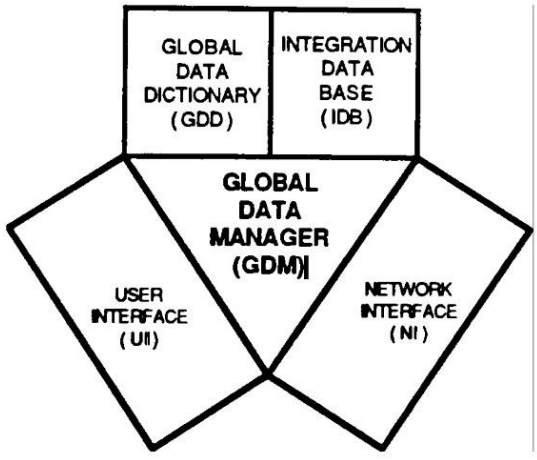


Figure 4.1 Global Data Base Module

managing their subqueries. Its tasks include maintaining integrity, providing concurrency control and supporting recovery of global transactions at the local level. The Local Interface Module interacts with the local DBMS in a manner similar to a local application program.

Query Processing Primary sites support the global users interaction with the global database. Global queries are submitted to the Global Data Manager through the User Interface component. They are decomposed at the site of origin and subqueries are submitted directly to the appropriate sites. In the case where sites are unavailable, replicated data, if it exists, is obtained from other sites.

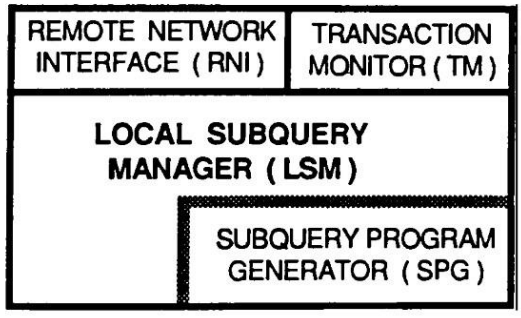


Figure 4.2 Local Interface Module

Communication between the primary and secondary sites, takes place between the Global Data Manager and the Local Subquery Manager components. Once the Local Interface Module has completed processing of the subquery, it returns the results to the GDBM. The data returned from the remote sites is integrated by the GDM for presentation to the user.

This is a brief overview of the proposed HD-DBMS architecture. The two core modules necessary to support this structure, the GDBM and LIM have been introduced. The next section will discuss each of these in detail and define what role each plays in the HD-DBMS.

4.3. Technical Aspects of the Distributed System

4.3.1. The HD-DBMS Modules

Two modules have been defined that are necessary to support the HD-DBMS, the GDBM and the LIM. At sites requiring distributed database capabilities, termed primary sites, the HD-DBMS is represented by a Global Data Base Module. This module is the foundation for operations that include global database definitions, schema integration, network communications and user interfacing.

Residing at local database sites is the Local Interface Module. It acts as a representative for the GDBM. An LIM

is required at any site if any portion of the local schema at that site supports the global schema. Sites supporting the global schema, but providing no global database capabilities, are termed secondary sites. The LIM is responsible for the execution of subqueries and supports recovery and concurrency control in the HD-DBMS.

4.3.1.1. Description of the GDBM Components

Within the GDBM are five basic components. The Global Data Manager component is the central process that controls the operations in the GDBM. The Global Data Dictionary and Integration Data Base are data resources providing information on global schema definitions and mapping functions for integration. The Network Interface and User Interface components are processes that support interaction between other sites and the users, respectively.

Global Data Manager

The GDM coordinates the global operations that occur in a distributed database. This component, described in section 3.2.1.1, uses all the remaining components in the Global Data Base Module to perform its tasks. These include:

-Management of Global Schemata

-Maintenance of the GDD and IDB

-Security of Global Data

-Processing Global Queries

-Global recovery operations

The GDM receives all global queries, data definitions and modifications through the User Interface. This interface provides a consistent method of communication for both the users and the GDM.

When the GDM examines the query, it uses the Global Data Dictionary to verify that the global schema specified in the operations does indeed exist. This verification is the first step in global query processing. The information in the Global Data Dictionary is used by the GDM during decomposition and optimization of the global query.

After a transaction plan has been formulated by the GDM, the subqueries are passed to the Network Interface for transmission to the Local Interface Modules. Even if a subquery is destined for the same site that the global query was posed from, the Network Interface will still be used as an intermediary to the Local Interface Module. This may seem to be an unnecessary step, however it provides both the Network Interface and Local Interface Module with consistent

methods of operation.

Once an Local Interface Module has processed the subquery, the results are returned to the originating Global Data Base Module. Its Network Interface receives the data, and passes it to the GDM. When the GDM has received the results of the subqueries from all of the Local Interface Modules, they must be integrated. The Global Data Manager integrates the results based on information stored in the Integration Data Base. Any necessary mapping functions are applied at this time to produce an integrated view of the results. This consolidated view is then passed to the User Interface for display to the user.

Global Data Dictionary

The GDD stores information about the global distributed databases for use by the HD-DBMS. It is one of the two information resource components in the Global Data Base Module.

The GDD retains the definitions of the global schemata, integrity constraints and security parameters. Also defined in the GDD is the fragmentation of the relations in the global schema.

Integration Data Base

The IDB is the second information resource component used in the Global Data Base Module. It stores the information necessary for integrating the local schemata into a global schema.

For each local database, the IDB describes the data model, the DBMS and its capabilities and the portion of the database schema used by the global schema. Physical locations of the databases and locations of replicated data (allocation schema) are stored here as well. In conjunction with the physical locations, are the communication and processing costs associated with each site.

The IDB stores the definitions of the mappings and functions required when integrating the schemata. These mappings and functions are applied to the data models, schemata and data manipulation languages.

Network Interface

This component manages communication between the local Global Data Manager and the Local Interface Modules at the remote sites. As mentioned earlier, the Global Data Manager uses this component to transmit subqueries to the remote sites.

When remote site or network failure occurs, the NI must take some type of action. The NI will notify the Global Data Manager that the subquery failed. The Global Data Manager may decide to abort the entire query, or resubmit the same subquery specifying to the NI to use alternate data sources. Most of these decisions are dependent on how the query was originally specified to the Global Data Manager.

The NI would also maintain records on communications costs, performance and availability. With this information, network data in the Integration Data Base can be kept up to date by each site.

User Interface

This component provides users with a consistent front end to the HD-DBMS. Global users will use the UI to interact with the global databases.

The UI supports use of a global data manipulation language to manipulate the global databases. If desired, this component could also be used to interact with the local DBMS. This would deliver to a user, a single interface for all database activities, whether global or local.

A user submits queries to the UI which processes them and passes them to the Global Data Manager. The UI manages user dependent functions such as user identification, global database security and device drivers necessary for terminal I/O. To provide adequate security provisions, additional checks are provided by the Global Data Manager. When the Global Data Manager has collected and integrated a result for the query, it is passed to the UI for display. User interaction with the results and additional queries are all managed by the UI.

The UI should provide a flexible and robust set of tools for a wide spectrum of users. A basic feature would be a command line driven interpreter. Using the global DML, users would submit queries to the global databases. Support should also exist for a menu driven, Query-By-Example [ZLOO75] type of interface. This allows casual users a method of interacting with the global databases without extensive training. Finally, the capability to embed the global data manipulation language into high level programming languages is also required. This permits complex batch and interactive jobs to be developed by experienced

programmers.

4.3.1.2. Description of the LIM Components

The Local Interface Module acts as an agent for each GDBM to support the distributed operations of their GDMs. The LIM consists of four components, the Local Subquery Manager (LSM), the Remote Network Interface (RNI) its Transaction Monitor (TM), and the Subquery Program Generator (SPG). This module is shown in Figure 4.2. In order to maintain site autonomy, the LIM appears to the local DBMS as if it were a local application program submitting transactions to a local database.

Local Subquery Manager

The LSM manages all subqueries that have been sent to that site for processing. Since it does not contain a user interface, no user interaction occurs directly with the LSM. It operates remotely from the Global Data Manager and performs its tasks in a specific sequence.

The LSM receives a sub-query from the Global Data Manager, through the Remote Network Interface. Subqueries, received in the global DML format, are first translated into the local DML. The LSM then passes the translated subquery to

the Subquery Program Generator for program creation. Using the Subquery Application Program (SAP), the LSM submits the subquery for execution by the local DBMS. The subquery results are then returned, using a standard format, to the Global Data Manager. It may integrate the results, or forward them to another primary site for further processing.

A critical responsibility of the LSM is to support global concurrency control. Since a single Local Interface Module exists at each site, it will be aware of all the global transactions directed toward that site. When the LSM detects possible deadlock between two or more subqueries, action must be taken. Initially, the subquery that created the condition will be suspended. The Global Data Manager of the subquery is then notified of the problem via the Remote Network Interface and Network Interface.

Another responsibility of the LSM is to support global recovery capabilities. The LSM uses the Transaction Monitor to keep track of active transactions. When local site failure occurs, the recovery facilities of the local DBMS will restore the local database. The LSM must manage recovery of the subqueries. The application programs submitted by the LSM will have been terminated and may or may not have completed normally. Since subqueries have been translated into application programs by the Subquery Program Generator, the LSM can determine which programs failed and at what point the failure occurred.

Upon restart of the system, the LSM will notify all the GDMs for which it had been managing transactions, that its services are again available. The LSM will use the Transaction Monitor to determine which transactions were interrupted and it will notify the Global Data Managers as to the status of their transactions. It will request further instructions (abort, continue, etc.) on each of the transactions and respond accordingly.

When network communications fail between the Local Interface Module and a Global Data Base Module, the LSM will store results of completing transactions. Any transactions that require Global Data Manager input will be suspended. Once communications have been reestablished, the LSM will resume interaction with the Global Data Manager. The status of the incomplete transactions will be determined and their disposition defined by their respective Global Data Manager.

Subquery Program Generator

The SPG takes, as its argument, the translated subquery. The subquery is used to produce an application program. Translated subqueries could be submitted to the local DBMS "as is", however this would not give the Local Subquery Manager the degree of control required. To maintain transparency of the distributed data, Local Subquery Managers must be capable of coping with local problems. Users are not available to make decisions, change the configuration of

the query, or resubmit it if data is locked. Therefore, the Local Subquery Manager needs a great deal of flexibility and control when processing subqueries.

To provide the Local Subquery Manager with as much power as possible, and without affecting site autonomy, an Subquery Application Program is generated. The Subquery Application Program has embedded in it, the subquery, ready for execution by the local DBMS. Error trapping for concurrency control, and integrity checks also placed in the program permit the Local Subquery Manager to intercede for correction or modification of conditions.

Remote Network Interface

The RNI manages network communications between the Local Subquery Manager and each of the Global Data Base Modules. It stores requests for services in the Transaction Monitor. After receiving a subquery request, the RNI passes it to the Local Subquery Manager for processing. Responses to subqueries from the Local Subquery Manager, are passed to the RNI for transmission to their corresponding Global Data Base Module.

Transaction Monitor

The TM records all transaction requests received from the Global Data Base Modules. This information may include an

ID number for the request, the request itself, site of origin and status. The TM stores a status for each subquery being managed by the Local Subquery Manager. The Local Subquery Manager will modify the status of the request as it changes. This log is critical for communication and recovery operations.

4.3.2. Configuration of the Modules

A Distributed architecture proves to be complex in implementation. While providing the same functionality as a centralized system, all of the modules do not exist on a single host machine (Figure 4.3).

In this proposed system, a Global Data Base Module and Local Interface Module would exist on every site requiring HD-DBMS capabilities. Local Interface Modules would also exist on additional sites needed to support the global schema. This arrangement creates a redundant architecture that provides a HD-DBMS which degrades gracefully. Any site with a Global Data Base Module can initiate global operations. Replication of data allows many transactions to still occur, even though the site containing the master data, may be unavailable.

Communications are not concentrated on a single site.

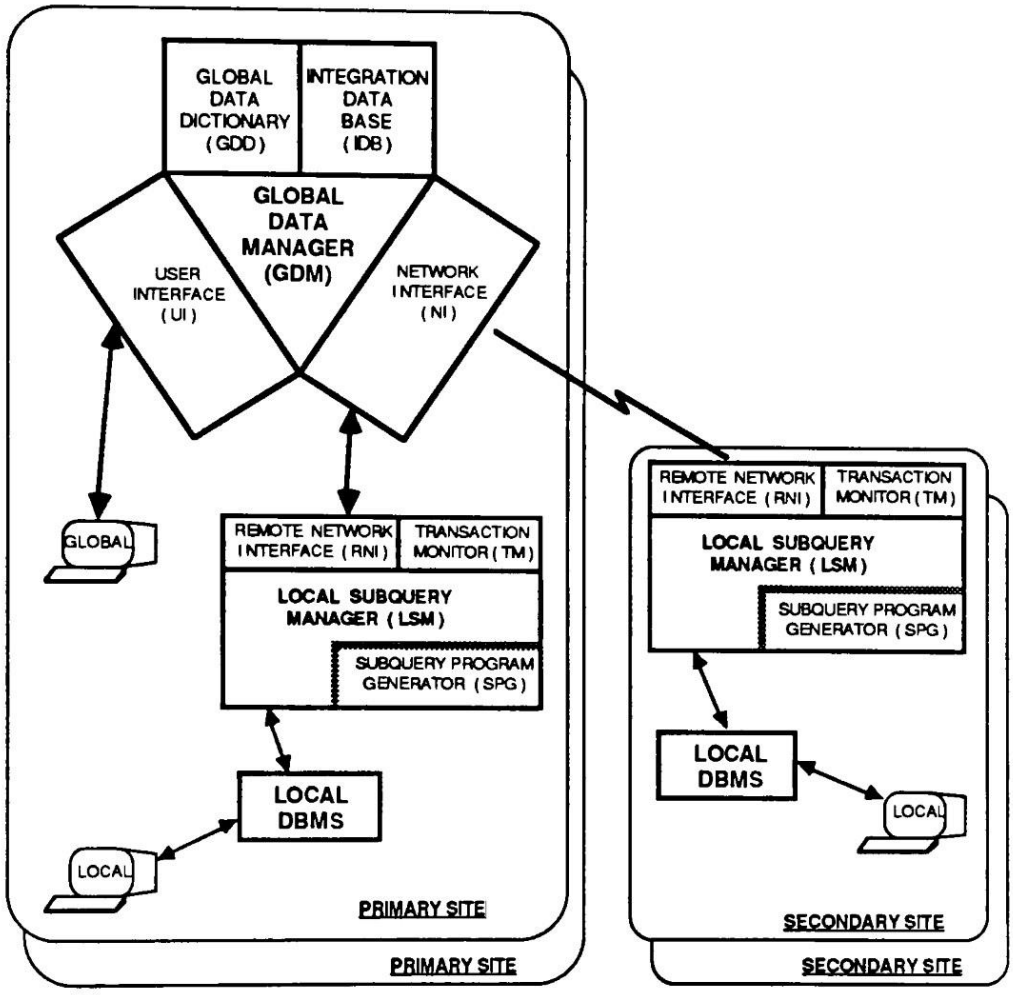


Figure 4.3 Architecture of System Components

Instead they are distributed throughout the network. They will concentrate on the sites generating the majority of global transactions and managing the most frequently accessed data.

4.4. Evaluation of the Proposed Architecture

The architecture described above is intended to be flexible and robust. To accomplish this requires the redundancy inherent in a distributed structure. In this manner multiple primary sites support global users of the HD-DBMS. This encourages distributed processing whenever possible and supports replication of data. However, with this type of architecture there are certain drawbacks.

Maintenance of the GDD and IDB is one of the more challenging tasks. Although changes to the database schemata occur infrequently, they do occur. In a distributed architecture, the global schema definition will exist at every primary site. Modifications made to the global schema must propagate to all the primary sites. This will insure that all sites have the same concept of the global schema. The IDB is affected, and must be treated, in the same manner.

Another concern with this type of architecture is the degree of system duplication. Every site that desires global database capabilities must support a GDBM. Additionally if a site supports the global database, it must have a LIM. This amount of duplication may seem excessive but it is necessary to provide the required reliability.

The load this amount of software may place on the sites is a

real concern. While many of the supporting sites may be secondary sites and not require the GDBM, they must still support their local DBMS and the LIM. Primary sites would be loaded the heaviest, supporting their local DBMS, a GDBM and a LIM. Depending on the CPU power and the configuration at a site (clustering, etc.) this may prove to be overwhelming.

How well this architecture performs would be best determined by implementing a prototype system. Three sites would be necessary to test all of the possible types of sites. One secondary site, one primary site and one site which has a GDBM but doesn't support a local DBMS or LIM. This type of configuration would provide valuable insight into a full implementation.

CHAPTER 5

SUMMARY

5.1. Introduction

In this research, the HD-DBMS has been defined and its components and structure analyzed. Background material related to distributed data base management systems was provided to familiarize the reader with the central issues currently faced in this field . Existing HD-DBMSs that have been constructed for research were presented for comparative purposes.

The issues specific to heterogeneous D-DBMSs were then examined in detail, important design decisions were introduced and their advantages and disadvantages discussed. This was followed by the proposal of an architecture for a HD-DBMS. The proposal included defining the components that need to exist, what their functions are and how they are structured in the system.

5.2. Motivation for a HD-DBMS

The issues have been presented and the requirements for the system have been stated. What advantages are to be gained with a HD-DBMS? Basically, distributed processing and data integration. This type of system provides the capability to obtain related data from different sources and integrate it to create a single uniform view.

One of the advantages of a D-DBMS, heterogeneous or not, is the ability to provide a degree of distributed processing. Each local DBMS can be simultaneously processing a portion of a global query. This distributes the load between many processors instead of a single site being relied upon to do all of the work. Another advantage of distributed processing is that local control of the remote processing occurs. This can be useful in a HD-DBMS where the site that is doing the actual processing usually "owns" the data.

Heterogeneity between DBMSs will always exist. Efforts may be made to standardize to one particular vendor or to use one data model, but this is unlikely to occur on a worldwide basis. In order to cope with this heterogeneous environment, a system must be designed to integrate dissimilar systems. The capability to translate between different data models, schemata and data manipulation languages is a fundamental requirement for database integration. The desire and

need to integrate dissimilar databases exists today and will not disappear.

Given the advantages of distributed processing and the presence of heterogeneity, where will HD-DBMSs be implemented? Within many organizations there exist internal islands of information that are related. Often, these islands were formed separately without a strategy for their future integration. To effectively manage the information and the organization itself, a tool such as a HD-DBMS proves invaluable.

The heterogeneity of such a system permits the integration of data from a variety of sources. Its distributed processing properties promote the sharing of the work load. Combining these features provides a robust and flexible system, the type of system that can be expanded in the future.

5.3. Feasibility of Implementation

Can this type of system be implemented? The answer is unequivocably, yes. In addition to the systems presented within this paper, other research projects such as PRECI* [DEEN85] and Integrated Information Support System (IISS) [HUBE86] are underway to build HD-DBMSs.

Implementation Issues

To implement a HD-DBMS, key design issues must be identified and addressed. Once adequate solutions to these issues have been developed, construction of the system can take place.

One of the first decisions would define what type of the system architecture will be used. Both of the centralized and distributed structures have their advantages and disadvantages. The type of structure chosen will determine critical characteristics of the system such as concurrency control and management of the global data dictionary.

What global data model and corresponding data manipulation language would best support the HD-DBMS is another important decision. Using the relational model has been a popular approach due to its flexibility. A functional data model and manipulation language such as DAPLEX [SHIP81] is yet another choice. The capabilities of the data model and its manipulation language will determine which data models can be supported and how extensive their support will be.

Integration of dissimilar data models and their schemata is at the heart of the whole heterogeneity issue. Accurate integration requires careful planning and development of the translators and the integration database. The difficulty lies in the manual definition of how each schema merges with

another. Once the mappings necessary for integration have been defined, the mechanism used for integration can be automated.

Although these three critical issues have been addressed, their solutions do not come easily. HD-DBMSs can be designed and implemented, but providing users with the transparency, flexibility and performance they require is still a long sought after goal.

Proposed Architecture

The architecture proposed in chapter 4 is an outline of how the author would construct such a system. It defines the components of the system, their functions and structural relationships.

The distributed system structure used supports a flexible and robust architecture for a HD-DBMS. It permits the system to operate independent of the degree of network partitioning. As long as the data required by a query can be obtained, a distributed database system will still be supported.

A disadvantage of this type of design is its complexity. Permitting all of the nodes in the distributed system to perform the processing means redundancy of code and of data.

Each node must be capable of managing the transactions for the subqueries it generates and receives. It must also be able to translate the global query and optimize its execution.

Though not defined in chapter 4, the global data model for the proposed architecture should be relational. Additionally the manipulation language would be based on SQL. These choices are based on the use of proven technology and provide for future developments.

Integration is supported by each primary site using the Global Data Dictionary and Integration Data Base. Even with these components, the Global Data Base Administrator must still define the initial mappings between schemata.

Sites supporting this type of architecture will most likely have to meet minimum processing requirements. These could include processing power/speed, functional capabilities and network connectivity. This design also increases the problems associated with concurrency control because of decentralized query management. Multiple sites may be trying to gain access to the same data simultaneously. Nevertheless the advantages of this architecture appear to be worth its complexity and overhead. How well this type of design will work, may be best determined by implementing a prototype.

Other Systems

Systems such as SIRIUS-DELTA and MULTIBASE have already demonstrated the feasibility of a HD-DBMS. Although conceived several years ago, their development has been a continuing effort.

The SIRIUS-DELTA prototype demonstrated the trade-offs that are made when implementing a D-DBMS. It provided researchers with valuable insight into problems and issues that arise in a distributed system. They gained practical experience in the design, development and implementation of a HD-DBMS. One of the realizations was the use of a "pivot" or global system. This provided a focal point for translations and schema integration. The SIRIUS-DELTA prototype verified that a heterogeneous system was feasible.

Creation of the MULTIBASE system also yielded valuable results. Although this system did not support distributed update capabilities, it did focus on the key issue of integration. It proposed the use an integration database for resolving conflicts between schemata. A standard user interface, logical schema integration and global query optimization were also features that were developed and implemented.

The AIDA system (section 2.5) has demonstrated its feasibil-

ity as well. However restrictions have been placed on interfacing to pre-existing DBMSs. AIDA requires that the local DBMSs support a relational interface. This criteria has simplifies the task of mapping between data models, but limits the types of pre-existing DBMSs that can be supported.

Up to this point, HD-DBMSs have concentrated on the translation and integration functions. Concurrency control, recovery and update capabilities have, in some cases, been reserved as a second milestone. These goals will not be easy to achieve. With the proliferation of different DBMSs, constraints must be set on what degree of heterogeneity will be supported.

Unfortunately, producers of centralized (local) DBMSs may not have the incentive to develop HD-DBMSs. The heterogeneous nature of the system implies integration with other (competitor's) DBMSs. Still, the desire for HD-DBMSs exists in organizations that have pre-existing DBMSs. Development and implementation of HD-DBMSs has been conducted in research environments, often being funded by government agencies. With data management becoming a critical issue in all types of organizations, it is only a matter of time until HD-DBMSs become commercially available.

5.4. Future Trends

Ten years ago initial development began on the first HD-DBMS. Since that time, many problems and solutions, have been identified. With HD-DBMS such as those mentioned above being developed, what is the next step?

In a recent publication [MYER86], it was stated that Relational Technology Inc., producers of the relational DBMS INGRES, are developing a distributed version of the product called INGRES* (INGRES Star). This in itself is not surprising, what is interesting are the end goals. Initially, the product will operate in a homogeneous distributed manner. An ultimate goal of INGRES* is to provide users with gateways to other DBMS. This is one way that heterogeneity is being introduced into products.

There are other benefits to be gained. Advances in network communications, management of distributed processes and interprocess communication may be driven by the research in HD-DBMS. Developing an application package such as this deals with many aspects of computers.

Looking even further in the future there are other possibilities. Expert systems have been maturing and are becoming commercially feasible. Their ability to manage rules could prove to be a good source of control for DBMSs. Business

rules that apply to the information residing in a database, may be more easily expressed and enforced by an expert system. This could be extended when applied to a HD-DBMS. An expert system could be used for global query management as well as rule management.

Abstract data types and object oriented systems represent other forms of data structures that are being investigated. Since information can exist in many forms (voice, image, knowledge, text, etc.), it is likely that DBMSs will be useful for their management. Keeping in mind that a heterogeneous distributed data base management system must be flexible and expandable. Support of new data structures and corresponding operators may well be a requirement in future HD-DBMS.

As the technology to store and manage information develops, it becomes increasing complex. Heterogeneous Distributed Data Base Management Systems will remove barriers between dissimilar systems and aid in the management of information.

Appendix I

GLOSSARY

ARIEL - a data manipulation language developed by System Development Corporation for user interaction with the AIDA system [MACG85].

ARPANET - a computer network created by the Advanced Research Projects Agency (ARPA) of the U.S. Department of Defense. It extends from Hawaii to Norway and connects over 100 computers.

Concurrency - two or more transactions occurring simultaneously within the same database.

CICS - Customer Information Control System, a product of IBM used to communicate to data base management systems.

CIM - Computer Integrated Manufacturing, the use of computer technology to automate the design, development and manufacturing processes involved in producing a product.

CMS - an IBM product, a single user interactive operating system.

CODASYL - COnference on DAta SYstems and Languages, developed COBOL and the 1971 Data Base Task Group (DBTG) standards.

CPU - Central Processing Unit, controls and supervises operations in the computer.

DAPLEX - a functional data definition and manipulation language developed by the Computer Corporation of America [SHIP81].

Database - a collection of occurrences of a number of record types. The record types are interrelated via relationships.

Data Dictionary - a collection of definitions describing the structure, storage and type of information residing in the database.

DBA - Data Base Administrator, the person or persons responsible for defining the schema, integrity constraints and performing other administrative tasks associated with the control and management of a database.

DBMS - Data Base Management System, a software system capable of supporting and managing a data base.

D-DBMS - Distributed Data Base Management System, a data base in which portions of the data are stored separately.

DIL - Data Intermediate Language, developed by Computer Corporation of America.

DML - Data Manipulation Language, used to define the data structure and/or modify the data in a data base.

EBCDIC - Extended Binary Coded Decimal Interchange Code, developed and used by IBM.

ethernet - a coaxial cable network in which all nodes monitor constantly.

GDBA - Global Data Base Administrator, the individual responsible for the definition of the global database schemata (see DBA).

GDBM - Global Data Base Module, a portion of the proposed architecture that is the combination of the GDM, GDD, IDB and UI components.

GDD - Global Data Dictionary, the data dictionary that stores information about the global database (see Data Dictionary).

GDM - Global Data Manager, a component of the distributed DBMS that acts on the entire collection of databases.

HD-DBMS - Heterogeneous Distributed Data Base Management System

IDB - Integration Data Base, the database that stores information needed to integrate dissimilar schemata at a global level.

IDM - Intelligent Database Machine, produced by Britton-Lee Inc, it is a machine dedicated to database management.

IMS - Information Management System, a hierarchical DBMS produced by IBM.

INGRES - a relational DBMS, initially developed at University of California at Berkley, now produced commercially by Relational Technology Inc..

INRIA - Institut National de Recherche d'Inflormatique et d'Automatique, a national research institute in France.

ISC - Inter System Communication, an IBM product that is a set of protocols which allow any (IBM) systems to communicate.

ISO - International Standards Organization.

LAN - Local Area Network, usually short distance (i.e. less than a mile).

LIM - Local Interface Module, a portion of the proposed architecture (section 4.3.1.2) that represents the GDBM at remote sites.

LSM - Local Subquery Manager, a component of the LIM that manages subqueries at the remote site.

MVS - an IBM product, a batch oriented operating system.

NEXUS - the user support environment in the AIDA system [TEMP86].

NI - the Network Interface, manages communication between a local GDM and a remote GDM.

POLYPHEME - a joint venture project between the University of Grenoble and the CII-Honeywell Bull Scientific Center [LEBI80].

QUEL - a relational data definition and manipulation language, developed for use with INGRES.

query - a request made to the DBMS. This includes insertion and update of data as well as retrieval.

RNI - Remote Network Interface, a component of the LIM that is responsible for all network communications between the GDM and LSM.

SAP - Subquery Application Program, a program generated by the SPG component to process a specific subquery.

schema - the logical description of the database. It describes the fields in the database, their name, type and relationships.

SDC - System Development Corporation, developers of AIDA.

SPG - Subquery Program Generator, a subcomponent of the LSM used to create an application program in which a subquery is embedded.

SQL - Structured Query Language, a relational data definition and manipulation language used with SQL/DS and DB2, relational DBMS produced by IBM.

timestamping - a method to uniquely identify the point in time a process took place.

TM - Transaction Monitor, a component of the LIM that monitors any active transactions currently managed by the LSM.

transactions - an atomic unit of execution, (i.e. a sequence of operations either entirely performed or not performed at all).

tuple - a record, a group of related field values in a relation or table.

UI - User Interface, a component in the GDBM that provides a uniform user interface to the HD-DBMS.

VAX - Virtual Address eXtension, a model of computer produced by Digital Equipment Corporation.

view - a logical description of a subset of the database.

Appendix II

About the Author

The author received his Bachelor's Degree in Mechanical Engineering Technology from Rochester Institute of Technology in 1979. After five years as a mechanical design engineer, he moved into an engineering computing environment involving mechanical CAD/CAM application software.

He began the graduate program in 1982. His concentration has been in the Database and Data Communications areas. Courses included Data Base Management Systems, Data Base System Implementation and Data Communications and Networks I and II.

His professional interests currently focus on databases, data base management systems and understanding the mechanics of integrating them in the Engineering and Business environments. He is currently working in the Manufacturing Technology Research Laboratories at Eastman Kodak Company.

BIBLIOGRAPHY

[ADIB80] Adiba, Michel E., Lindsay, Bruce G., "Database Snapshots", Proc. 6th International Conference on Very Large Data Bases, October 1980, pp 86-91

[ANSI76] ANSI/SPARC DBMS Model: Proc. of 2nd SHARE Working Conference, (Donald A. Jardin, Ed.) Elsevier Science Publishers B.V. (North-Holland), April 1976

[APPL85] Appleton, Daniel S., "The Technology of Data Integration", Datamation, Technical Publishing Co., November 1, 1984, pp. 33-43

[BILL78] Biller, H., Neuhold, E.J., "Semantics of Data Bases: The Semantics Data Models", Information Systems, Vol. 3, London, Great Britain, Pergamon Press Ltd., 1978, pp. 11-30

[BILL79] Biller, Horst, "On the Equivalence of Data Base Schemas- A Semantic Approach to Data Translation", Information Systems, Vol. 4, London, Great Britain, Pergamon Press Ltd., 1979, pp. 35-47

[CARD85] Cardenas, Alfonso F. Data Base Management Systems, Boston, Massachusetts, Allyn and Bacon Inc., 1985

[CERI84] Ceri, Stefano, and Pelagatti, G., Distributed Databases Principles and Systems New York, New York, McGraw-Hill, Inc., 1985

[CCA78] Computer Corporation of America, Datacomputer Version 5 User Manual, Cambridge, Mass., July 1978

[CODD70] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", CACM 13, No. 6, June 1970

[DATE83] Date, C. J., An Introduction to Database Systems, Volume II, Reading Massachusetts, Addison-Wesley Publishing Co., 1983

[DEEN85] Deen, S. M; et al., The Architecture of a Generalized Distributed Database System - PRECI*, The Computer Journal, 28, 3, (March 1985), pp. 282-290

[EFFE84] Effelsberg, W., and Mannino, M., "Attribute Equivalence in Global Schema Design for Heterogeneous Distributed Databases", Information Systems, vol. 9, No. 3/4, 1984, pp 237-240

[ESCU84] Esculier, Christian, "The SIRIUS-DELTA Architecture: A Framework for Co-Operating Database Systems", Computer Networks, Vol. 8, 1984 pp 43-48

[FERR82] Ferrier, Arlette, and Stangret, Christine, "Heterogeneity in the Distributed Data Base Management System SIRIUS-DELTA", Proceedings of the Eighth International Conference on Very Large Data Bases, September 1982, pp 45-53

[GLIG84] Gligor, Virgil D., and Luckenbaugh, Gary L., "Interconnecting Heterogeneous Database Management Systems", Computer (IEEE), January 1984, pp. 33-43

[HUBE86] Hubele, Norma F., and Wilson, Dennis, "IISS Testbed and Beyond" Proceedings of the Conference "CAD/CAM Databases '86: Control for the Decade Ahead", Framingham, Massachusetts, April 14-15, 1986

[LAND82] Landers, Terry, and Rosenberg, Ronni L., "An Overview of MULTIBASE", Distributed Data Bases, (H.J. Schneider, Ed.) Elsevier Science Publishers B.V. (North-Holland), 1982, pp. 153-183

[LEBI80] Le Bihan, J., et al., "SIRIUS: A French Nationwide project on Distributed Databases", 6th International Conference on Very Large Data Bases, October 1980, pp 75-85

[MACG85] MacGregor, R., "ARIEL - A Semantic Front-End to Relational DBMSs", Proceedings of VLDB 85, Stockholm, August 1985

[MCGE74] McGee, W.C., "A contribution to the study of data equivalence", Data Base Management, (J.W. Klimbie and K.L. Koffeman, Eds.) North-Holland Publishing Co., 1974

[MYER86] Myers, Edith, "No DBMS is an Island" Datamation, Technical Publishing Co., June 1, 1986

[ROTH80] Rothnie, J.B., et al, "Introduction to a System for Distributed Databases (SDD-1)", ACM Transactions on Database Systems, Vol. 5, No. 1, March 1980, pp 1-17

[SHIP81] Shipman, David, "The Functional Data Model and the Data Language DAPLEX", ACM Transactions on Database Systems, 6, 1, March 1981 pp 140-173

[SMIT81] Smith, John M., et al, "MULTIBASE - Integrating Heterogeneous Distributed Database Systems", AFIPS conference proceedings, vol. 50, 1981, pp 487-499

[TEMP83] Templeton, M.; et al, "An Overview of the Mermaid System - A Frontend to Heterogeneous Databases", IEEE Electronics and Aerospace Conference Proceedings (EASCON), 1983, pp. 387-402

[TEMP86] Templeton, M.; et al, "An Introduction to AIDA - A Frontend to Heterogeneous Databases", (private communications, IEEE publication pending), Jan. 1986

[ZLOO75] Zloof, M.M., "Query by Example", Proceedings, 1975 National Computer Conference, Anaheim, Ca., May 19-22, 1975