1995

Presentation of consistent information from independent databases

Pattarasinee Bhattarakosol
University of Wollongong

Presentation of Consistent Information from Independent Databases

A thesis submitted in fulfilment of the

requirements for the award of the degree of

Doctor of Philosophy

(Computer Science)

from

THE UNIVERSITY OF WOLLONGONG

by

Pattarasinee Bhattarakosol, B.Sc., M.Sc.

Department of Computer Science

1995

This is to certify that the work detailed in this thesis was done by the author, unless specified otherwise in the text, and that no part of it has been submitted in a thesis to any other University.

Pattarasinee Bhattarakosol

Abstract

This thesis addresses a particular aspect of the retrieval of information from a wide variety of global information sources. This aspect is based on the model of a user working on a topic of interest over an extended period of time. During the time period, information is accessed, assembled, and correlated to satisfy the user's view of the topic.

The objective of the research described in this thesis is to ensure that the information accessed by the user over the extended period of time is both complete and consistent from the user's viewpoint.

A fundamental problem is achieving such consistency is that the information sources, typically databases, or data files, are independently controlled with their own individual viewpoints. Changes in content, structure, and access can therefore be made without the direct knowledge of the user.

In investigating this problem a number of current implementations of Heterogeneous Distributed Database Systems (HDDS) have been evaluated, including WAIS, ANSAware, and DATAPLEX. The mechanisms available in such systems do not address the requirements of the problem outlined in this thesis. A new set of mechanisms have been researched and implemented on a testbed as the central part of this thesis in order to match the requirements.

The core of this testbed is a workstation-based interface for the user termed the Computer Software Interface (CSI). The CSI has been implemented to demonstrate that the set of mechanisms proposed are viable. One major aspect of the CSI has been the design and

development of a local working environment for the user, and the associated theoretical proof needed to demonstrate that successful and complete access of this environment may be performed.

The thesis demonstrates, both theoretically and practically, how the user may be presented with consistent data from independent data sources over an extended time period.

Acknowledgements

I wish to express my gratitude to my supervisor, Professor Fergus O'Brien for his constant support and valued guidance. Dr. Janusz R. Getta has provided most valuable assistance in the field of database systems. I must warmly thank too, Dr. Peter R. Nickolas for sharing his understanding of the theoretical significances which are the basis of this study, Associate Professor Greg Doherty and Associate Professor Neil Gray in they great efforts to help me re-writing this thesis. Invaluable support was also giving by Mr. Phil Herring with the Oracle System and Mr. Sam K. Tan with HyperCard.

My gratitude also extends to the staff of the Department of Computer Science for their help and encouragement. Finally I wish to thank my family and friends who have helped to give me the strength and determination to finish this work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

This chapter will describe the system design parameters of a workstation based interface for handling data from a variety of sources, which are subject to possible unreliable network connectivity. The original problem was based on the heterogeneous design of seven major databases within the data processing environment of Telstra, where Telstra has been trying to solve the problem by combining the databases, but this is still under development, and this approach has yet to be proven. The objective is to develop a Computer Software Interface (CSI), which will facilitate data retrieval from a Heterogeneous Distributed Database System (HDDS) which consists of various kinds of data management systems (databases and files), which are independently owned. There are two different cases to be considered. The first involves databases which are being continually updated. In the second, it is anticipated that the data will be accessed by a user over extended time periods, during which the data may be updated at its source, or access to the primary source may disappear intermittently.

1.1 Overview

In accessing the various data sources, a number of problems may arise, ranging from communication link failure through to changes in data schema at the data source. In the past,

proprietary operating systems and communication protocols made shared access difficult, but the adoption of open systems and TCP/IP as a connection mechanism has greatly improved prospects for sharing data across systems. Higher level protocols have been implemented to facilitate information retrieval processes in particular situations, such as the Z39.50 [NISO,1988] originally for bibliographic retrieval.

In the situations that we wish to consider, although correctness of transferred data is guaranteed by TCP/IP, it is possible that during the data communication process communication failure or some other failure at the source might occur. Such problems may cause the user to obtain incorrect results, or impede further progress. A mechanism is given for bringing together such data from different sources, and making sure that the data from different sources can be assembled. Methods of achieving data integrity are discussed in this work.

1.2 Motivation

A number of software systems have been developed to provide access to heterogeneous distributed databases, providing full rights for users in the system to be able to update data, delete data, add new data, or read data. Software such as DATAPLEX [CHUNG,1990], and Pegasus [AHMED et al.,1991], manage all types of transactions such as retrieval, updating, and so on. Some software such as the Wide Area Information Servers (WAIS) [KAHLE,1989] provides only the retrieval transaction which is entered by a user using a WAIS user interface [KAHLE et al.,1992]. Yet another system is ANSA [APM(2),1991], which provides commands that are embedded into the user application program and interact with the data holder system to gain access, thus shielding the user from dealing with all difficulties in handling different access methods. Such software is running under the assumption that the entire system is stable, and that there is no error in data transmitted between transaction processes. At the

moment, there is a huge growth in Internet traffic related to the World Wide Web, WWW [BERNERS-LEE et al.,1994] which can provide hypertext links between different data sources prepared using the HTML markup language.

One situation that will be discussed in this thesis is the original problem of a problem that has occurred in Telecom Australia. Telecom Australia has many subcompanies and departments, each of which manages itself and installs its own databases, and they use the telephone number as a key to link data from every database in the different Telecom companies. Consider the management system in the Telecom departments, such as the Telephone Installation Department, the Service Department, and the Billing Department. Each department has installed different database systems. The Telephone Installation Department installed DB2 on an IBM system, the Service Department installed IDMS on a BULL system, and the Billing Department installed ORACLE on a UNISYS system. This environment is a classic heterogeneous distributed database system.

The data in the Telephone Installation Department include the telephone number, name and address of the owner of the phone. The data in the Service Department will be the telephone number, numbers of local calls, STD and international calls. The data in the Billing system will be the telephone number and the details of number of calls, last payment, debt from the last service charged, and the current charge for this period. So, the bill issued from the Billing system must obtain the name and address of the owner of the phone from the DB2 in the Install Department, the number of services from the IDMS of the Service Department, and include this information with the data in its ORACLE database to prepare a bill for each telephone number.

Consider the situation that a telephone number has been transferred to another owner, and the details of the new owner have been updated on DB2 and IDMS, but not ORACLE. In

this case, when the bill has been prepared and sent to the address obtained from the DB2 with the service details from IDMS, the bill will be sent to the wrong person with the wrong information because it will include the previous debt of the previous owner. From Telecom's perspective, such an error arises because the different databases have become "inconsistent". Such inconsistencies seem to occur too frequently in these distributed environments where many separate agencies are responsible for updating different parts of the total data.

A different type of problem occurs in the retrieval process for information in a HDDS in which the data from each data source will be updated occasionally. The retrieval process that retrieves data relevant to a certain period we refer to as a *sustained task.* Suppose that a cable TV company in JAPAN has many representatives in different countries such as the USA, CANADA, the UK, and FRANCE. Each representative has authority to implement their own database and control their local data. Suppose that the headquarters in JAPAN would like to check the occupational status of cable TV subscribers in each country to find their main preferences. This will constitute the sustained task to be performed. The data from each database in the HDDS needs to be assembled, so that the senior manager in JAPAN can see the differences in preferences between groups of subscribers in different countries.

Suppose that the head office in JAPAN announces a new customer service to all members, such as an offer to switch from using cable to a satellite dish. This offer will be made in every country, so each subsidiary company has to update its data on customers who take up the offer, in their local databases. If a member does not want to change to the new service for some personal reason, that member can still use cable. Once the new service has been available for a time, the senior manager in JAPAN would like to see the number of members who have converted to the new service. Some databases in some countries may have been updated, while others have not. Thus, at this moment, data across the various databases are not consistent. Therefore, when the director in JAPAN asks for information on the use of the new service, the

result will only show services from the updated repositories, and no new services on those which have not been updated. This is a relatively common form of inconsistent information, and is the sense in which we use inconsistent in this thesis. The problem occurs during the update period because the data repositories are independent, so it is possible that one or more databases are not updated at the same time. A query across more than one of them may obtain inconsistent results.

One possible approach in dealing with these problems would be the introduction of some form of "intelligent agent", which would monitor or control a user's access to the combined databases. If the agent detected a potential inconsistency among the data sets accessed, it would warn the user or take other corrective action. A general solution is impractical, there are too many ways that data can be "inconsistent". However, some common problems seem to admit a relatively simple solution. If the "intelligent agent" can identify when data were last changed, and has some knowledge of likely lifetime of data, then it can use this information to detect several common problems.

The idea of an intelligent agent that can utilize "timestamps" on data was the starting point for the work in this thesis. The investigation aimed to explore a number of related problems that might be dealt with using an intelligent agent. These problems include incomplete updates of data being unavailable (would the last used value still held locally be good enough or must the query fail?), and other problems related to the maintenance of local data caches.

1.3 Objective

The objectives were identified as:

1. explore the scope of application of an agent using timestamps

2. implement a prototype version realizing a simple agent through programs

3. evaluate the performance and utility of the implemented system.

1.4 Scope and Problem Domain

We are interested in the total set of information spread across a number of distinct and separately owned databases. The main part of our research is to build a prototype HDDS dealing with information owned by different owners, in different databases possibly with different data manipulation languages, data definitions, etc., where, nevertheless, information across the network should be consistent, as defined above. The assumptions underlying our work are as follows.

1. End users are not necessarily owners of a database in the HDDS, so they can only retrieve data, but from more than one database at a time.

2. Mechanisms such as concurrency control, security control on servers, and deadlock detection and recovery need not be discussed in this thesis, since we are dealing with read access only.

3. Problems of semantic heterogeneity will not be considered, but the problem of syntactic heterogeneity will be addressed.

4. The data in the databases is assumed to be textual.

We will focus on the problems that affect the correctness of integrated data in the presence of updates to individual databases.

1. Whenever a server changes its services or updates its data, how can our database management system know that a change has occurred at the server site and inform users?

2. When a user uses the local database which duplicates data from a remote database in the HDDS, how can the database management software verify that the data in the local database is the same as that at the remote site?

3. How can the HDDBMS continue its service to a user when a disconnection between a client and a server occurs?

1.5 Work Undertaken

This section outlines the research undertaken, the results of which are described in the remainder of the thesis. The following sections describe processes, methodologies, and a simulated heterogeneous distributed data system which has been implemented.

1. Study of computer systems.

2. A general study of a file systems and relational database systems.

3. Study of the characteristics of TCP/IP, MacTCP and HyperCard.

4. Study of HyperCard scripts, Hyper*SQL and MacTCP toolkit.

5. Creation of files and a database on *servers* [1]: Server1 and Server2.

[1] *Server* is a computer system that to accept an incoming connection, to provide a set of defined services.

6. Use of a Macintosh as a *client* [2] workstation to implement programs which retrieve data from the file on Server1 and the database on Server2, using TCP/IP and HyperCard. The data retrieved from both systems are merged and shown on the client screen. This provides a fully functional prototype of the heterogeneous data base idea.

7. Simulation of the situation of communication failure and changes in the definition of data held in files and in the database.

8. Identification of problems arising in the previous process.

9. Proposing solutions for the problems identified, and implementation of a prototype program.

10. Analysis and evaluation of the outcome of using the program.

1.6 Introduction to the Research Methodology

The programs we implemented are as follows.

1. A C language program - running as a *server*, on Server1. The server program uses the fork-exec mechanism [SUN,1988] to connect different clients as they arrive for service.

2. A HyperCard program running as a *client program*, on a Macintosh connected to the ethernet. The client program contains a new mechanism developed from a combination of the logging mechanism [CERI & PELAGATTI,1985] and the

[2] *Client* is a computer system that asks for connection, it will connect to a server.

duplicate database mechanism, [GOLDING,1992] to serve users when the communication breaks down.

3. A trigger[3] program in the RDBMS to record the last update time of data in a table. Some tables also have the update time of individual records as a field.

1.7 Computer Systems

The computer systems in this trial consist of two SUN servers, Server1 and Server2, and one Macintosh. The Macintosh is a client workstation. The TCP/IP set of protocols are used to transmit data between the client and servers. See Figure 1.1.

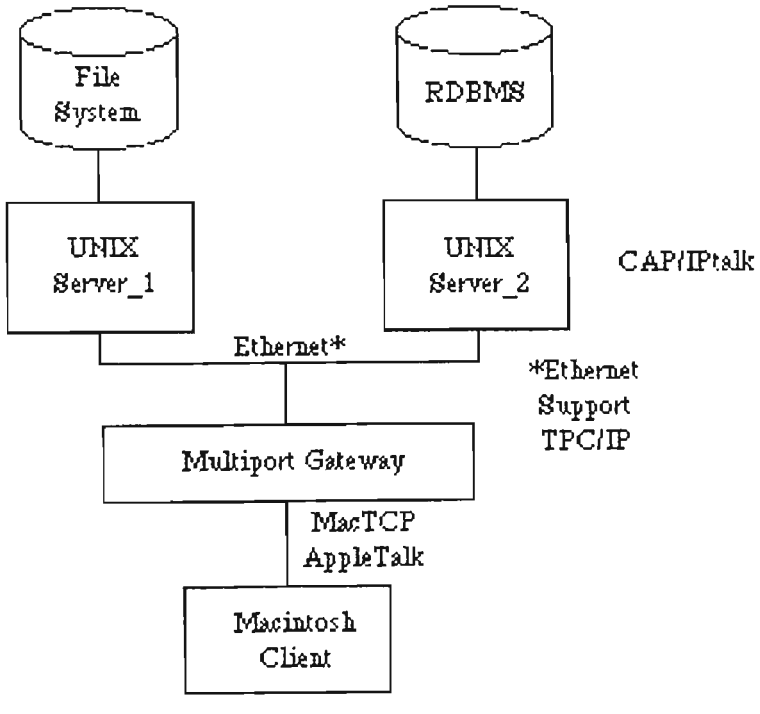


Figure 1.1 A Computer Network System.

[3] Trigger is a command in the Oracle database to perform actions such as update data in some specific table after or before a transaction gains access to data in a particular table.

The data are artificially split into two components, one component on each server. One part is a sequential file, and stored on Server1 using a random access algorithm. The other part is recorded on Server2 using the ORACLE relational database management system. When the client requests information from either server, server programs send information back to the Macintosh. The server program on Server1 is a C program which accesses its data files, and uses TCP/IP to transfer data to the client. The server program on Server2 is an ORACLE PRO*C program which also uses TCP/IP to transfer data to the client.

The client program consists of *HyperTalk scripts*. It is separated into two conceptual components. The first part retrieves data from the database system using embedded *Hyper*SQL* from Server2 The second part, which uses TCP/IP utilities, is the *HyperCard TCP toolkit*. which initiates commands to retrieve data from a file system on Server1. The data from both servers is merged and shown on the client screen.

Communications between the client and servers take place through a multiport gateway. The multiport gateway performs the necessary encapsulation of AppleTalk packets into TCP datagrams, shown by Figure 1.1. Communication between the multiport gateway and the client uses MacTCP [APPLE-MACTCP,1988]. The protocol between the multiport gateway and servers is TCP/IP which runs over Ethernet.

We did not use the alternative communication mode described below in which the Macintosh is relegated to the role of a dumb terminal. In Figure 1.1, CAP is a software package which runs on a UNIX host which lets the host utilise the AppleTalk encapsulated in IP packets transmitted over the Ethernet. The information transferred from a UNIX host to a Macintosh using CAP/IPtalk will be sent via an AppleTalk Ethernet Gateway such as a Webster multiport gateway. At the multiport gateway, an IP packet is unwrapped to obtain AppleTalk packets to be sent to a Macintosh. When a Macintosh wants to send data or information over the Ethernet,

AppleTalk packets will be wrapped at the multiport gateway and sent via the IPtalk to the UNIX computer.

1.8 Database and File Systems

The sample data loaded onto the servers were extracted from the "FAO Product Year-book: Food and Agriculture Organization of the United Nations" [FAO(1),1986] [FAO(2),1987] [FAO(3),1988]. The data forms of the original tables were separated and stored in two systems: an ORACLE relational database on Server2, and some data files on Server1. The data on Server2 is obtained from the Land Area Table, Table 1.1, and this table is implemented in ORACLE with eight columns. The table name in the database system is FAO_DATA, consisting of columns named COUNTRY, YEAR, TOTAL_AREA, ARABL_LAND, LAND_AREA, PERM_CROPS, PERM_PASTURE, FOREST_WOODL. The structure of data on Server2 is shown in Figure 1.2.

The original data from four tables, Table 1.2, which are Agriculture Table, Alimentaries Table, Cereals Table, and Cultures Table will be stored individually in the file systems. The file system was created on Server1 as a sequential file, using a random access method to retrieve data. The data of each original table is separated into two files. The first is an index file containing a key and an index pointer to data in the other file. The second contains the real data corresponding to the key in the previous file. The structure of data in these files is shown in Figure 1.3 and Figure 1.4. On Server1, there are four data file systems: Agriculture, Alimentaries, Cereals, and Cultures.

Land Area Table

Country	1985	1986	1987
Hong Kong			
Total Area			
Land Area			
Arab&Perm CR			
Arable Land			
Perm Crops			
Perm Pasture			
Forest Woodl			
Other Land			
India			
Total Area			
Land Area			
Arab&Perm CR			
Arable Land			
Perm Crops			
Perm Pasture			
Forest Woodl			
Other Land			
.			
.			
.			

Table 1.1 The Form of the Original Data Stored in the ORACLE

	Alimentaries Table										
Country	1977	1978	1979	1980	1981	1982	1983	...	1986	1987	1988
Hong kong India ⋮											

	Agriculture Table										
Country	1977	1978	1979	1980	1981	1982	1983	...	1986	1987	1988
Hong kong India ⋮											

	Cereal s Table										
Country	1977	1978	1979	1980	1981	1982	1983	...	1986	1987	1988
Hong kong India ⋮											

	Cultures Table										
Country	1977	1978	1979	1980	1981	1982	1983	...	1986	1987	1988
Hong kong India ⋮											

Table 1.2 The Form of the Original Data Stored in the File Systems.

```
CREATE TABLE  FAO_DATA
(
     COUNTRY         CHAR(15)      NOT NULL,
     YEAR            NUMBER(4)     NOT NULL,
     TOTAL _AREA     NUMBER(8),
     LAND_AREA       NUMBER(8),
     ARABL_LAND      NUMBER(8),
     PERM_CROPS      NUMBER(4),
     PERM_PASTURE    NUMBER(7),
     FOREST_WOODL    NUMBER(7)
)
```

Figure 1.2 Data in the Relational Database System.

```
Struct  File_index
{
     char       Country[15];

     long int   Offset;
}
```

Figure 1.3 The Format of an Index File.

```
Struct  File_Rec
{
     unsigned int     Year_1;

     float            Data_1;

     unsigned int     Year_2;

     float            Data_2;

     unsigned int     Year_3;

     float            Data_3;
}
```

Figure 1.4 The Format of a Data File.

1.9 Summary

This chapter has given an outline of the objectives, the system environment, and the tools which will be used in this research. The operating environment is assumed to be one in which users work in an uncontrolled environment where they have no authority to control any alteration of data in the system, and where communication between them and the data sources may be lost. The data values are time dependent data which should be synchronised by the client program for presentation to the user. Our objective is to explore a system software prototype to meet these constraints.

The prototype system for the HDDS contains one Macintosh and two SUNs. The Macintosh is used as a workstation and a client. The SUN systems are used as servers. The data sources are a relational database system and a file system, each of which is located on a different server. The protocols which are used to transmit data are TCP/IP and MacTCP. The language used to implement the client software is HyperCard scripts running on Macintosh. HyperCard scripts consist of Hyper*SQL, HyperTalk, and HyperCard TCP toolkit statements. Data are separated into two different data holder systems, file systems and the relational database system.

This prototype system includes all the functionality contained in a generalised HDDS, and can be used to demonstrate the viability of the approach adopted.

Chapter 2

Literature Reviews with Some Example Approaches to Implement Heterogeneous Distributed Database Systems

The need to use data from distributed database systems is increasing. Data may be stored in different repositories, located at distinct addresses over a network. This leads to the development of a system called a Heterogeneous Distributed Database System or HDDS. System software that can access data over the HDDS has to take many factors in account. Here we summarise some of the design constraints and solutions which have been proposed by some authors, and present a summary of three designs for Heterogeneous Distributed Database Systems (HDDS). The systems under discussion are DATAPLEX, Wide Area Information Server (WAIS), and Advanced Network Systems Architecture (ANSA). The main reason for looking at these three is to illustrate the diversity of design for similar objectives under different constraints. The DATAPLEX approach emphasises database access aspects, WAIS is an information system approach to bridging the heterogeneity of information located at physically distant sources, and ANSA is a more general approach to distributed programming.

2.1 Introduction to a HDDS

A Heterogeneous Database System (HDS) is a system that may contain various kinds of database systems such as relational databases, hierarchical databases and network databases [BREITBART,1990]. Each of these systems maintains control over its individual DBMS participating in the federation. A Heterogeneous Distributed Database System (HDDS) is characterised by the distribution of the component databases over the network. The component databases may have different data models. The system which manipulates all accesses to databases in the HDDS is referred to as a Heterogeneous Distributed Database Management System (HDDBMS). Furthermore, heterogeneities in HDDS can be classified into two different categories: syntactic and semantic.

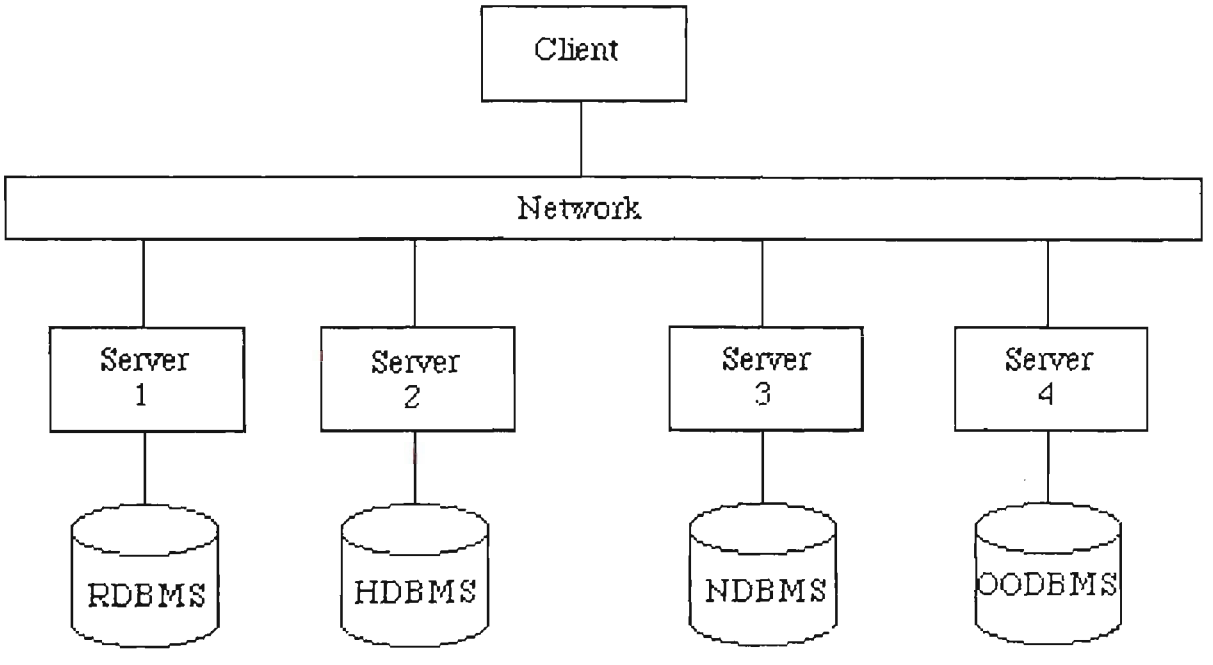


Figure 2.1 A Schematic of a Heterogeneous Distributed Database System.

Syntactic heterogeneity refers to differences in data models, or data schemas of the component databases. The component databases participating in the federation can include

databases such as Relational data model, Hierarchical data model or Network data model. This leads to the differences in the Data Manipulation Languages (DML) of the component databases. Each database system has its own DML to access data, for example RDBMS uses SQL statements, and NDBMS uses Codasyl DML [HSIAO & KAMEL, 1989].

Semantic heterogeneity refers to the difference in the semantics associated with the data of the component databases. The databases which are semantically different may represent data by the same data model but use different data interpretations [SHETH & LARSON,1990]. For example two tables may have the same columns SALARY, but one is the gross payment from a company and the other is the net payment from a company.

There are two approaches to solve the problem of schema integration from users. The first approach presented by [DAYAL & HWANG,1984] lets the DBA of the HDS or HDDS create a global schema for a set of databases being integrated, and each user application is provided with its own view of the global schema. This approach can be called a non-federated database system. The second approach presented by [HAMMER & McLEOD,1980] is that, for each application, the DBA creates a schema describing data that the application may access in the component databases, i.e. an import schema for that application. The DBA for each component database creates a schema for the data in her database which she is willing to share. This approach is called a federated database approach. Federated schemas may correspond closely to user views, or user views may be further derived from federated schemas [THOMAS et al.,1990].

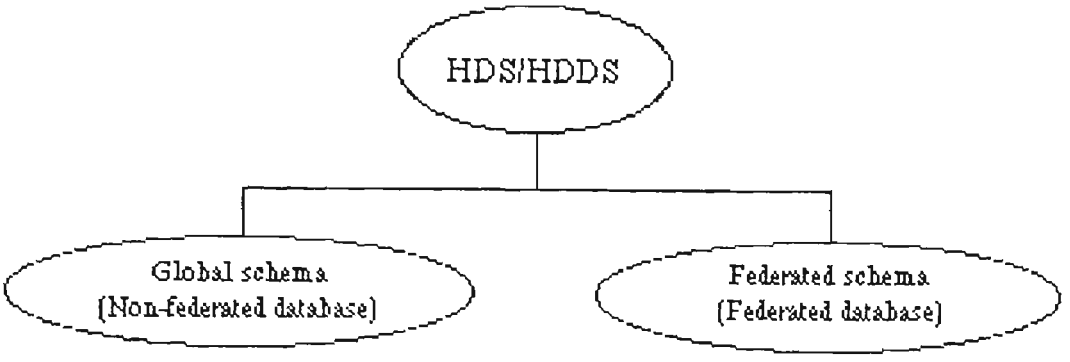


Figure 2.2 Two different approaches in HDS or HDDS to schema integration.

2.2 Design Constraints

The most obvious factor in the design is related to the area of Distributed Computer Systems [SINGHAL & CASAVANT,1991]. This leads to the communication problems - different computer systems might use different connection protocols. The design of communication protocols can have a direct impact on system efficiency and reliability [SINGHAL & CASAVANT,1991]. Moreover, protocols must guarantee the correctness of transferred data.

The syntactic heterogeneity and semantic heterogeneity of data in different repositories are other factors that system designers must take into consideration when designing system software to access data across the network. The interpretation of different names from different systems which, in fact, represent the same object or data is needed. [BERSHAD & LEVY,1988] [SMITH & OMAN,1990]. Moreover, data model translation may be required where different data models are transferred to a local system [BERSHAD & LEVY,1988].

2.3 Solutions to Syntactic Heterogeneity

[WIEDERHOLD et al.,1992] proposed the megaprogramming technique which is a technique for programming with large modules - megamodules. This technique can be applied to the design of system software by considering each access function as a function module in the software. Then, access functions over the HDDS can be referred to as megamodules that capture functions of services provided by large organisations. The architecture of a megaprogramming system consists of a collection of geographically distributed megamodules linked by high-capacity networks [WIEDERHOLD et al.,1992]. Using the principle of orthogonal design, it is easier to maintain the whole system by performing maintenance on each megamodule. The technique which is applied in maintaining each megamodule is to implement a megamodule repository and dictionary which is a collection of information relating to megamodules in the megaprogramming environment.

The idea of implementing a dictionary which stores information about megamodules can be applied to the design of system software to manage various kinds of data schemas in a HDDS. For example, the dictionary of servers of Wide Area Information Servers (WAIS) is a collection of all information of available services, including access method on each server.

[PAPAZOGLOU,1991] proposed the idea of introducing appropriate software modules on the top of a set of interconnected autonomous data/information repositories. This software is referred as Corporate Information System (CIS). This method is similar to the method of [WIEDERHOLD et al.,1992] in that "the CIS utilises some sort of a kernel data model whose prime purpose is to furnish the system as a whole with the appropriate structural and semantic capabilities through which data unification is made possible" [PAPAZOGLOU,1991]. Furthermore, the method of using a federated database to maintain the topology of the federation and oversee the entry of new services has also been presented by [PAPAZOGLOU,1991].

Another method which is used in managing distinct data schemas in the HDDS is to define a global schema for all existing schemas in the HDDS [PAPAZOGLOU,1991]. All data schemas from different repositories will be translated and mapped into the defined global schema. This method has been applied to the design of some system software such as DATAPLEX, as discussed below. The database systems which are integrated to each other using a global schema are classified as non-federated database systems, which are a subset of possible system designs.

2.4 Solutions to Semantic Heterogeneity

Solutions to the problem of semantic heterogeneity can be achieved by various methods. [SMITH & OMAN,1990] proposed a dictionary called a *semantic dictionary* to provide high efficiency in accessing shared data from diverse database systems. The other method presented by [SCHLICHTER & MILLER,1988] is to *set up standard data information.* This method is implemented in a publication management system, FolioPub[4]. A third method which has been implemented in some system software is the method called *Name Translation* [BERSHAD & LEVY,1988]. System software such as THERE[5] [BERSHAD & LEVY,1988] and ANSA use this method to determine relevant information from distinct servers.

---

[4] FolioPub is the publication management components of an experimental production publishing system; focussing on publication definition and automatic publication processing in a distributed environment [SCHLICHTER & MILLER,1988].

[5] THERE is a general-purpose metaservice designed to simplify the adaptation of non-network, nonheterogeneous applications to a distributed heterogeneous environment [BERSHAD & LEVY,1988].

2.5 Supporting Languages

Generally, such system software contains many modules and the total size is large. There exist some problems relating to the requirements of persistence, diversity, and infrastructure, which deserve as much attention as sheer size [WIEDERHOLD et al.,1992]. The concept of using Module Interconnection Languages (MILs) has been proposed by [WIEDERHOLD et al.,1992] to facilitate the construction and management of programs. Furthermore, Megaprogramming Languages (MPLs) has also been introduced to handle information transfer between heterogeneous modules, between dynamic queries and updates by users, between distributed network communication protocols and by dynamically changing the specifications of interfaces. The effect of implementing MPLs is that difficulties in dealing with various modules are reduced and users gain high efficiency in accessing large systems.

Many system software packages apply the concept of using a standard language to reduce the complexities of operations over the network, such as deleting, retrieving and updating data, or concurrency control. The THERE system introduces the THERE Programming Language (TPL) [BERSHAD & LEVY,1988]. TPL handles problems such as naming, communication, location of data, binding, and data transfer. FolioPub manages the file process using the File Processing Language (FPL) file, which contains operating system commands that invoke programs to perform the actual computation [SCHLICHTER & MILLER,1988]. ANSA provides the Distributed Processing Language (DPL) to clearly distinguish the concurrency expressed in the computation [APM(3),1991].

The following sections will describe three different heterogeneous distributed database management systems, each of which is the federated database system, in the sense in which we have defined it above.

2.6 Some Design Alternatives

DATAPLEX, WAIS, and ANSA are database management systems with different design approaches. However, the common framework among these three is that they are federated database systems. The DBAs of component databases in each system must register themselves to the headquarter of each HDDS before granting access to the data via the HDDS. Users of DATAPLEX must use SQL to access data in a HDDS, the client and server systems use WAIS protocol in order to establish the retrieval communication, while in the ANSA system, clients and servers must install the *ANSAware* package to enable communication between participating systems.

Here we present a short summary of the three different designs.

2.6.1 Design of DATAPLEX

DATAPLEX [CHUNG,1990] is a heterogeneous distributed database management system (HDDBMS) which has been developed by General Motors Research Laboratories. The DATAPLEX approach uses the relational data model as the canonical data model and SQL as its standard query language.

DATAPLEX is implemented in manufacturing industries to allow users to retrieve data from diverse autonomous databases, allowing semantic and syntactic heterogeneity, to support efficient engineering and manufacturing activities, and business operations. DATAPLEX allows sharing of data and reduces problems in operations [THOMAS et al.,1990].

Architecture of DATAPLEX

DATAPLEX treats the entire HDDS as a single relational database model that containing three schemas: local schema, conceptual schema, and external schema. The local schema contains data definitions used by each database system. The conceptual schema consists of data definitions of all sharable databases in a HDDS which are transformed to an equivalent relational data definition; it is implemented as a set of overlapping relational schemata, one for each location and the relational at each location represent data objects that need to be accessed by users at that location [THOMAS et al.,1990]. The relational data definition of a conceptual schema contains only the information on stored data objects and local views. The external schema consists of each user's view of data in a HDDS which is contained within the user box in the Figure 2.3, but has not been explicitly drawn. A user's view of data is a set of data that is presented to the user in the format defined in the user's application program. Figure 2.3 illustrates the DATAPLEX system in a HDDS.

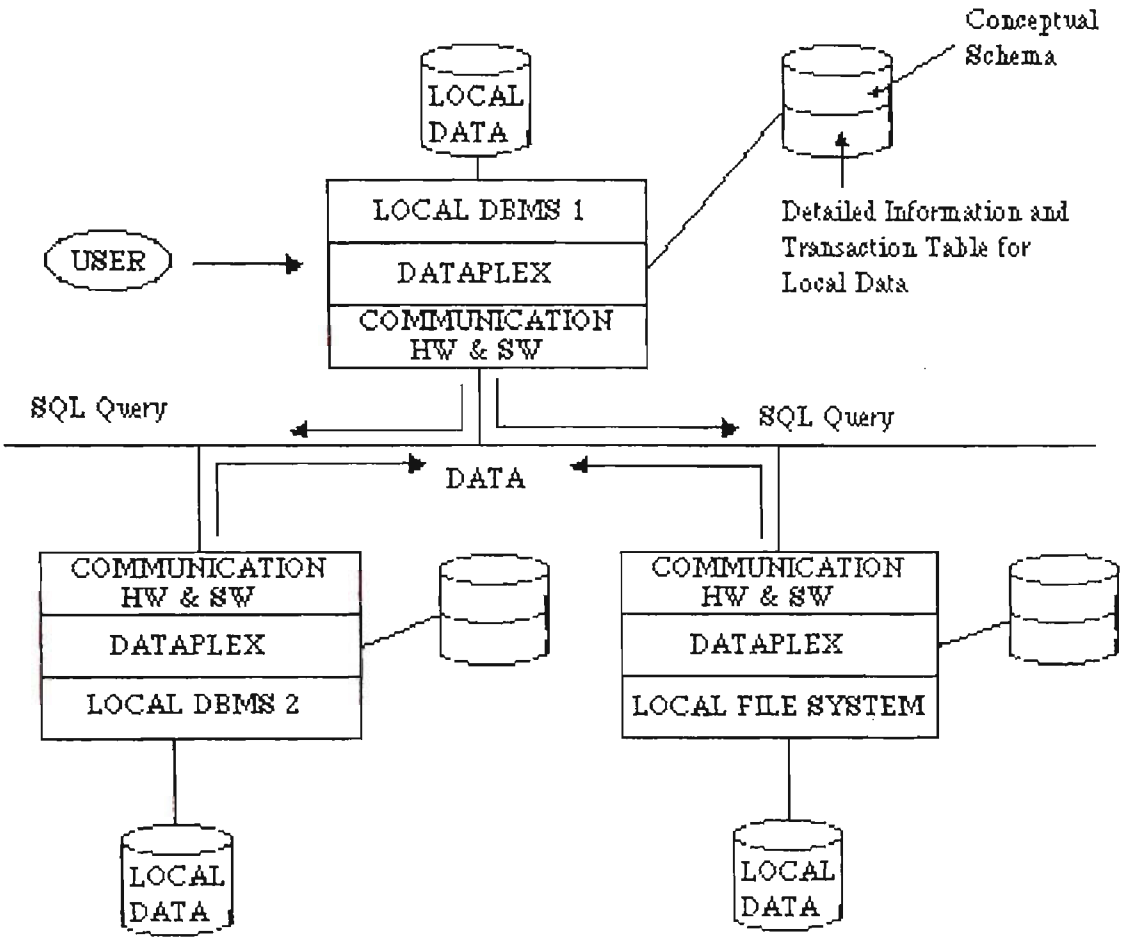


Figure 2.3 DATAPLEX in a HDDS [THOMAS et al.,1990].

The functionality of the DATAPLEX system is divided into fourteen modules which are functionally independent, in that each module is responsible for its implementation and provides services to other modules. Using these modules, data access is transparent to users. The names and functions of these fourteen modules are as follows:

1. *Controller*

   This module arranges for the required modules to process a transaction in the right sequence, and also performs the multiple processes by multitasking.

2. *User Interface*

This module provides and controls the screen that allows users to issue the SQL query and presents the result to users after the processing of a query.

3. *Application Interface*

This module is responsible for communication between an application and DATAPLEX.

4. *SQL parser*

This module checks the syntax of a SQL statement, and includes referred names which are set up by the user in the application program.

5. *Data Dictionary Manager*

This module provides facilities to create the DATAPLEX dictionary. Data Dictionary Manager is also responsible for aiding in the retrieval and update of information in the data dictionary. The DATAPLEX dictionary includes information of locations of referenced data.

6. *Security Manager*

Security manager is responsible for global data object names and user-id to perform the content-independent access control.

7. *Distributed Query Decomposer*

This module decomposes a distributed query into individual local queries which may be transactions at remote data or local data sources. A distributed update statement will be decomposed to distributed retrieval and distributed update queries.

8. *Distributed Database Protocol*

This module provides communications between the DATAPLEX software at user locations and the data sources, and handles the detection and reporting of all problems dealing with remote access.

9. *Translator*

The translator is responsible for the translation of a query into the component specific transaction mechanism of each component database.

10. *Local DBMS Interface*

This module passes the translated transaction to a local DBMS and obtains the result.

11. *Distributed Query Optimizer*

This module executes the data-reduction plan by sending commands to the locations involved.

12. *Distributed Transaction Coordinator*

This module controls locking and unlocking on each local database to achieve the desired result of a distributed transaction.

13. *Relational Operation Processor*

This module at the user location merges the results from the local sites to provide the final query result.

14. *Error Handler*

This module is responsible for recovery of the system under an event of an error. It will abort the transaction and clean up the system after a non-recoverable error, and then send an error message to a user and record the error in the transaction log.

Distributed Data Security

The data security mechanism of DATAPLEX is Content-Dependent Access Control (CDAC) [CHUNG,1990]. The access rules of CDAC include a predicate whose value depends on data content. In the DATAPLEX environment, security control involves two steps: security control by DATAPLEX itself, and security control by each local database management system .

A function of a global database administrator of a DATAPLEX system is to create a DATAPLEX dictionary. Under certain circumstances, the administrator has to create a view which contains a predicate to enforce a CDAC. After the DATAPLEX dictionary is created, the administrator lets users access the global data objects in the dictionary. Whenever a user accesses the global data objects, there are two possibilities:

1. *The user is an authorised user of local data objects.*

   This situation occurs whenever global data objects correspond to local data objects managed by a local DBMS.

2. *The DATAPLEX system is an authorised user of local data objects.*

   If the global data object is a global view defined on one or more local data objects, then DATAPLEX becomes an authorised user of local data objects at their location.

Distributed Retrieval

To process a distributed retrieval query, the Distributed Query Decomposer decomposes the distributed query in an SQL statement into a textual form constituting a set of local queries the results of which are merged and presented to the user. A nested query is executed as a sequence of non-nested queries by executing the inner-queries of the nest first. The nested-query is transformed into a set of aggregate-free conjunctive queries. The non-conjunctive query is decomposed into a set of conjunctive queries which contains only *AND* boolean operators. After obtaining a conjunctive query, the distributed conjunctive query will be transformed to a query graph. A node of the graph represents a relation and a qualification term is represented by an edge. Using the location of information, this graph is decomposed into connected sub-graphs by deleting the edges that correspond to global join terms. The global join terms are assigned to the qualification of the user-location query. Every sub-graph is transformed to local queries. The user-location (source) DATAPLEX software sends local queries to data-location (target) DATAPLEX software using DDP.

The Translator finds query translation information from a translation table that keeps a mapping of data names and data structures between the conceptual schema and the local schema. The Translator translates a user SQL query to a query (or program) in a component DML using the translation information. The Local DBMS Interface sends the translated query to component DBMS and obtains the result, which is in a report form similar to a relation regardless of the data structure used by the component DBMS.

The Distributed Query Optimizer at the source schedules an optimal data reduction plan using the statistical information from the targets. The data reduction plan is a sequence of semi-joins that consists of local data reduction operations and data moves among computers. Upon completion of the execution of the data reduction plan, the reduced local results are sent to the

source, where the relational Operation Processor of the source merges the component results to answer the original query.

Distributed Update

As a consequence of a user request to update data, a transaction is invoked which performs the updating operation. The update operation invoked by the user may reference data residing at logically distinct and physically distant sources, raising three problems: distributed concurrency control, distributed deadlock handling, and distributed data recovery. These problems are solved in DATAPLEX in a manner similar to that used for a local DBMS .

Concurrency control is addressed by two-phase locking. DATAPLEX uses this method to perform distributed concurrency control by doing global protection of release locks from LDBMS, so that locks will be locked by the LDBMS and released under the control of DATAPLEX. DATAPLEX will allow LDBMSs to release their locks when the update processes at all relevant locations have been completed. The method used to detect a distributed deadlock is using the time-out mechanism, which is also used by the individual LDBMSs. The distributed data recovery method is the two-phase commit mechanism similar to a LDBMS. The distributed two-phase commit forces LDBMSs to commit and release locks only after all DBMSs involved in a distributed update are prepared to commit.

2.6.2 Design of Wide Area Information Servers

The Wide Area Information Servers (WAIS) [KAHLE,1989] is a system that allows the end users to gain information from a variety of data sources which may reside at physically distinct locations. These sources of information are under autonomous control and operate in heterogeneous environments. WAIS is the outcome of a joint project between the Thinking

Machines Corporation, Apple Computer, Dow Jones & Co., and KPMG Peat Marwick [KAHLE et al.,1992].

The primary objective of WAIS was to define an open protocol that would allow any user interfaces or information servers using the protocol to interact with any other components which using the protocol [KAHLE et al.,1992]. Today, WAIS is implemented in many systems. Examples of WAIS applications are library catalogues, movie schedules, class schedules and catalogues, bus schedules, etc. The characteristic of all applications is a text retrieval from a remote access [KAHLE(3),1991].

Architecture of WAIS

The WAIS system has been developed from the standard protocol Z39.50[6], which has been under continual development for nearly a decade. The WAIS system consists of three major components: *clients*, *servers*, and the *protocol* that links them together. A client is a user interface; a server is a data repository which maintains indices to aid retrieval of documents. The protocol is used to transmit queries and responses between clients and servers [KAHLE(1),1991].

WAIS Client System

A WAIS client is used to send queries to access documents possibly located at different locations. These queries are in the form of English language (natural language) questions. A query is translated into the WAIS protocol and transmitted over a network to a server. The client contains information on each server that includes the access method, a description of contents, and the access cost. Client programs for different computer systems are described in Table 2.1.

---

[6] The Z39.50 is a search and retrieval protocol, developed by NISO.

Interface_Name	Target_Machine	Language	Communications
WAIStation	Macintosh Plus 9" Monochrome Screen	Think C	TCP/IP and Modem
ROSEBUD	Macintosh II, Color Screen	Smalltalk MPW-C	TCP/IP using IPC package
XWAIS	X-windows Terminals on UNIX Machines	C	TCP/IP

Table 2.1 Examples of Interfaces for Different Systems.

A user interface allows users to enter queries in a natural language. The user can mark up the replies from the server as yes or no, maybe, and select parts that are of particular interest. These marks then construct the second query if it is necessary [KAHLE(2),1991].

WAIS Server System

A WAIS server is a computer that maintains information on a specific theme to apply to client applications [KAHLE,1989]. A WAIS server can be located anywhere on a network. A user of the WAIS system is only allowed to retrieve data. The data security and data manipulation are controlled by the server. A server will allow data to be retrieved by a user who has explicit access rights. Documents are distributed with an explicit copyright disposition in their internal format [KAHLE(1),1991].

A WAIS client may keep track of the server from which it has previously received information, for subsequent retrievals. As the number of servers grows, it is impossible for a client to keep track of all servers, thus requiring the development of a dictionary of servers.

Dictionary of Servers

The dictionary of servers is a database which maintains information on all available servers. It contains descriptions of how each server can be contacted and the relative cost of doing so. A dictionary includes the following information [KAHLE,1989].

1. Description of servers in English.

2. The parent server, if it is a subsidiary of a larger server.

3. Related servers.

4. Public encryption key.

5. Contact information - including networks and contact points.

6. Cost information.

A dictionary of servers is updated whenever a new server appears on the system. Each new WAIS server must send a description itself to register into the dictionary. If registration of the new server fails, the server will not be visible.

WAIS Protocol

The WAIS protocol evolved from the standard protocol Z39.50. Z39.50 [NISO,1988] was designed to search electronic catalogues, returning a list of titles and document IDs. The WAIS protocol is an extension of the Z39.50 protocol from NISO. It has been found to be effective for the federation of full-text retrieval systems. The WAIS protocol includes support for multimedia, large files, or parts of files, and can handle graphics images and sound or video formats [KAHLE(2),1991].

Distributed Retrieval

There are four different methods to retrieve text - using identity, content, association with other items, and criteria [KAHLE(2),1991]. WAIS retrieves document IDs from servers, allowing users to select documents of relevance. The user typically formulates a query in the form of text strings. Following the formulation of the query, sources which are to be searched for the query are selected. On the execution of the query, the text associated with the query so formulated is packed into the WAIS protocol and transmitted over a network to one or more servers. When the query is transmitted, the WAIS system automatically queries all servers for the required information without intervention from the user.

When a server receives a query from a client, the query packet is translated into server query language and used to search for documents satisfying the query. The outcome of searching process is a list of relevant documents, identified according to the WAIS syntax with a document ID, a title, score, types and date. This list will be encoded in the protocol packets and transmitted back to the client. The client decodes the packets and displays the results on the screen. The user then selects documents from the displayed list to request the complete document from the particular server.

The location of a document is embedded in the WAIS protocol. This protocol encodes search terms and Boolean constraints or relationship among words, and includes an optional procedure for relevant feedbacks[7]. Using an optional procedure, users can send document IDs and optional subsetting parameters - which are transformed into a document by the system as the text of a query.

---

[7] The relevant feedback is a method of information retrieval when a user marks the retrieved documents and re-run the search to obtain the similar documents.

2.6.3 Design of Advanced Network System Architecture

The Advanced Network System Architecture (ANSA) is an architectural framework for the design and construction of distributed systems. ANSA does not attempt to hide the distribution of the individual systems but makes the distribution transparent to users, administrators, and application programmers. As a consequence, ANSA allows exploitation of the inherent concurrency of distributed systems [APM(1),1991].

There are two systems that developed at the same time: the Advanced Network System Architecture (ANSA), which was developed in Europe and the Common Object Request Broker Architecture (CORBA), which was developed in USA. These systems have the same objective in addressing the problems of developing distributed systems in data processing. ANSA concentrated on the ability to find and use software components via a trading mechanism, and did not include any form of inheritance, whilst CORBA came from the Object Management Group (OMG), and used inheritance in its interface language, IDL, from the start [TOMLINSON, 1991]. Nevertheless, CORBA will not be mentioned in detail in this Chapter.

The US National Aeronautics and Space Administration (NASA) has an Astrophysics Data System (ADS) which is a collection of all information on NASA space programs. The ADS consists of various systems and databases including IDMS, Ingres, Oracle and a number of home-built packages. The information are text, data, and images. ANSAware, an infrastructure for distributed systems of ANSA, is used to join with the Knowledge Dictionary System (KDS), to provide the means for client applications to use the KDS as a distributed catalogue and to direct queries to an appropriate database site.

Architecture

Under the ANSA approach, the five projections presented below must be taken into consideration when designing a suitable architecture [APM(1),1991]:

1. *The Enterprise projection* is concerned with the processing roles of an information system within an organisation.

2. *The Information projection* deals with the representation of meaning and value of information within an organisation.

3. *The Computational projection* is concerned with the structured programming language to be run on a distributed computer system.

4. *The Engineering projection* is associated with the problem of efficiently running distributed programs within the finite resources available.

5. *The Technology projection* refers to components of hardware and software, which are used to construct a distributed system.

From these five projections, three models unfold: the computational model, engineering model, and technological model. The computational model can be thought of as a set of building blocks for constructing programs which can be distributed across a large distributed system [APM(1),1991]. The engineering model, or a system builder viewpoint, is an engineering entity that allows the same functions to access other parts of the system. The technological model represents the operating system on which ANSA is implemented. There are a variety of operating systems each constituting a different technological model [APM(2),1991].

ANSA Design Aspects

There are five design aspects to be considered when constructing distributed applications. These are *separation*, *heterogeneity*, *federation*, *concurrency*, and *scaling*.

1. *Separation*

The first assumption of separation is that all services are physically or logically remote from each other, which raises the problem of accessing information residing at distant locations. This requires that each service is able to encapsulate its data, and that the state and data of each remote service can only be manipulated indirectly by interacting with one or more interfaces[8] supported and made available by the service.

The most important property of the computational model is that remote services at an interface can be shared by clients invoking the services via interface references[9]. The encapsulated data and the service operations for manipulating that data are referred to as a *computational object*. Each computational object of a service is stored in its private memory space, distinct from other objects.

2. *Heterogeneity*

Many incompatibilities may be experienced in the construction of distributed applications, as a consequence of the different modules of the application residing at different locations. Such incompatibilities include different operating system interfaces, physical and logical data representations, and communication protocols. The solution of the ANSA model to heterogeneity is to use interface instances to perform remote service interaction and make services public enabling sharing by publishing *offered* services.

---

[8] An interface is a unit of service provision.

[9] An interface reference is an entity which refers to an instance of an interface [APM(2),1991].

Communication between services is performed by passing interface references to each other. The possession of an interface reference by a client allows an invocation of service operations provided at the interface by a server [APM(2),1991].

3. *Federation*

ANSA does not impose a centralised control on each component of the system, but allows each component to control itself. Thus, a negotiation of shared services between cooperating systems must be defined. The cooperative systems also have to identify all available services via a context-relative naming scheme.

To manipulate the information in the federation, it must be possible to identify *names* and *trading services* of a component system. Many names may be used to refer to a specific object. To resolve this problem, ANSA defines the federated naming model as follows:

1. separating of *Naming domains*[10];
2. separating of *Naming conventions*[11];
3. separating of *Naming contexts*[12];
4. *Naming networks*[13];
5. *Path names*[14]; and
6. *Name transparency*[15].

---

10 A *Naming domain* is developed for heterogeneous entities that can be named.
11 A *Naming convention* is a method of naming an entity.
12 A *Naming context* is a set of binding between entities in a naming domain and names in a name set.
13 A *Naming network* is a structure which names a naming context from another naming context.
14 A *Path name* is an extended name which traces a path through the naming network.
15 A *Name transparency* is a Naming context that is not visible to the interpreter.

As well as naming methods, ANSA also concerns itself with interactions between clients and servers. The client uses interface references to interconnect to accessible servers. The process which separates clients and servers and is used to interconnect initially is called *trading*. After the trading process has completed successfully, two objects[16] are able to interact. The federated cooperative systems can organise and control sharing services - using trading facilities of ANSA. The trading is accomplished by *typename* and optionally by *property name/value pairs*.

A typename denotes a set of permissible interactions that a service instance can engage in. It identifies a set of common service interface instances. A property name/value pair is used to support decision making from a set of instances with the same typename.

Servers register typename and property name/value pairs representing services with the trading service (or trader). The operation by which a server advertises services to the system by registering an interface with the trader, is called *exporting*. The trader also contains operations to search for a service which a client intends to use, called *importing*. Thus, the client receives the interface reference from the trader by an importing operation.

The function of a trader is only to search through exports of the required type. It tries to match on the interface type conformance and required service properties [APM(3),1991].

4. *Concurrency control*

A distributed system will inevitably be faced with the problem of distributed concurrency and synchronisation. It also cannot avoid the possibility of an overlapped request for a server. To arrange distributed concurrency and synchronisation, ANSA introduced a

16 The two objects might refer to a client and a server, or between two servers.

Distributed Processing Language (DPL). The difference between the computational model and engineering model can be distinguished by the DPL.

A program which runs on the ANSA system using the DPL must declare opportunities for computational parallelism and requirements for synchronisation, without reference to possible mechanisms for their implementation.

5. *Scaling*

In real life, a system always changes. It is difficult to set up system software or a package to support all possible changes. ANSA will extend naming and trading facilities to support the growth of the system, with each computational object keeping the information on the services and data it provides. However, this method does not guarantee that ANSA is able to support the scaling of any system for an unlimited period.

Distributed Retrieval

In ANSA, a programmer must state distribution requirements in the client program; ANSAware interpreters can be applied to the source to yield components corresponding to these requirements. The client program also declares all variables which are used to store data from a HDDS without knowing their locations.

There are two systems that were developed at the same time: the Advanced Network System Architecture (ANSA), which was developed in Europe, and the Common Object Request Broken Architecture (CORBA), which was developed in USA. These systems have the same objective in addressing the problems of developing distributed systems in data processing. The ANSA concentrated on ability to find and use software components via a trading mechanism, and did not include any form of inheritance, whilst CORBA came from the Object Management Group (OMG), and used inheritance in its interface language, IDL, from

the start [TOMLINSON,1991]. Nevertheless, CORBA will not be mentioned in details in this Chapter.

The communication between objects is performed through an interface. Client programs or computational objects must define interfaces, which are written in the *Interface Definition Language (IDL)*. The IDL statements are embedded in the client program and compiled by the *stub* compiler. After computational objects are compiled by the stub, the result is a set of stub routines or engineering objects.

The client program is also embedded with PREPC[17] statements. PREPC is a language providing a means for embedding invocations of an interface operation in C. The compiler, called the PREPC compiler, translates PREPC statements into invocations of stub routines [APM(2),1991]. After completing the compilation of files with the stub and the PREPC compilers, the programmer must assemble object modules into an executable program file.

Traders are divided into two types: local traders, and a master trader. The local traders bind their local contexts to the master trader. All client and server programs will interact with the local trader before interacting with the master trader.

The engineering objects are imported to a trader to look for services which need to be used. The result of this search is an interface reference, which will be passed to a *nucleus*[18]. Then, the nucleus provides engineering objects referred as *capsules*. The relationship between a capsule and a nucleus is shown in Figure 2.4. Additionally, there is a service called a *transparency service* that manages nucleus-provided resources in a capsule, and communication with it's peers in other capsules - to provide the required transparency [APM(2),1991].

---

[17] PREPC provides three basic facilities which are interface type declarations and lexical binding of interface refere e variable, dynamic creation/destruction of interface instances, invocation of operations in one or more interface instances.

[18] A *Nucleus* manages the resources of a node; it includes a service definition for the protocol required for communication between nuclei.

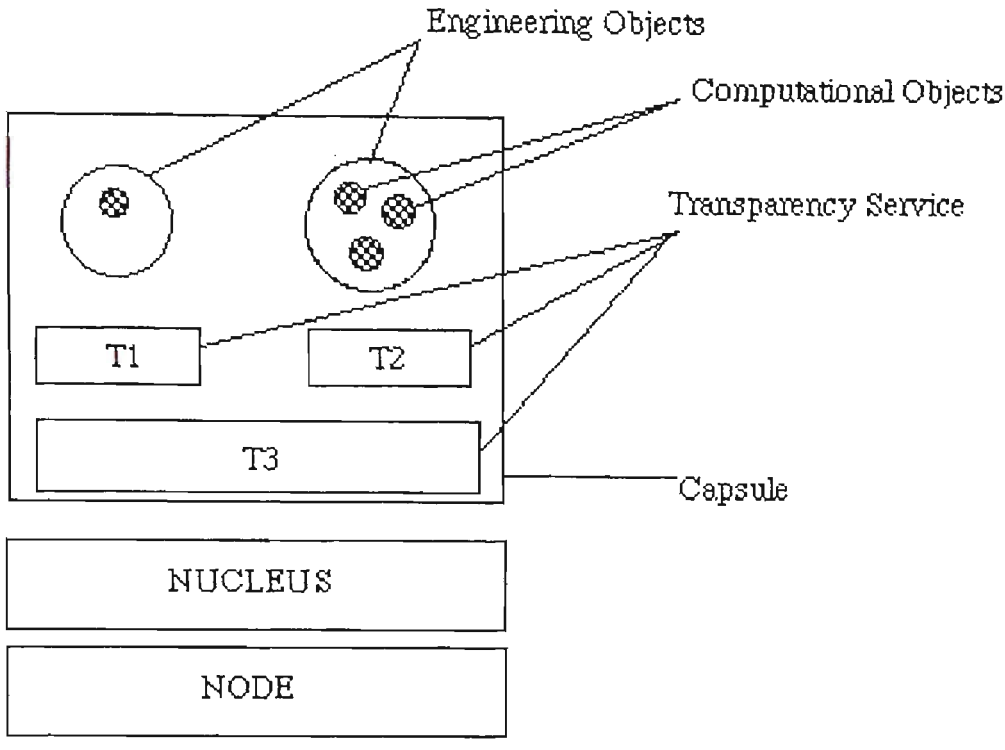


Figure 2.4 The Relationship between a Nucleus and a Capsule.

Generally, a network is assumed to be unreliable and the communication is based on Remote Procedure Calls (RPC) [APM(2),1991] [BIRRELL,1984]. Each remote operation is assigned a sequence number, client session number, and server session number. If a service is a local service then the service is processed on a local node on the network. On the other hand, a *Message Passing Service* (MPS)[19] is involved when a service requires a remote service. The message from a MPS will be sent by a *Remote EXecution protocol* (REX)[20].

The ANSA system on a server receives the message and interprets the message into a form of the server program; the interpreted command will be executed and the result will be returned back to the client by the REX protocol.

[19] An MPS provides a transport service between nuclei.
[20] An REX provides a simple service for process-to-process interactions across a network.

2.7 Analysing and Defining Problems

In this section problems that may arise in each of the three systems outlined in the previous sections are presented.

2.7.1 Problems with the DATAPLEX Approach

Under the DATAPLEX approach, a HDDS is assumed to be a relational data model. All component database systems which are not relational have their schemas transformed to the relational data model. The data definitions of all sharable database systems are stored in the conceptual schema. We will now discuss some potential problems with the approach.

1. Inconsistency between the data definition of a local database and that of the conceptual schema.

If the data definition of a local database is updated and the data definition at the conceptual level of DATAPLEX has not been correspondingly updated, this leads to an inconsistency between the local database and DATAPLEX which may cause an error in the retrieval process. The data schema of a local database might be altered as a consequence of the following operations.

A data field is deleted from data records.

In this case, if the deleted field is a sharable field then the data definition at the conceptual schema must be updated. Otherwise, the user will obtain incorrect details of data.

A data field is added into data records.

This added field will not be available for use by DATAPLEX until the data definition of the conceptual schema is updated.

A data schema of a local database is absolutely changed.

As a result of changing the whole data definition in a local DB, the conceptual schema of DATAPLEX is completely different from that currently defined. Thus, an error will occur when DATAPLEX translates a SQL command to retrieve information from the changed database.

2. **No retrieval process after the DATAPLEX dictionary has been damaged.**

The retrieval process can be performed after the location of data is found from the DATAPLEX dictionary. Thus, if the dictionary is destroyed, then locations of data will be undefined and the retrieval process must be terminated.

3. **Loss of information from a non-relational data model.**

This situation might occur when DATAPLEX translates a non-relational data model to be a relational data model that is a part of a conceptual schema. Even though DATAPLEX uses a transaction table to store all information which could be lost, it still cannot guarantee that all information that can be lost are kept in the table.

2.7.2 Problems with the WAIS Approach

Problems which may arise using WAIS are as follows.

1. **The dictionary of servers might be inconsistent with the current servers and their details.**

The server must register itself to the dictionary of servers to convey the details of its service. WAIS uses this dictionary to find the location of a required document, and the method to connect to a server. Whenever some relevant details of a server change, the corresponding information maintained in the dictionary of servers must also be updated to reflect the changes. If such details are not updated at the dictionary of servers, a client will gain incorrect information from the server and might not be able to retrieve the required document.

2. **Retrieval cannot be done if the dictionary of servers is unavailable.**

The same problem can exist with the DATAPLEX dictionary, if access to it is unavailable.

2.7.3 Problems with the ANSA Approach

The ANSA approach is more general than the first two approaches presented. The main idea of ANSA is to use the interface to communicate between objects. The important part before communication commences is the trading process. The trader gives an interface reference to the client to obtain information from servers. The problems that may arise under the ANSA approach to implement distributed systems include the following.

1. **The information of a trader is out-of-date.**

Sometimes a server might change its services and may not inform traders. Under such circumstances, the service information in traders will be incorrect. This may cause a client to receive an incorrect interface reference thus preventing contact with a suitable server.

2. **The information on a local trader and the master trader may not be the same.**

This case may arise if information of a local trader is altered but is not updated at the master trader. This may cause an error to occur as the client will not receive the correct interface reference from the master trader.

3. All traders are updated but the client program does not match the updated detail.

In ANSA, all variables must be declared and defined before receiving values from a server, passing through an interface. Consequently, if a value from the interface is changed, then a variable declared in the program may not match the received value.

2.8 Comparisons between Three Approaches

In this section, a comparison of the designs of DATAPLEX, WAIS, and ANSA will be presented. All three are database managers which provide information from various databases on the network, and protect information from unauthorised users. Using DATAPLEX or ANSA, an authorised user can perform data manipulations such as deleting, adding or updating data on databases available to them, while WAIS allows users to perform only the retrieval process on a database. DATAPLEX and ANSA allow users to access more than one database at the same time using one query, while WAIS can retrieve one document at a time for a user.

WAIS has a global dictionary of servers located in only one place, at the Thinking Machines Corporation., into which details of servers are recorded. Every query on WAIS will be sent through the dictionary of sources before distributing to individual servers. Thus, if the central dictionary of servers is unavailable, users can only access server addresses stored locally by them. The transaction management concept of DATAPLEX and ANSA is different from WAIS. DATAPLEX uses the distributed process to perform a retrieval transaction; each server

must install the DATAPLEX software to enable communication between a client and the server. ANSA uses distributed programming which means that every server and client in ANSA has to install an ANSAware program so that any operations issued from a client can be executed on a suitable server system.

DATAPLEX implements a distributed database protocol (DDP) so that a remote access can be done with an efficient error detection module. Remote access or remote data retrieval on WAIS is performed by a special protocol named the WAIS protocol. ANSA implements a computational model using an interface reference, which is an entity that refers to an instance of a unit of service provision, and uses special protocols named Message Passing Service (MPS) and Remote EXecution (REX) [APM(2),1991].

These systems implement a database to store information of shared data repositories, called respectively, a DATAPLEX dictionary, a dictionary of WAIS servers, and a trader in the ANSA system. They also inform the systems about data models in component databases, and the type of data manipulation language (DML) that can access the data in each.

If differences exist between data models and DMLs in a HDDS, it is the responsibility of a system to hide syntactic heterogeneity. DATAPLEX uses a User Interface module which allows a user to enter a query using a standard query language (SQL). The user of DATAPLEX system will interact with an external schema defined by the DATAPLEX software; this schema is a relational data model. The query in the DATAPLEX will be decomposed, translated to a suitable DML for each data source, and distributed to the locations defined in the DATAPLEX dictionary. WAIS has a user interface client to allow users to enter queries, this time using English-like statements, which will be translated into the WAIS protocol and be distributed to retrieve data from its locations. ANSA allows a user to write his/her program using embedded ANSAware language with the IDL and PREPC statements. The user can define his/her own

data model in the program declaration section, then compile the program with the STUB and PREPC compilers to obtain engineering objects, and integrate the object modules into an executable program file.

WAIS only performs a retrieval process, so need not have a mechanism for concurrency control. DATAPLEX uses the 2-phase locking mechanism for concurrency control ,while ANSA uses the DPL embedded into a client program.

Comparisons	DATAPLEX	WAIS	ANSA
Data manipulation operations	retrieve, update, delete data in a HDDS	retrieve data only	retrieve, update, delete data in a HDDS
Number of databases able to be accessed each time	more than one database can be accessed.	one database can be accessed.	more than one database can be accessed.
Controlling system	distributed control	centralised control	distributed control
Mechanism to overcome the semantic heterogeneity	DATAPLEX dictionary	Dictionary of Servers	Traders
Mechanism to hide the syntactic heterogeneity	a User Interface module (external schema) and conceptual schema (using relational data model)	a user interface	the data declaration in a user program
Query language to access data	SQL	English-like query	IDL and PREPC embedded into a client program
Transaction process	DDB protocol	WAIS protocol	Interface References, MPS and REX
Concurrency control	2-phase locking	no method required	DPL embedded into a client program

Table 2.2 Comparisons between Three Approaches

2.9 Summary

In this chapter, three different design approaches have been discussed: DATAPLEX, WAIS, and ANSA. Although their designs are different, the mechanisms to control semantic and syntactic heterogeneities of these software are quite similar: the implementation of a database called a dictionary (for DATAPLEX and WAIS) and a trader (for ANSA). These dictionaries and traders keep details of available servers on the network. The details are such as the location of a server, the connection method, the data model of the server's repository, etc.. Moreover, if a server wants to grant access to its database as part of a HDDS under the DATAPLEX or WAIS or ANSA systems, the administrator must register details about shared databases. These details will be inserted into the DATAPLEX dictionary or dictionary of servers or a trader. The location of the data definition database of the component database is not supplied, so automatic checking of this information is not available in these systems, and updates will occur only if a server administrator informs the administrator of the HDDS.

DATAPLEX uses SQL as the standard query language for users to access data, WAIS allows users to write an English-like language for their queries, and ANSA uses the ANSAware embedded with some special command such as IDL, and PREPC. Each implements its own protocol.

In my research, a file system is considered to be a sharable repository. Information on that file system must be stored, along with information on any participating databases. Figure 2.5 illustrates the situation when a file is used as a sharable source without any data definition available in the HDDBMS or a file management program.

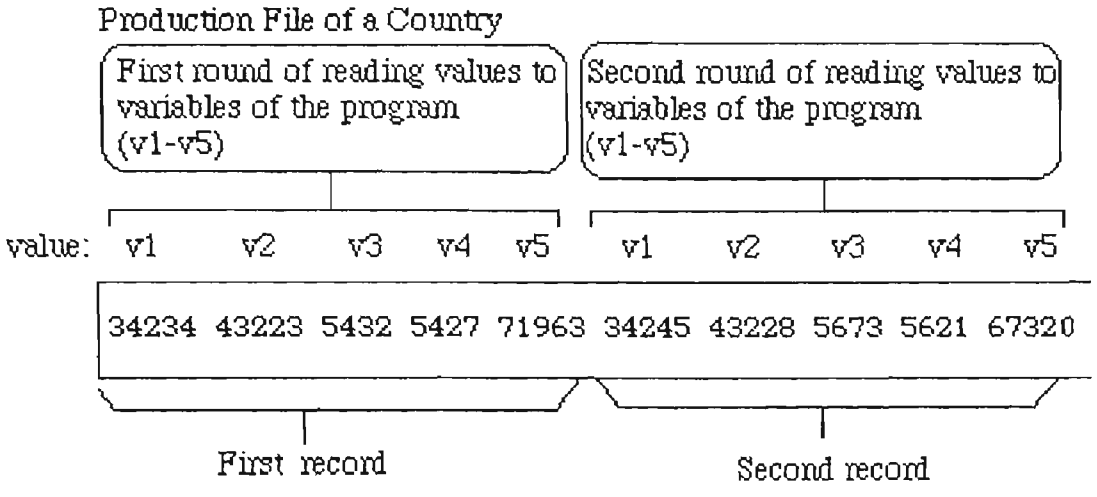


Figure 2.5.1 The Original Data File, before Adding a new Field.

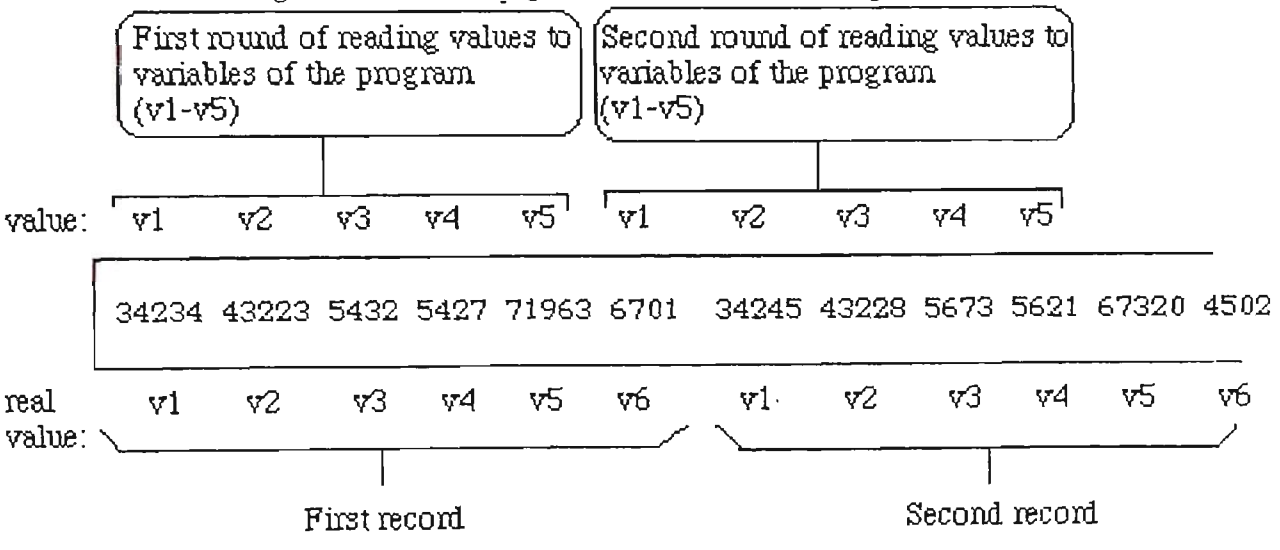


Figure 2.5.2 The New Data File, after Adding a new Field.

v1=grape v2=apple v3=rice v4=plum v5=orange v6=tomato

Figure 2.5 An Example of Error when a Data Field is Added.

From Figure 2.5, a file *X* contains production values of grapes, apples, rice, plums, and oranges of a country (Figure 2.5.1). Thus, a program will read 5 fields per record into variables *v1, v2, v3, v4, v5* respectively. The first time of reading, values of all variables will be v1=34234, v2=43223, v3=5432, v4=5427, v5=71963. The second round of reading, values of variables v1-v5 will be: v1=34245, v2=43228, v3=5673, v4=5621, v5=67320. If a product value of tomato is added into this file, then each record will contain 6 fields (Figure

2.5.2). After reading a first record from a file, variables v1-v5 will be 34234, 43223, 5432, 5427, 71963. Consider the second round of reading. The values of v1-v5 will be 6701, 34245, 43228, 5673, 5621, which are not correct because the value 6701 is the product value of tomato from the first record. The correct values of v1-v5 should be 34245, 43228, 5673, 5621, 67320.

To solve the above problem, the data definition of a sharable file system must be declared and stored as the header of the file, or a data definition file of any sharable files in the network must be implemented. In this way, all files can be shared in a HDDS and be controlled by any HDDBMS. An example of a data definition for a file is presented in Table 2.3

Field_name	Field_length	Field_type	Field_format
Country	15	CHAR	15
Year	4	INT	4
Product	15	FLOAT	13.2

Note: Assume that One Record Contains 3 Data Fields

Table 2.3 An Example of a Data Definition Table.

The next Chapter will propose a design of a Computer Software Interface (CSI) which allows a user to retrieve information from a HDDS. Data consistency and data integrity have been achieved. The system software designed in this thesis focuses on the situation where end users have been granted read-only privileges. In some systems the data is subject to continuous update, while in others a communication failure may isolate the user from some data repositories. The problem of dealing with some types of changes of data services has been considered and solved. Moreover, consideration of the problem of communication failure leads to the possibility of approximating some data when mathematical methods are inappropriate.

Chapter 3

System Analysis and Design

In this chapter, we describe our proposed system software design under the assumptions that end users lack authority to control or to change data, and in some situations communication between the client and servers is unstable, but that the protocol responsible for the communication of data between component databases guarantees correctness of transferred data. From the user point of view, data in a HDDS are related to each other in some way. For example, there is a key linking data from various databases, such as the telephone number in Telecom systems, or the cable TV entries in the databases for different regions. The proposed system software will consist of three subsystems: an Information Server System (ISS), a Query Generator System (QGS), and a Preserved Data System (PDS).

3.1 System Environment

As mentioned in Section 1.2 and Section 1.3, users cannot control the actions of owners of the databases participating in the HDDS and the communication between a client and a server may unexpectedly close. The configuration of a component node in a HDDS may vary in hardware, operating system, and data manager. Access to data stored in the HDDS may be granted by many applications. It is possible that there are multiple users who wish to access

data simultaneously for different purposes. Data manipulation on data sources can be initiated at any time by an owner or local system administrator. Figure 3.1 shows a possible HDDS configuration.

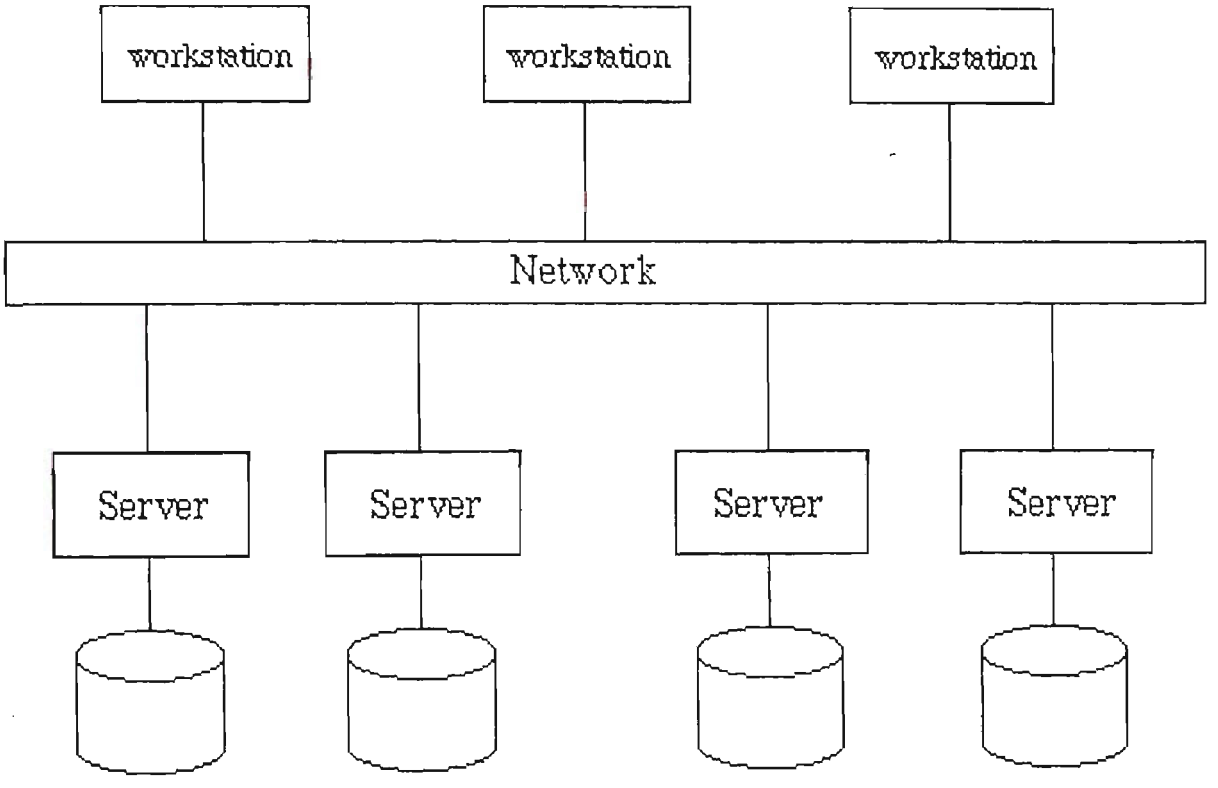


Figure 3.1 A possible HDDS configuration.

There are many distinct database systems holding particular data sets. Each individual system is characterised by a specific Data Manipulation Language (DML). This prevents the use of the DML of any individual data sources as a global DML. Furthermore, each data source will have a data schema for its own local data. These assumptions imply customised data access methods for the individual database systems.

A file system may or may not maintain any external record of the semantics of the stored data, nor will it support a DML. There are three primitive file structures: a sequential file, an indexed sequential file, and a random file, each with its own access method. Applications in

various computer languages are able to access the same file. Programmers have to know the file structure and the current data definition before writing any application that uses the file. Whenever a change of data definition of the file occurs, the existing application programs must be modified.

The last component that has to be considered in a HDDS is the communication protocol. For the various components of the heterogeneous database to be able to communicate, they need to employ the same connection protocol, in our case TCP/IP [DAVIDSON,1988]. TCP/IP is a collection of network protocols which support host-to-host communication which allow connection with any number of heterogeneous networks. The TCP resides above the IP in an internet in the same way as a UDP (User Datagram Protocol). The TCP is a reliable[21] stream service whereas IP is unreliable and connectionless[22]. In addition, there may be data transmission protocols such as Z39.50, or WAIS protocol, employed in the system.

3.2 Definitions and Theory

3.2.1 Data Consistency

In HDDS, such as the Telecom systems described in Section 1.2, data are separately stored and distributed amongst various data holders. However, the key values will be used to link data from different sources. In this thesis, the *relationship* between data will be defined by the user viewpoint, examples of which are:

---

21 The reliability of a protocol refers to the delivery service. It guarantees that the packet will not be lost, duplicated, or out of order.

22 Connectionless means that each packet is treated independently from all others and a sequence of packets sent from one machine to other may travel over different paths, or be lost while others are delivered.

1. the situation that every database in the HDDS contains the same key value, such as a telephone number, so that data from different databases can be combined;

2. the situation that the user has an expectation that a functional relationship, not expressed in the form of an explicit global integrity constraint, exists between certain items in different databases, such as that corn production is related to fertiliser used;

Whenever a user sets up a rule to combine data in a HDDS, we describe that data as *interdependent data* [RUSINKIEWICZ et al.,1991].

Definition 1: There are two types of data in a HDDS: data which have relationship with other data (interdependent data) and data without relationship with others (independent data).

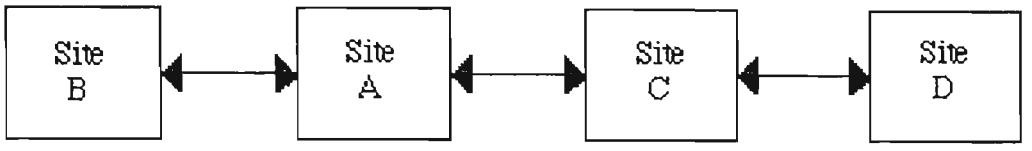
We assume that all data sources belong to different organisations or are independently owned, and that remote end users cannot control the management of the data in the HDDS. Therefore, consistency of integrated data in the network is defined as follows.

Definition 2: The data in different databases carry a timestamp, either explicitly in the individual record, or implicitly in the database update time. We will take consistency to mean that the timestamps of the data to be integrated agree.

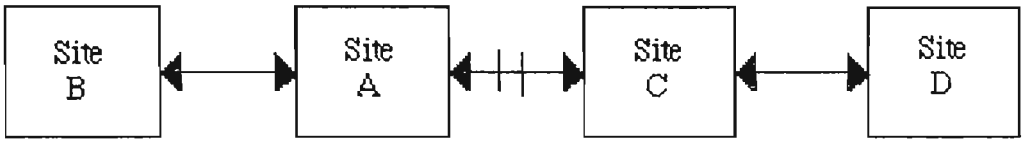
3.2.2 Communication Failure

Communication between client and server may fail if the connecting network is partitioned, or if the computer running the server program, which is not under user control, is temporarily taken out of service. The status of network partitions is that sites on different

subnetworks cannot communicate to each other. For example, a site(A) can communicate to site(B) but not site(C). Site(B) also cannot communicate to site(C). This means the network is separated into two subnetworks, one contains site(A) and site(B), another contains site(C). Figure 3.2 illustrates the network partitioning situation.



A: Communication line in normal situation.



B: Communication line between site A and site C is closed.

Figure 3.2 An Example of a Network Partition

3.2.3 What Constitutes Unreasonable Data?

The information from a database or a file system is transferred to a client by the TCP/IP protocol. The protocol ensures that the message is correct, in order, and not duplicated. It is possible that an owner inadvertently entered an incorrect data item, which a user can see is unreasonable. Table 3.1 shows an example of unreasonable data.

Definition 3 : An *inconsistent data* item is an item that satisfies one of the following:

1. Data consists of unexpected characters or special characters, such as #$@$.
2. Data has a type different from the expected type, for example, data must be a numeric value but an alphabetic value is presented.

3. Data is missing.

4. Data is out of range, for example, a data should be in the range of [10,400] but the received data value is 7687.

5. Data is not out of range but it is not resonable when compared with the other values, such as a total_payment is 2345.00 but a partial_payment is 3345.00.

Most of these problems can only arise from a lack of integrity constraints in the individual databases. Some might arise because there is no mechanism to impose global integrity constraints on data entered by independent owners.

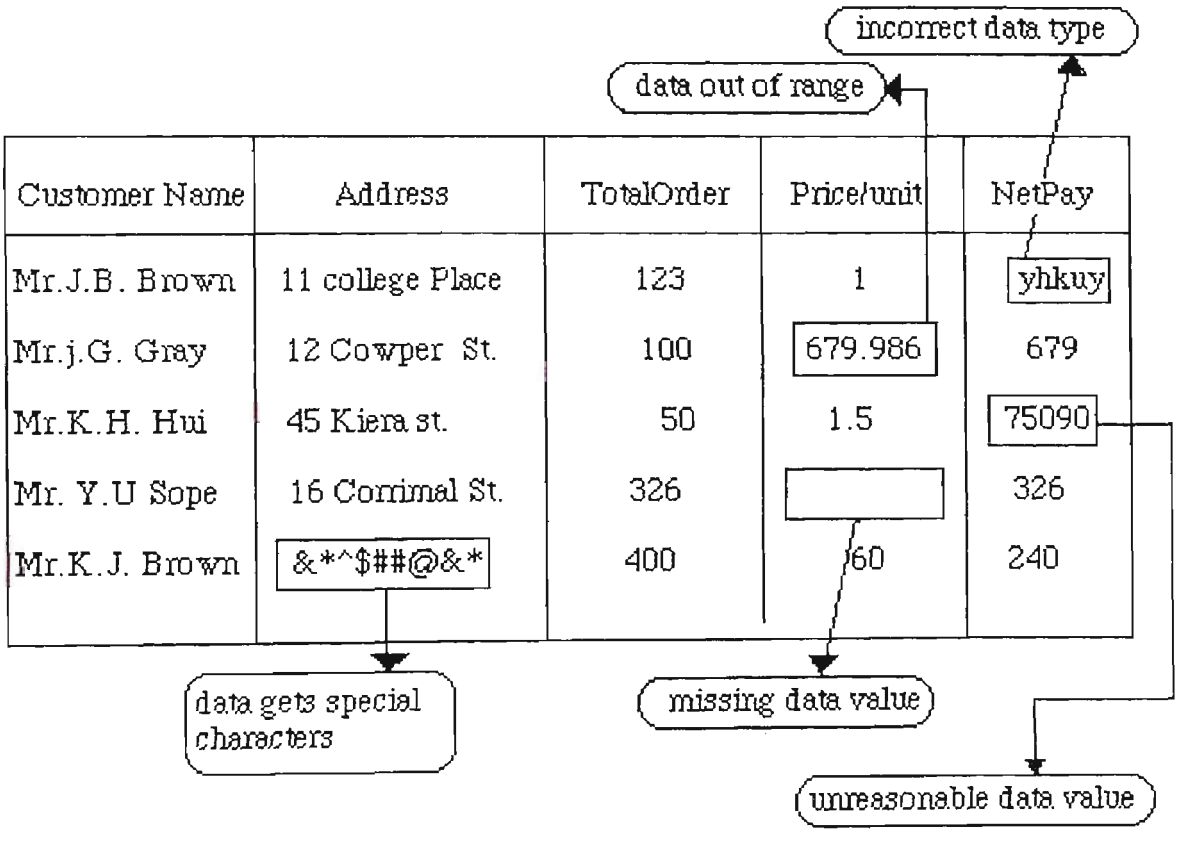


Table 3.1 An Example of Unreasonable Data.

3.2.4 A Change at a Server

A change at a server could be either of the following:

1. a change of a data schema at the data source.

2. a change of a data value

The change of a data value may affect the timestamp consistency of data in the HDDS but not disrupt the user application program, while the change of a data schema can affect every application program that uses the old data format to retrieve data from the source.

3.2.5 Implementation of a Local Database

According to [HSIAO & KAMEL,1989], every data model can be transformed to be a relational data model, so we use relational notations consistent with those in [MAIER,1983] as far as possible. However, the consistency of data contained in the local database must be determined and it must be proven to ensure that users can obtain the same data from this database as from the remote repository. The following proof is due to [NICKOLAS,1992]. We make the following assumptions.

1. The database has a single relation T.

2. The (designated) key K of T is singleton.

3. The relation T has no nulls.

4. All queries made on T (or at least those which are used to maintain the PDB) are of the form

SELECT *Fields*

FROM T

WHERE *Cond*

where *Fields* is a subset of the attributes of T which includes the key K, and where *Cond* is as usual a Boolean combination of simple comparisons of attributes (of T) with attributes, or of attributes with values. That is, more compactly, each query Q applied to T is of the form $\pi_{Fields} \circ \sigma_{Cond}$.

The assumptions 2 and 3 are probably not essential, but are convenient.

Furthermore, we will model a local database system as a relation over the same relation scheme as T, and denote it by T'. Two variants suggest themselves for initialisation of T'.

1. Initialise T' as an empty table (over the scheme of T).

2. Initialise T' with one row for each tuple of T, with the key values copied from T, and with all other values set to the null value ($\perp$).

Whichever of these variants is used, maintenance of T' is thereafter the same. Let Q be a query $\pi_{Fields} \circ \sigma_{Cond}$ applied to T, where *Fields* is $KF_1F_2...F_n$. Note that for each tuple $s \in Q(T)$, we have $s(K)$, $s(F_1)$,..., $s(F_n)$ non-null. For each such s, proceed as follows.

(a) If there exists $t' \in T'$ such that $t'(K) = s(K)$, perform an update

CH(T' ; K = s(K); F_1 = s(F_1), ... , F_n = s(F_n)).

(b) If there does not exist $t' \in T'$ such that $t'(K) = s(K)$, perform an addition

ADD(T'; K = s(K), F_1 = s(F_1), ..., F_n = s(F_n), $G_1 = \perp,\ldots,G_m = \perp$),

where $G_1,\ldots,G_m$ are the attributes of T (or T') other than K and $F_1,\ldots,F_n$.

The following result gives most of the information we need about the properties of T', whichever form of initialisation is used.

Lemma *Let K be the key attribute of both T and T'. Suppose that t' ∈ T'. Then*

i) $t'(K) \neq \perp$;

ii) *there exists a unique t ∈ T such that t'(K) = t(K); and*

iii) *if t'(A) ≠ ⊥, for some attribute A, then t'(A) = t(A), for the above t.*

Proof We prove the result by induction on the number of queries (and consequent maintenance steps) performed.

At the start, before any queries have been made, there are two cases, depending on which initialising step was used. If T' was initialised as an empty table, the required statements hold vacuously. Consider the other alternative-initialisation of T' with one row for each row of T, with key values copied from T, and with all other values set to $\perp$. Then for $t' \in T'$, (i) and (ii) above are immediate, and since $t'(A) \neq \perp$ implies that A = K, (iii) holds because of (ii). Thus (i), (ii) and (iii) holds initially.

Now suppose that (i), (ii) and (iii) hold for all $t' \in T'$ before the execution of a query $Q = \pi_{Fields} \circ \sigma_{Cond}$ and the subsequent maintenance step. Suppose also that *Fields* = $KF_1F_2...F_n$. We fix $t' \in T'$ and prove (i),(ii) and (iii). First note that if t' was present in T' before Q, and is unaffected by Q (and the maintenance step), then (i), (ii) and (iii) are automatic. We consider the case when t' is either created or updated by Q.

(i) Suppose first that t' is updated by step (a) of the maintenance procedure. Note that only one such update occurs, since there can be only be one $s \in Q(T)$ with $s(K) = t'(K)$. Now t' before the update satisfied $t'(K) = s(K)$, and t' is updated by setting $t'(K)$ to $s(K)$, so

t'(K) in fact unchanged. Since t'(K) $\neq \perp$ before the step, this also holds afterwards. Second, if t' is created by step (b) of the maintenance procedure, we have t'(K) = s(K) $\neq \perp$. Thus (i) holds.

(ii) If t' is updated by step (a), then t'(K) is not changed, so (ii) holds by the inductive assumption. If t' is added by step (b), then by the definition of a key, there is a unique t $\in T$ such that t'(K) = s(K) = t(K). Thus (ii) holds.

(iii) Let A be an attribute of T such that t'(A) $\neq \perp$. First suppose that t' is updated by step (a). If A is none of $K,F_1,F_2,\ldots,F_n$, then t'(A) is unaltered by the update, so since t'(K) is not changed by the update either, t'(A) = t(A) holds by the inductive assumption. If A = K, then t'(A) = t(A) holds by (ii). Suppose that A = F_i for some *i*. Now t'(F_i) becomes s(F_i) in the update, and we want to show that s(F_i) = t(F_i). But t is the unique tuple in T such that t(K) = t'(K) = s(K), and therefore s = t($KF_1F_2\ldots F_n$), giving s(F_i)= t(F_i), as required. Second, suppose that t' is created by step (b). Then t'(A) $\neq \perp$ implies that A is K or one of the F_i. If A = K, we have t'(A) = t(A) by (ii). If A = F_i, then t'(A) = t'(F_i) = s(F_i), which as before is t(F_i) = t(A). Thus (iii) holds.

Therefore by induction we have the result.

In order to retrieve data from the local database, we will say that a query Q of the form $\pi_{Fields} \circ \sigma_{Cond}$ (with *Fields* and *Cond* as earlier) is *permitted* (on T') if for every tuple t'$\in T'$, all fields of t' whose attributes names occur in *Field* or *Cond* have non-null values. Only permitted queries are applied to T'.

Theorem 1.1 *If Q is a permitted query, then Q(T') $\subseteq$ Q(T).*

Proof

As before, write Q as $\pi_{Fields} \circ \sigma_{Cond}$, where *Fields* $= KF_1F_2...F_n$. Consider any $t' \in T'$ such that t' is selected by σ_{Cond}. By Lemma, there is a unique $t \in T$ such that $t'(K) = t(K) \neq \perp$. For each such attribute A occurring in *Fields* or *Cond*, the fact that Q is permitted shows that $t'(A) \neq \perp$, and so Lemma shows that $t'(A) = t(A)$ for each such A. It follows in particular that σ_{Cond} also selects t from T (strictly by induction over the structure of *Cond*). Now $\pi_{KF1F2...Fn}(t') = t'(KF_1F_2...F_n)$ and t'(K) and each $t'(F_i)$ is non-null. Therefore $t'(F_i) = t(F_i)$, and so $t'(KF_1F_2...F_n) = t(KF_1F_2...F_n)$. Therefore $\pi_{KF1F2...Fn}(t') = \pi_{KF1F2...Fn}(t)$, and so $\pi_{KF1F2...Fn} \circ \sigma_{Cond}$ produces all the tuples when applied to T that it produces when applied to T'.

Simple examples show that the reverse inclusion does not hold in general if the first initialisation scheme is used. If the second scheme is used, however, the reverse inclusion does hold.

Theorem 1.2 *Suppose that T' is initialised with one row for each tuple of T, with the key values copied from T, and with all other values set to* $\perp$. *If Q is a permitted query, then* $Q(T) \subseteq Q(T')$.

Proof

Take Q to be $\pi_{Fields} \circ \sigma_{Cond}$ as earlier. Let $t \in T$ be selected by σ_{Cond}. Now there is a $t' \in T'$ such that $t'(K) = t(K)$, because of the initialisation step, and because no key values are changed in the maintenance process. Also, by Lemma, there is a unique $t'' \in T$ such that $t'(K) = t''(K)$ and such that $t'(A) \neq \perp$ implies $t'(A) = t''(A)$ for all attributes A of T. Since $t(K) = t'(K) = t''(K)$, we must have $t = t''$. Now as Q is permitted, $t'(A) \neq \perp$ for all attributes A of *Cond* and *Fields*, so we have $t'(A) = t(A)$ for all such A. Hence σ_{Cond} selects t' from T', and by an argument similar to that of Theorem 1, we have $\pi_{Fields}(t) = \pi_{Fields}(t')$, giving the result.

Corollary *Suppose that T' is initialised with one row for each tuple of T, with the key value copies from T, and with all other values set to* $\perp$. *If Q is a permitted query, then Q(T) = Q(T').*

Proof

From Theorem 1 and Theorem 2, $Q(T') \subseteq Q(T)$ and $Q(T) \subseteq Q(T')$. Therefore, $Q(T) = Q(T')$.

As a consequence of the Lemma, the data stored in the local database is related to data at a remote repository. Therefore, when the timestamp of data at the local site is equal to the remote server, Theorem 1, Theorem 2, and Corollary have proven that the retrieved data from the local area are a part of, or the same as, needed data from the original data sources. Thus, reading data from the local database while the timestamp is equal to the remote storage using permitted queries, the consistency of data can be ensured for users.

3.3 Problem Domain

As a consequence of granting read-only privilege to users, we need not consider areas of security control, concurrency control and distributed deadlock. The areas of interest include the design of system software and methods to access data from a HDDS (with respect to different DMLs and data schemas) and to present what data it can be to users under the circumstances of uncontrolled and unpredictable relations between client access and server update. Re-iterating potential areas of difficulties:

1. How can users know that there are some changes have occurred at a server site?

2. When a user implements a local database by copying some significant data from a HDDS, how can the system software maintain consistency of data in this local database with the remote data?

3. Unavailability of remote data to a client may be caused by disconnection of link, malfunction of the remote data source, etc. How can the system software present some data to users under this circumstance?

3.4 System Analysis and Design

3.4.1 Dealing with a Change at a Server

There are two changes at a server site to consider: a data definition, and a data value. In order to detect a change of a data schema of a service, a creation timestamp for the schema should be recorded at the server site, and forwarded together with the the location of the data schema, such as user_tab_columns[23] to the central database manager. When the schema of data has been changed, the HDDBMS can inform users and automatically update to the new data schema. To overcome the problem of independent update of data bases, a record update timestamp, either explicit or implicit, will be employed.

The notion of consistency we are using is tied to the consistency of the update timestamps. Many aspects of time arise in the real world, three possible aspects of time relevant to databases are described as follows [KIM et al.,1990] [KLAHOLD et al.,1986]:

1. time of storage into the database; eg 20 July 1992;08:30:50

2. data update time or timestamp; eg 12 June 1990;10:35:00

[23] user_tab_columns is the name which is used in the Oracle database.

3. period of data validity eg (12-02-92,11-04-93).

When interdependent data are distributed over the world, it is difficult to use only the data update time to examine consistency. As a consequence of having different time zones, the time in different time areas cannot be compared directly. To compare the data update time values from different time zones, the easiest method is to convert time values to Greenwich Mean Time. Whether time conversion is required or not in an application depends on the relativity between time increments of update and varaiation in time zones.[KIM et al.,1990].

There are two primary mechanisms to represent the time value. These may be defined as absolute time and abstract time.

Absolute Time

The absolute time is the primary method to represent the time value which is of the form YYYY/MM/DD [KIM et al.,1990] or DD-MM-YYYY [RUSINKIEWICZ et al.,1991], where YYYY refers to year, MM refers to month, and DD refers to day.

Abstract Time

The abstract time is used in the "time-concerning event" [KIM et al.,1990]. The abstract time uses some special characters to define the valid time such as *exclamation mark (!)* to define the "before or after a specific instant of time or date", or *on* to define "a particular date", or *at-sign (@)* to define "a particular time" etc [RUSINKIEWICZ et al.,1991], for example, *25-Aug-86!* means 'after August 25, 1986', *on27-May-1990* means on May 27,1990, or *@7.00* means at 7.00 a.m.. All of these special characters can be combined together in order to obtain the specific date/time such as *on17-March-1990,@8.30* means on March 17, 1990 at 8.30 a.m..

Timestamp will be used to determine the data consistency within the entire HDDS. In implementing a timestamp value, we should consider the frequency of data update. If data in each data repository is updated occasionally, such as an Agriculture report which will be updated yearly, it is reasonable to implement a timestamp on each table, see Table 3.2. On the other hand, if data in a database is frequently updated, such as the Telecom databases, then the timestamp of each record should be implemented as a field, see Table 3.3, or in order to avoid the re-creation of an original table, the timestamp table will be implemented - Table 3.4. To implement a data update time for each record as shown in Table 3.3 or Table 3.4, the DBA of the database can implement a trigger program so that the timestamp of each record will be stored into the timestamp field or table.

Table: Agricultural Product

Label of a Data Update Time Value

20-04-95

Product	Amount (10^6)	Value (10^9)
Rice	5.67098	1.789429
Corn	3.67509	1.006798
Peanut	0.09878	0.008906

Table 3.2 The Data Update Time Value is a Label of a Table in a Database.

Table : Customer_Data

Telephone_no	Update_time	Name	Address
042+230657	04-06-94	Mr. Patrick Bolk	23 Cowper St., NSW 2519
042+214508	05-07-93	Ms. Rosel John	5 Madoline St., NSW 2500
...	...	...	.....
...	...	...	.....
...	...	...	.....

Table 3.3 The Data Update Time Value is a Field of Each Record in a Table.

Table : Customer_Data

Telephone_no	Name	Address
042+230657	Mr. Patrick Bolk	23 Cowper St., NSW 2519
042+214508	Ms. Rosel John	5 Madoline St., NSW 2500
...	...	.....
...	...	.....
...	...	.....

Table : Update_time_table

Telephone_no	Update_time
042+230657	04-06-94
042+214508	05-07-93
...	...
...	...
...	...

Table 3.4 The Data Update Time Table for Each Record in an Original Table.

An alternative method to confirm consistency of data would be to notify clients whenever an update takes place at a server site [GOLDING,1992]. This idea is impractical in the situations for which we are designing.

3.4.2 Using a Local Database

In some organisation, a user such as a Japanese resercher in cable TV company who is working for an advertisement policy might need some piece of data stored in a HDDS so that the cable TV company can run an advertising promotion. With this aim, the researcher wants to obtain information from all cable TV sub-companies, but the information of interest will be only small pieces of data from participating databases. The researcher might implement her local database to keep only the specific data used. Thus, the local database will contain replicated data from original data sources and be managed individually by the researcher. Inconsistency of data in the local database can arise if the orginal source has been updated, but not this local database.

To maintain consistency of data in the local database, it must contain a timestamp value retrieved from the remote sources so that verification of data consistency in the entire HDDS, including the local database, can be performed. Furthermore, if the implementation date of a

data schema in each data source has been recorded into the local database, a change of a data format affecting the local database can also be detected and notified to the user.

To maintain consistency of the local database, a logging mechanism has been applied. In the logging technique of [CERI & PELAGATTI,1985], every process of a transaction will be stored in a *log file* We can consider our local database as a log file, and a change in a repository as a process of a transaction (indicated by a data update time value). When a retrieval transaction has been transmitted to a remote repository, and a change is detected, then remote retrieval starts and the database management performs a synchronised update to the local database.

3.4.3 Unavailability of a Server

There are many reasons that a server cannot conect to a client, such as a disconnection of the communication channel, the network becoming partitioned, or the server system shutting down for maintenance. In order to serve a user while a remote server is not available in the HDDS, the concept of *implementing duplicated databases* on some other servers over the network was proposed by [GOLDING,1992]. The alternative is to make a copy of data in the HDDS into the user database at the client. However, there is a limitation on storing data into the local repository if the participating databases are huge. Under these circumstances the local database should store only the significant data in which the user is interested. We will use the timestamp comparison described above to ensure the local database will be consistent with other databases in the HDDS.

When some data are not available in the local database or the data in the local database is inconsistent with the original data source then the remote retrieval process needs to be performed. It is possible that during the remote retrieval process, that communication between the client and a server will be terminated for some reason. In this circumstance, the user will

not be able to obtain data from the original repository. A possibility for maintaining service to the user during the disconnection period is for the heterogeneous distributed database management system to shift the client transaction to perform at another replicated source[GOLDING,1992]. Alternatively, if there is no other replicated database in the network, it will terminate the transaction, close the connection, and wait until the sever is available on the network again.

Our design will assume that there is no other duplicate database available in the network except the local database at the client site, which is provided with the aim that the user will be able to obtain some data instead of terminating the transaction. In this situation, our system software will retrieve the previous data with the same key value from the local database, adding the data timestamp to warn the user that the data presented belongs to the previous timestamp period. The user need not terminate the task if that is acceptable, and the system software will connect to the required server whenever the connection can be re-established.

A *Computer Software Interface (CSI)* has been developed to demonstrate the heuristic solutions presented above. Our design contains three components, a client, servers, and the protocol for exchanging information. The design will focus on the client software to serve users in the retrieval of data only. As mentioned by [STALLINGS,1991] TCP/IP enables communication between heterogeneous computers. Despite the minor differences in implementations of TCP/IP, it has become the default industry standard and has the support of a large number of vendors. TCP/IP is therefore the obvious connection protocol to transmit data between clients and servers.

3.5 Computer Software Interface (CSI)

A Computer Software Interface or CSI has been developed to facilitate accessing of data from a HDDS and supporting the presentation of data to users. The result of a query on the CSI is relevant data, or an error code. The CSI function is defined as follows.

Definition 4: A Computer Software Interface or CSI is client software containing retrieval functions to read data from a HDDS and return the result to a user application program.

A primary objective of a CSI is to reduce complexities in retrieving data from a HDDS. Desirable characteristics of a CSI are:

1. A CSI is a user friendly program, using standard commands which will be interpreted to perform all retrieval processes. Users need not deal with the details of different retrieval statements for different data holders. Furthermore, it provides some security control.

2. A CSI generates suitable commands to access data from a HDDS.

3. A CSI should be built on an open system connection protocol. Our CSI is built using the ubiquitous TCP/IP.

4. A CSI is able to check all available services of servers and inform users whenever changes occur.

5. A CSI is able to automatically update information of servers services, which are stored in the CSI storage.

6. A CSI checks the timestamp of data after a retrieval process starts.

7. A CSI attempts to present data to users even when the communication fails .

8. If the user can detect data which is obviously wrong or has a data definition that is not consistent, an efficient CSI might contain a function that can support an approximation routine for a user application. Ours contains a mechanism to do so.

9. A CSI is able to choose suitable servers to minimise cost and retrieval time, if data are held in more than one place.

10. A CSI is able to present information on available services to users. Thus, users are able to check the available services before or after receiving a warning message from the CSI. Moreover, users should be able to check their priority and scope of accessed data.

In order to reduce the difficulties in retrieving data from different servers, the user program will use embedded CSI standard commands which are described in Chapter 4. If a user application program is written in a language different from the CSI, then, a pre-compiler will be needed. As with other heterogeneous distributed database systems, our CSI allows users to query the available data sources on the network by retrieving information from a local database that stores information of all services in the HDDS. The query language in CSI is SQL-like in the following format.

present *<column_name>/<variable_name>*
interested area [is/are] *<table_name>/<file_name>*
under condition *<only one condition allowed>*.

All variables in the "< >" must be defined and registered in the CSI. At the **present** command <*column_name*>/<*variable_name*> must exist under the name of the area of a user's interest. The **interested area** must be a <*table_name*>/<*file_name*>.

The processing of a query is separated into three main components. The first step is to check the local access rights of a user, verify required variables such as column names, obtain data addresses, and establish links to servers. The second step is to generate suitable statements to access data from participating servers, to transmit queries to the required servers, to recover the process if communication between the client and a server fails, and to determine the consistency of the data being retrieved. A mechanism is available to supply data for the user when communication between the client and a server is unexpectedly unavailable.

Figure 3.3 illustrates the components of the system software designed in this thesis which are an Information Server System (ISS), a Query Generator System (QGS), and a Preserved Data System (PDS). The functions of the ISS are to check user authority, to resolve the addresses of the required servers, to check for any changes of services and data values of those servers, to decide the lowest cost and fastest servers in the situation that data are duplicated, and to open and close connections between the client and servers. The function of the QGS is to generate suitable commands for each data source as required. The QGS also controls the retrieval process if communication between the client and a server fails. It contains an approximation routine which can suggest an approximated value if one is required. The PDS logs incoming data and is able to supply data back to the user if the normal mechanisms fail. The cooperation between these three subsystems is illustrated in Figure 3.4.

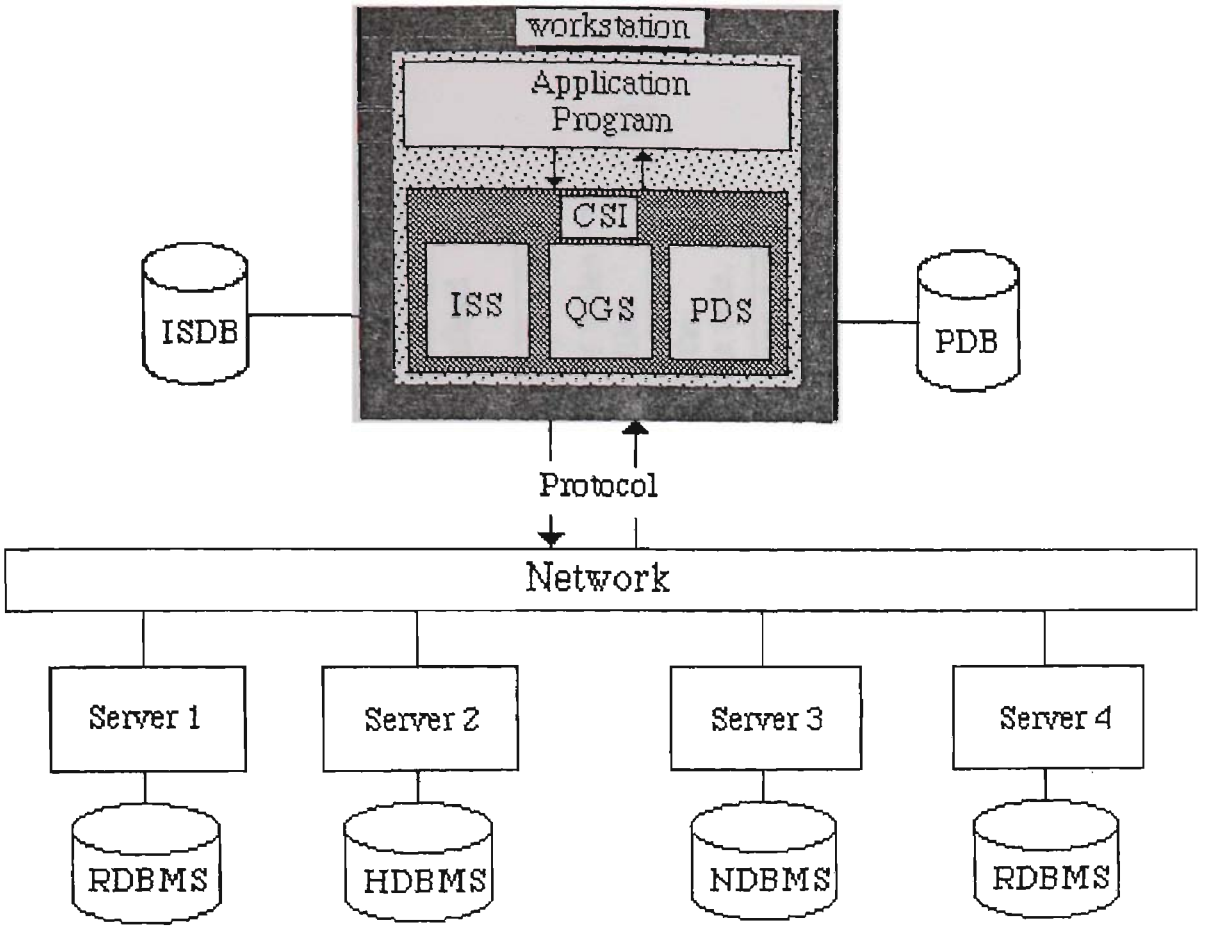


Figure 3.3 The Design Environment of a CSI over a HDDS.

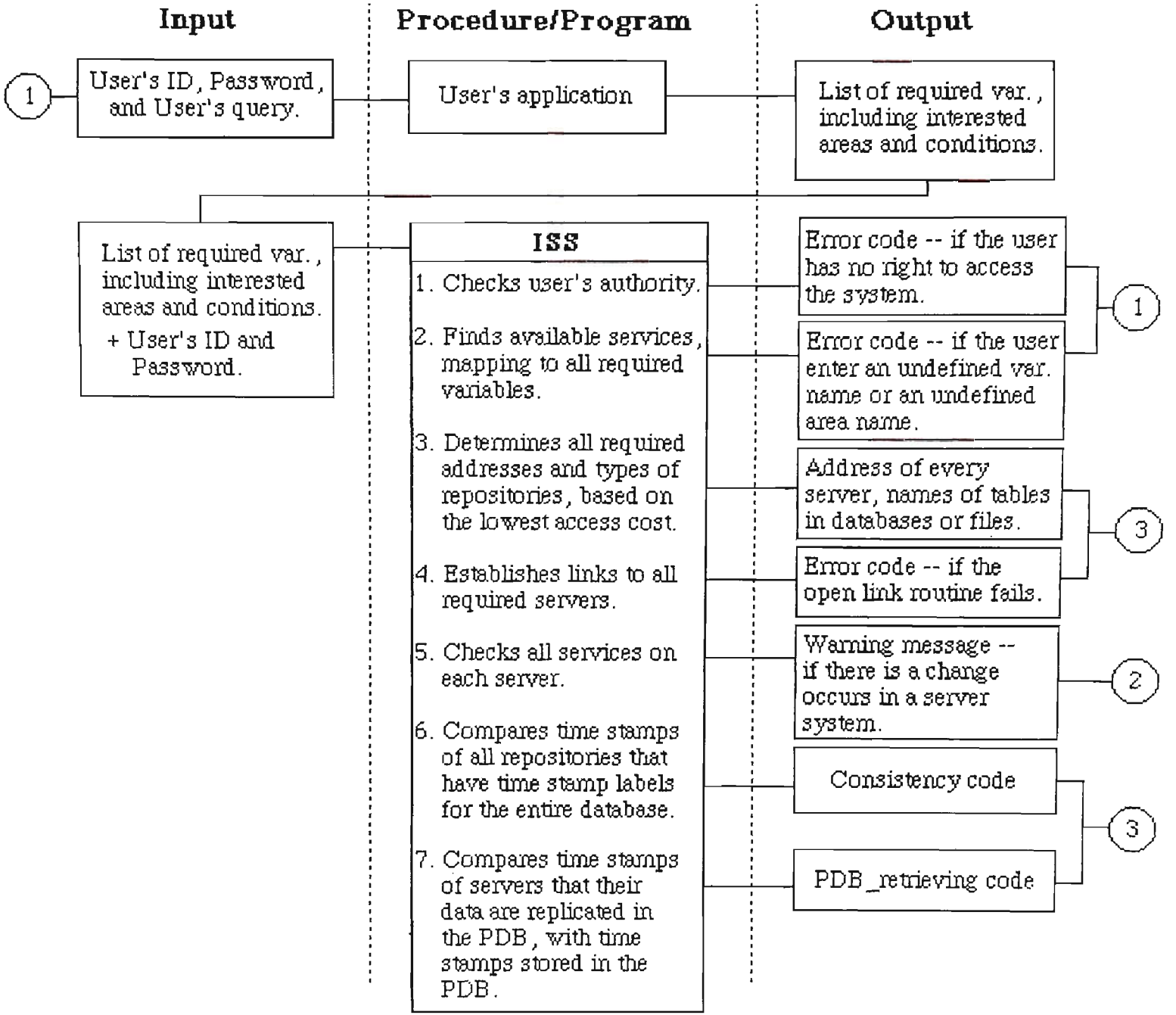


Figure 3.4 Coopreation between Three Subsystems in the Retrieval Process.

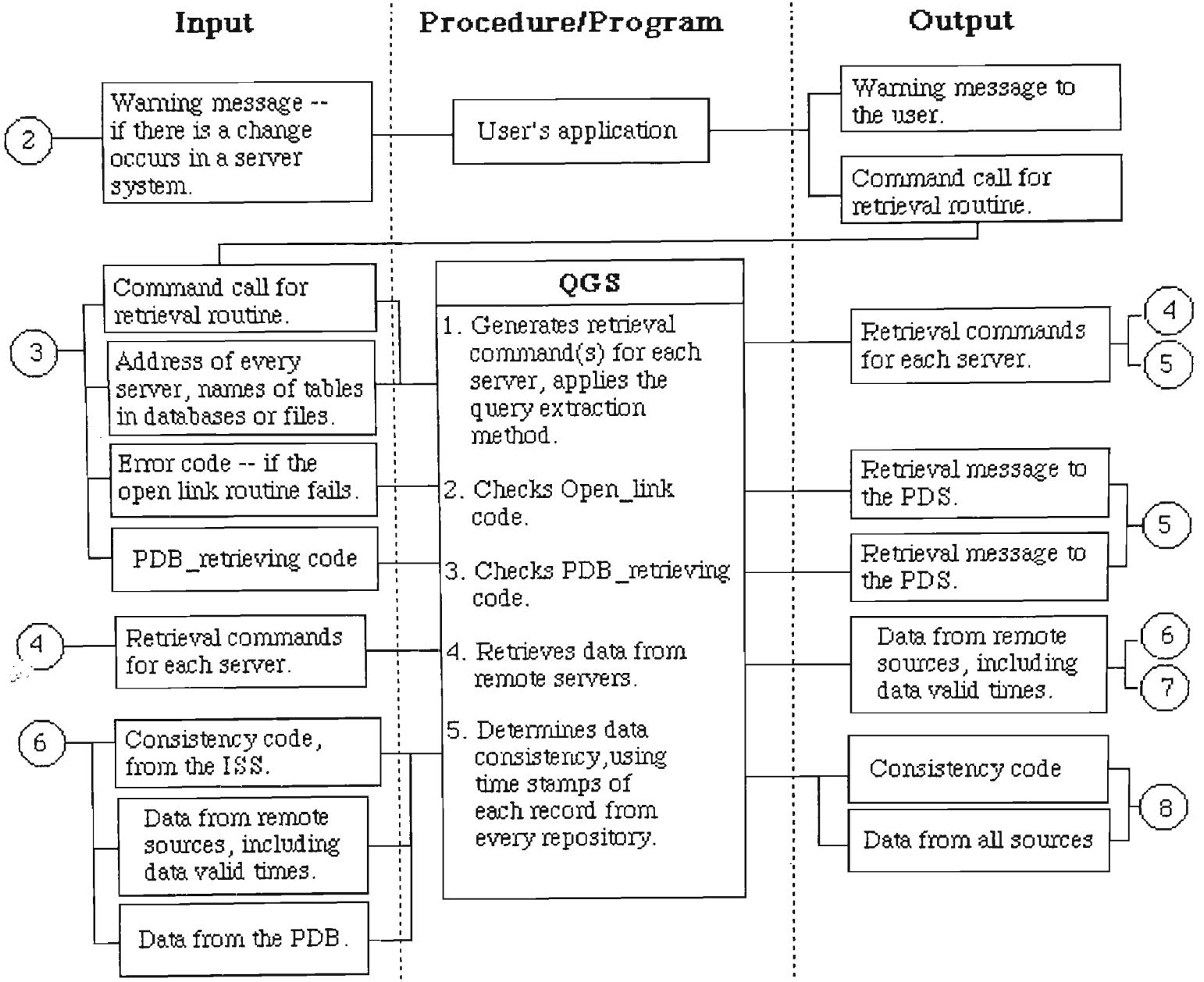


Figure 3.4 Coopreation between Three Subsystems in the Retrieval Process (Cont.).

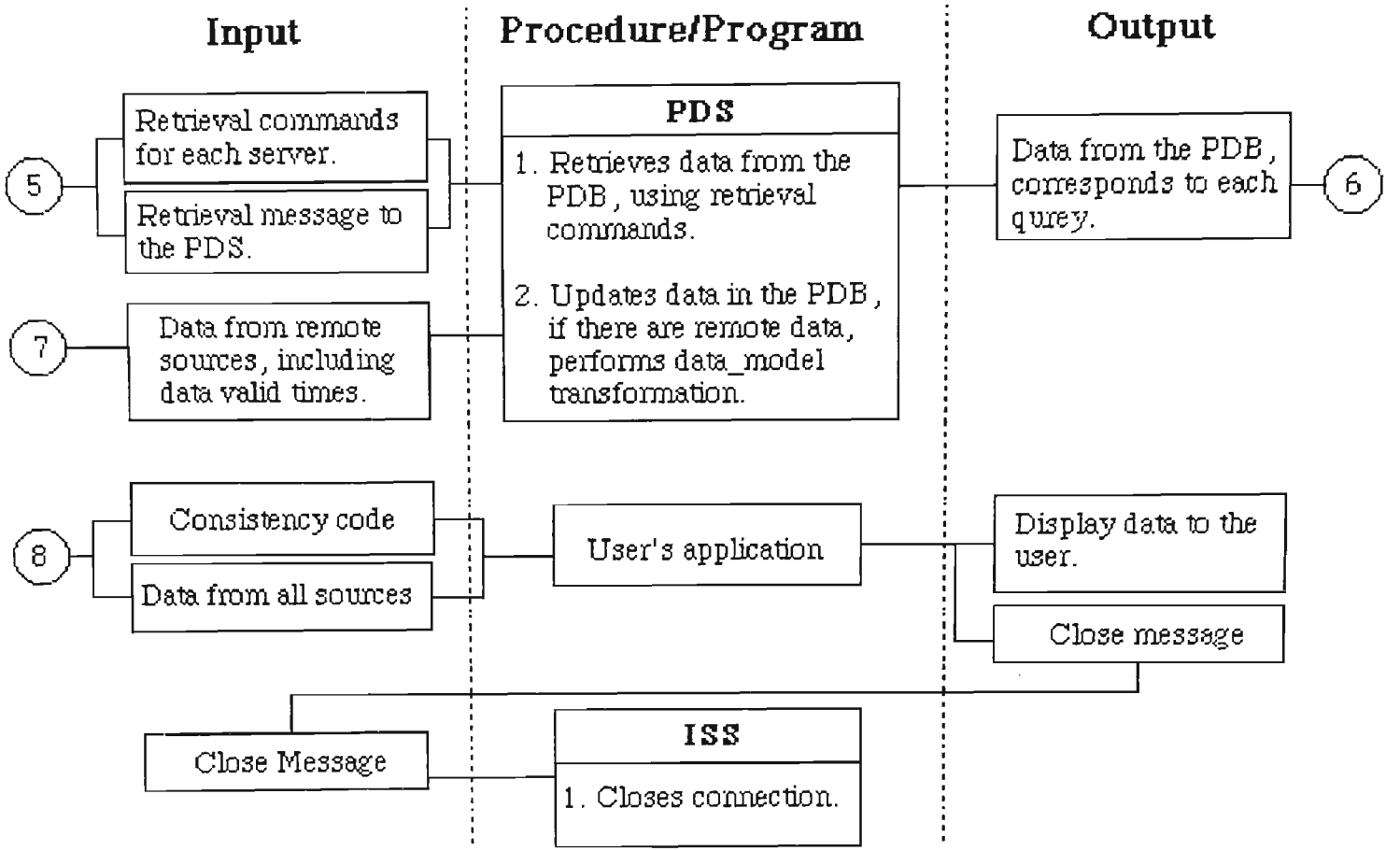


Figure 3.4 Coopreation between Three Subsystems in the Retrieval Process (Cont.).

3.5.1 Information Server System (ISS)

In the systems we discussed in Chapter 2, in each case there is a database that stores information of server services, viz. the DATAPLEX dictionary, the WAIS dictionary of servers, and ANSA traders. We noted that the manual updating of these databases could result in out-of-date information about available services being offered to a client. The information kept for each server is the location of the service, type of the service, data definition for the service, access method, and cost of access. A change of a service may occur if the server discards the service, or the server alters the service definition[24]. The effect of altering information at a server site without updating the server information database will be errors in user applications.

The first objective of the ISS is to find all locations and formats for the data specified in the user query, thus avoiding the problem of a changing data schema affecting a user's application as it might in DATAPLEX, WAIS or ANSA. Moreover a change of a data value at a server site can be detected and the user informed, if the timestamp is implemented as a label of the entire database. If the timestamp is attached to individual records, the data will have to be fetched anyway for examination.

Definition 5: The *Information Server System (ISS)* is the subsystem that allocates all available services, assigns server addresses to variables for retrieval, controls local access rights, and establishes links to servers.

The ISS uses a local database called an Information Server Database (ISDB) where the information related to the user data area and server information area are kept. The information in the ISDB will be updated by ISS after the ISS indicates the differences between the

[24] The service definitions are such as available data models, data interpretation, etc.

information in the ISDB and the current information from the connected server. The ISDB will be partitioned into two data areas: user data area and server information area. The user data area stores information of authorised users in the HDDS, in the same way as the user database is kept in a normal DBMS, and will not be described here, while the server information area will be a repository for all available services offered by servers.

In consideration of the data model for the ISDB, the chosen data model must be efficient in supporting the searching process. Besides, the chosen data model should be compatible with the file structure. The structure of the file system is similar to the hierarchical model which implies that the ISDB data model should be implemented as a hierarchical model. Although there are four models in the database area, the relational model, hierarchical model, network model, and object-oriented model, the hierarchical model was selected for the following reasons.

Using the relational data model, a database must consist of relational tables, each table consists of attributes and each attribute consists of a number of related constraints where each constraint must be well defined. However, it is possible that relations between some tables from different data sources cannot be implemented as clearly as required. Moreover, storing the information of the structure of the file, using this model, the information must be converted to the relational model which is not efficient. So, the relational model is not suitable for the ISDB. Consider the network data model. This data model is also not the right model for ISDB because in this Thesis there is no relationship across data tables or files from different repositories. For the object-oriented model, it has no unique model but the structure of a database must be classified to be class, subclass, and sub-subclass. Inheritance between classes can occur. Thus, using this model to store a structure of a file does not fit the requirements of the database. Therefore, the object-oriented data model is not suitable for the ISDB. In conclusion, the hierarchical model is the most suitable for the ISDB, as shown in Figure 3.5.

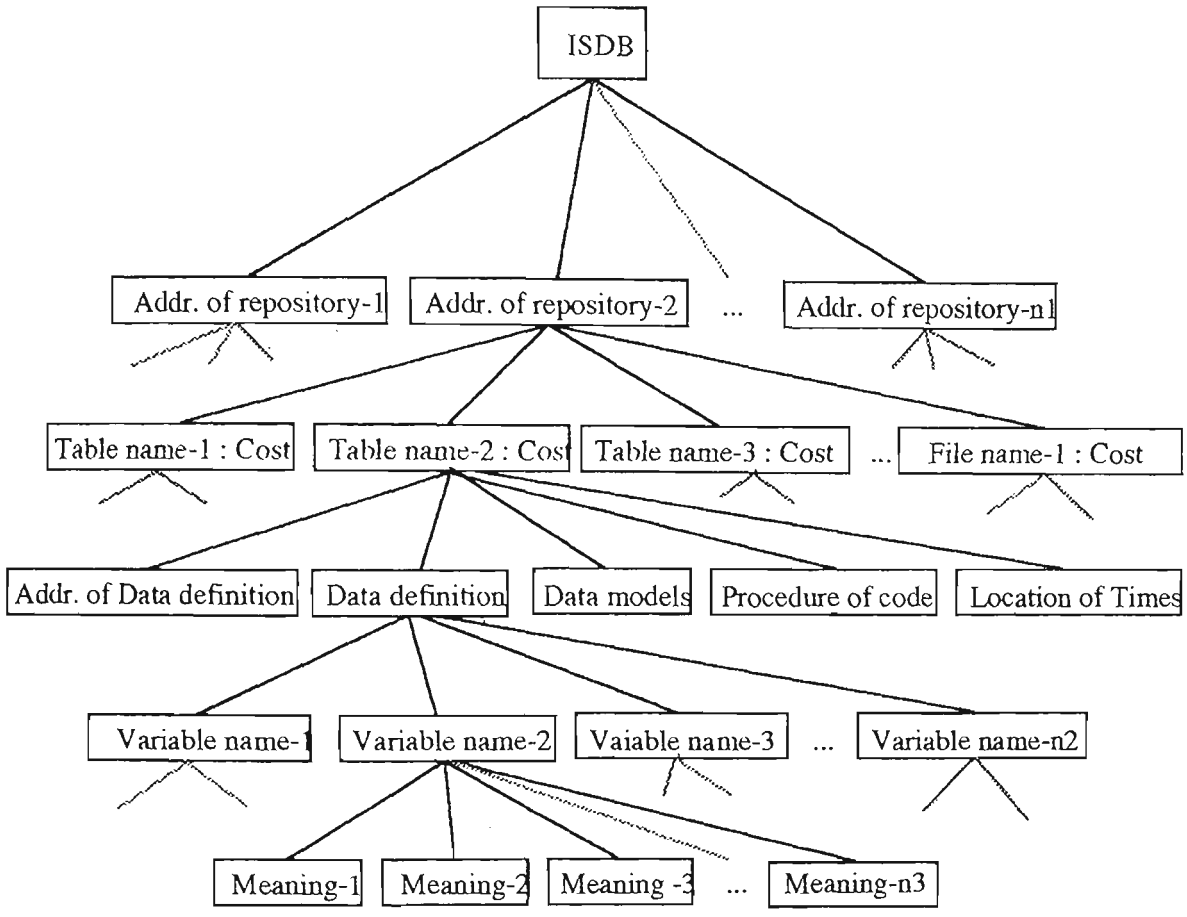


Figure 3.5 The Data Structure of the ISDB.

To allow automatic updating of this database, each server must register the location of information about itself, such as the name of its data definition table in the database. For example, locations of data schemas in a database system on the server have to be defined and registered with the ISDB. The information stored in the server information area will include:

1. addresses of sharable databases or files;

2. names of sharable tables or files or services;

3. locations of data schemas, the data schemas themselves, data models of databases or files and a number that can be used as an indicator to choose the command generator procedure;

4. locations of the timestamp value (timestamp label's address or timestamp table's address or tiemstamp field in the required database) and the creation time of data definition at the remote database.

5. meaning of each defined data column or variable such as Payment means Gross payment or Net payment. This cannot be used to resolve semantic questions, without the use of a universal vocabulary.

6. all access costs.

Data files often do not contain a data definition header defining the data that they store. For our system to work properly, this information will have to be stored somewhere, either at the front of the file, or in a dedicated file of data schemas. If the latter is used, the ISS can monitor changes on the servers and update the ISDB, whenever connections between the client and servers are established. If there is a change at the server site, the ISS will inform the user. The ISS can perform the following functions:

1. An ISS can check the access rights of users of the client system, as a first security control measure, before the access rights to remote data are validated by servers.

2. An ISS stores all available services of servers, and their access costs. This information is registered by server administrators when granting access to users of the ISS.

3. An ISS is able to select sources of data required by a user, based on the cheapest access cost and data update time values of every server that has a label of the timestamp, including the Preserved Database (PDB).

4. An ISS can check and inform users when a change related to the data definition arises at the server site, using the data schema's create date value.

5. An ISS can automatically update services of servers in the ISDB whenever the details of required data stored in the ISDB is different to the real information stored in the server database system.

6. An ISS performs open-link and close-link routines to all required servers. Every link is opened until the user receives complete information for one query; consequently links for each user enquiry may be different.

The first function of the ISS is to check a user's authority in the local system. The objective is to protect the data against unauthorised use. Once authorised, users are able to use or obtain access on the local database PDB, which will be described in the next two sections. As a further security control, a remote check may be made by servers.

The outcome for each service may be represented by an *(i,j,k)* triple, where *i* stands for the address of the required data, *j* represents the type of data source (such as RDBMS or file), and *k* is the required data name, entered by the user. After finding all details of the required data, the ISS will send these results to the next subsystem. If a data address cannot be resolved, an error will be sent to the user's program which will decide what to do next.

To accommodate many kinds of data sources, we decided that links between a client and servers should be opened and closed for each user enquiry. This implies that connections between the client and servers depend on which servers are needed to service each query. So, links between the client and servers of query_1 may not be the same as links between the client and servers of query_2.

After the ISS issues links to all servers, the data schema creation time value on each server will be sent to the ISS. The ISS compares all these time values with the created time value of each remote database stored in the PDB so that when a change of data schema occurs, the PDB will be able to detect and inform the user of this change. In the situation that the timestamp is implemented as a label of the entire table of a database, as shown in Table 3.2, this timestamp will be sent to compare with the timestamp of that table which is recorded in the PDB. Thus the possible change of a data value at the original source will be detected, or the consistency of data will be confirmed. The PDB can be used as a server in the HDDS when the the comparison of timestamps are equal.

Once the ISS confirms a change of data schema at the servers occurs, it will update the ISDB and warn the user of the change. As an objective of the CSI is that the user is able to check the available services details on the system, the ISS will perform the retrieval process to read all services defined in the ISDB and present them to the user when required. These details are recorded in the ISDB. The services information will be presented based on the areas of interest.

The last function of the ISS is to close all opened links after the retrieval process has been completed. The user application program uses a standard command to initiate closing links with all servers.

3.5.2 Query Generator System (QGS)

In a heterogeneous database, data-model transformation, data-language translation, and cross-model accessing are required [HSIAO et al.,1989]. The function of the data-model transformation is to transform one data model from a data source to another data model. The data-language translation is included to resolve the problem of handling different data

manipulation languages. The cross-model accessing is similar to the data-model transformation, but it transforms the data model from the data source to the data model defined by a user.

Data-language translation is performed in our Query Generator System (QGS), which uses the idea of data-language translation to develop and implement a data-language generator to access data on different servers, so that syntactic differences can be hidden from the user.

Definition 6: The *Query Generator System (QGS)* is the subsystem that generates commands to retrieve consistent data from the HDDS without any user intervention.

The retrieval commands generated for each server are based on information supplied by the ISS. Therefore, the QGS is *active* if and only if the ISS is able to obtain details of required services. Each generator routine will consist of retrieval command(s) which are suitable for the type of server on which the data is held. The information from the ISS which are the data model (such as Relational) and type of data source (such as DB2) will be the indicator for QGS to select the right procedure for query generation. For example, a command generator named Oracle_command will be implemented to generate retrieval commands for every ORACLE server.

QGS is separated into three main parts. The first is responsible for the generation of retrieval commands. The second retrieves information and sends outcomes to the user. The second part also checks for consistency of data from different areas before returning the outcome to the user's application program, using the timestamp values retrieved from every data source, irrespective of the form in which they have been implemented. The third part can offer an approximation where the user believes that a data item is incorrect.

The retrieval process of QGS starts after receiving results from the ISS. The reading process on each server can encounter two particular situations. The first is the normal situation - communications between the client and servers are established. The second is when the communication between the client and servers fails. Usually the user is interested in some particular area, so the situation that a server is unreachable can be overcome by implementing the PDS and the PDB because the amount of information to be stored in the local database is small compared with an original source. If the user is accessing large databases subject to continual update, as in the Telecom database example, the loss of connection implies all functionality is lost.

After an appropriate command is generated for each participating server, the retrieval process begins. The method of decomposing complex queries into simple queries is similar to that of the Distributed Query Decomposer of DATAPLEX [CHUNG,1990]. The result from each simple query will be merged to obtain the final result for presentation to the user. An example of extraction from a query is illustrated in Figure 3.6. The objective of performing extraction from the complicated query is that the result from each simple query will be storable in a local database.

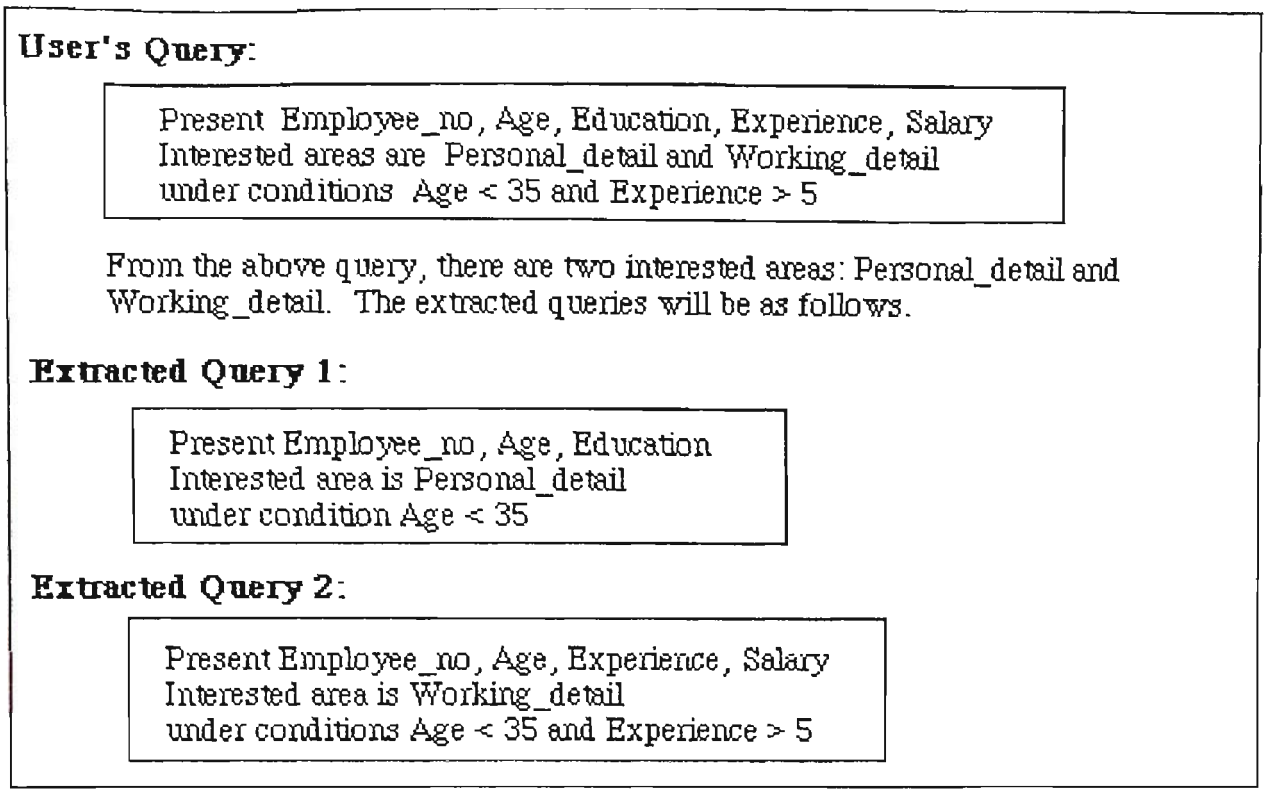


Figure 3.6 An Example of Extraction a Query.

Suppose that a user is interested in some piece of remote data. The client may not be able to communicate with the server when a link between the client and the server is closed, or the server system is not switched on, or the network becomes partitioned. In other systems like ours, when communication between the client and a server fails, the client process would be terminated. We would like to present some data to users even if the servers do not contain duplicated data sources, so our QGS process co-operates with a subsystem called a Preserved Data System (PDS), which we now describe.

The QGS retrieves data from each repository using simple queries extracted by decomposing the original user query. After the QGS received data and if the information in the ISS indicates that the timestamp was assigned for each record for a data source, then the data consistency has to be confirmed by comparing the timestamp values of the individual data. After the comparison, data will be marked as either consistent data or inconsistent data.

The results of the retrieval process are data from each relevant server. These will be merged and presented to the user as a table in a form determined by the user application program and based on the consistency flag obtained from the QGS. On receiving data, the user might be able to see that a data item is in a format that is unreasonable, for some reason. In this case, the user might prefer an approximation to the item if our system can supply one. In this thesis, an approximation of data can be attempted because there is a local database, PDB, available at the client site. The approximation command will be embedded in the user application to be invoked whenever estimation is required. The approximation function in QGS determines whether approximation can be done mathematically or by retrieving previous data from the PDB. If the data can be estimated mathematically, the QGS will retrieve related values that can be used as input to the mathematical model. Otherwise, the QGS will send a message to the PDB to retrieve data for the same resource name from a previous time period.

3.5.3 Preserved Data System (PDS)

The PDS will be implemented at the client assuming that a user uses only some piece of data in the HDDS which is not replicated on another server. When the transaction from the client cannot access the server, the transaction would eventually time out. In such circumstances, the client program may wait until the communication becomes available again or adopt one several methods to avoid termination of transaction while the client is not able to communicate with its servers. The method we use to ensure that information is presented to users while communication between the client and a server is not available is to implement the *Preserved Data System (PDS).*

The PDS will cooperate with a local database called a *Preserve Database (PDB).* The PDB is developed from the concept of *implementing duplicated databases* on some other servers over the network [GOLDING,1992], and a recovery method -- *logging mechanism*

[CERI & PELAGATTI,1985]. The PDB will store only those items the user has queried from all data repositories in the HDDS. In order to maintain consistency of the PDB, the logging mechanism has been applied, in that every process of a transaction will be stored in a *log file.* Consider the PDB as a log file, and that a change in a repository is a process of a transaction. Therefore, when a retrieval transaction has been transmitted to a remote database or a file, and ISS confirmed a change on that source, the remote retrieval starts and the PDS performs the synchronise update to PDB.

Definition 7: The *Preserved Data System (PDS)* is the subsystem that supports the retrieval process under some circumstances such as when communication between the client and servers fails.

Definition 8: The *Preserved Database (PDB)* is a local database at the client node that contains data retrieved from remote servers.

The PDS acts as the database management system of the PDB. In designing the data model of the PDB, the following points were considered. The model should be easy to implement, easy to understand, and should be able to represent other data models. As noted by [CHUNG,1990] all of these criteria are met by a relational data model, so we have used it. Because the HDDS may contain different data models, data-model transformation may be required. The information which is stored in the PDB is listed below.

1. The data source address and name corresponding to the stored data.

2. The update time of current data when retrieved from its server.

3. The created time of the current data schema as issued by the server.

4. The retrieval constraints which users use to limit the area of interest under the same timestamp. These values are used to confirm that the required data are available in the PDB.

5. A list of variables or column names plays the role of data definition of the original database.

6. The data values themselves.

The functions of the PDS which are performed on the PDB are defined as follows.

1. The PDS updates information in the PDB whenever the QGS receives new data from the servers.

2. The PDS retrieves information from the PDB when processing a transaction from the QGS.

3.5.3.1 Maintaining the PDB

As a result of extracting simple queries from a complicated query, the PDB is able to store data from simple queries on retrieval. The primary operation that the PDS performs on the PDB is the insertion of new data when updates on the servers are detected. Based on the maintenance algorithms in Section 3.2.3, the PDB can be considered as a subset of every primary source.

Consider the case that approximation of data is required. If the approximated data cannot be calculated mathematically, then the approximate value should be obtained from the previous data, which should be kept for the purpose. This historic data may also be presented to the user if the client cannot communicate to the server. However, it is not worth keeping all obsolete data, so we keep previous data from *one step before* the current value. Thus, the PDB

will contain up-to-date data and the out-of-date data which were valid one step before the current values.

3.5.3.2 Querying the PDB

The primary objective of implementing the PDB is to support users gaining information while the client cannot connect to the server, or to service queries quickly and cheaply if the required data items are present and up to date. The result from the PDB (the effect of each extracted query) will be sent to the QGS and let the QGS manage all data. In the case of using the PDB for a long time without remote updates, the PDB will contain almost all significant data. In the case wherein the current update time is the same in the PDB and remote servers, the result from the PDB is a subset of the primary result from the remote sources.

As mentioned in the previous section, data in the PDB will be used in the approximation routine of the QGS. Whenever the QGS determines that the approximated value cannot be calculated mathematically, the QGS will send a message to the PDS in order to retrieve previous data from the PDB. The PDS is able to retrieve data from the PDB using the address, source name, and service name (or variable name). The result of searching the previous data is either success or failure. It is success if the previous data was stored in the PDB, otherwise, an error message is returned to the QGS.

3.6 Summary

The objective of a user is to gain access to relevant documents. The conditions of the system environment are that users cannot control all changes of data, as described in Section 1.2. Moreover, data in the HDDS are interdependent data with relationships possibly defined by a user's view point. The protocol guarantees the correctness of transmitted data but

communication between a client and servers may not be available sometimes, and there are no duplicate databases available in the HDDS except those provided locally by our software to a user.

Three main problems we have considered are out-of-date information in a database in a HDDS, inconsistency across the entire HDDS including a local database implemented at the client system, and failure of communication between the client and servers.

To solve the first problem, an Information Server System (ISS) has been developed, which co-operates with a local database system called an Information Server Database (ISDB). The ISS has four main functions. Firstly, the ISS performs security control, checking the user access rights locally before checking at the remote servers. Secondly, the ISS verifies and locates variables in user queries. Thirdly, the ISS establishes communication between the client and servers, and terminates the connections when the retrieval process is completed. And, finally, the ISS checks and updates information in the ISDB whenever a change in a service occurs, using the address of the data definition database such as users_tab_columns in Oracle database and the creation time value of the data schema.

Servers have to register the addresses of services and service models into the ISDB during the initial phase. File systems do not usually have extrenally accessible data schemas, but we require that the characteristics of a file have to be stored, in a way similar to the characteristics of a database system, in a location accessible to our system, so that the server can register the data model and address of the data model of the file into the ISDB. Ultimately, the ISS will use this information to update information in the ISDB automatically. Moreover, users are able to retrieve information on available services from the ISDB, to confirm user authorities and the scope of data to be accessed, before or after receiving a warning message.

The Query Generator System (QGS) reduces the complexity in managing retrieval processes from heterogeneous data sources. The main functions of the QGS are to generate suitable commands to read data from remote servers, retrieve information from available sources, and recover processes when a communication between the client and a server fails. The QGS applies the concept of a data-language translator for the DMLs in the various servers to generate a suitable retrieval command for each server. In addition, it may generate simple queries from a complicated user query. The retrieval process uses the extracted queries. After receiving data from all required sources, the QGS determines the consistency of data using the timestamp values of data before assembling and presenting it to users.

The consistency of data will be determined by both the ISS and QGS subsystems, based on the timestamp value presented in each database, or individual record of each database, as appropriate for the frequency of update of both tables and individual records. If the data item is also recorded in the PDB, the timestamp of the data stored in the PDB will be compared with the remote timestamp value, so that the ISS can choose the lowest cost access, including the PDB.

Consider the possibilities for a retrieval process which cannot reach the remote servers. Normally, in these circumstances, the transaction will be aborted. An objective of our CSI is to present data to users even though the communication is not available for a user performing a sustained task, so the Preserved Data System (PDS) has been developed. The QGS cooperates with the PDS to achieve this objective. The PDS stores the results of previous data retrievals from remote servers into a local database called a Preserved Database (PDB), which can be considered to be a subset of the data on the remote servers, but located locally. By reference to Theorem 1.2 and Corollary, we can guarantee that the result retrieved from the PDB is a subset of the results which could be obtained from the remote source, based on the same update time.

In the design, some approximation methods are available to serve a user if there is a local database available at the client system. The first is to apply a mathematical approximation. The second is to present previous data to the user. Whenever an approximate value is required, the QGS will determine whether the required field is numeric or not. If numeric, the QGS could retrieve the related data required to create the approximate value. If not, two previous values are available in the PDB and can be presented.

Chapter 4

Implementation and Evaluation

This chapter describes a prototype system implemented using HyperCard based on the design given in Chapter 3. An ORACLE database and file systems are the data sources. The communication between a client and servers uses the TCP/IP protocol.

4.1 Demonstration Environment

The demonstration environment consists of three major parts: the client, servers and the protocol used to transmit data between a client and servers.

4.1.1 General Assumptions

A schematic of our system environment is presented in Figure 4.1. Our general assumptions are as follows.

1. There are various kinds of hardware, DBMSs, and file systems on a network.
2. All data sources are assumed to be remote.
3. All data on each server are related to each other by the key value(s).

4. A query may require merging of data retrieved from diverse sources.

5. A client system is a workstation with some secondary storage.

6. Our client is not required to update data on the remote servers.

7. The data security, concurrency control and distributed deadlock will be handled by a DBMS of the database system or by a file management system available at any server site through the use of their standard facilities.

8. If a physical disconnection occurs, the client tries to present some information to users.

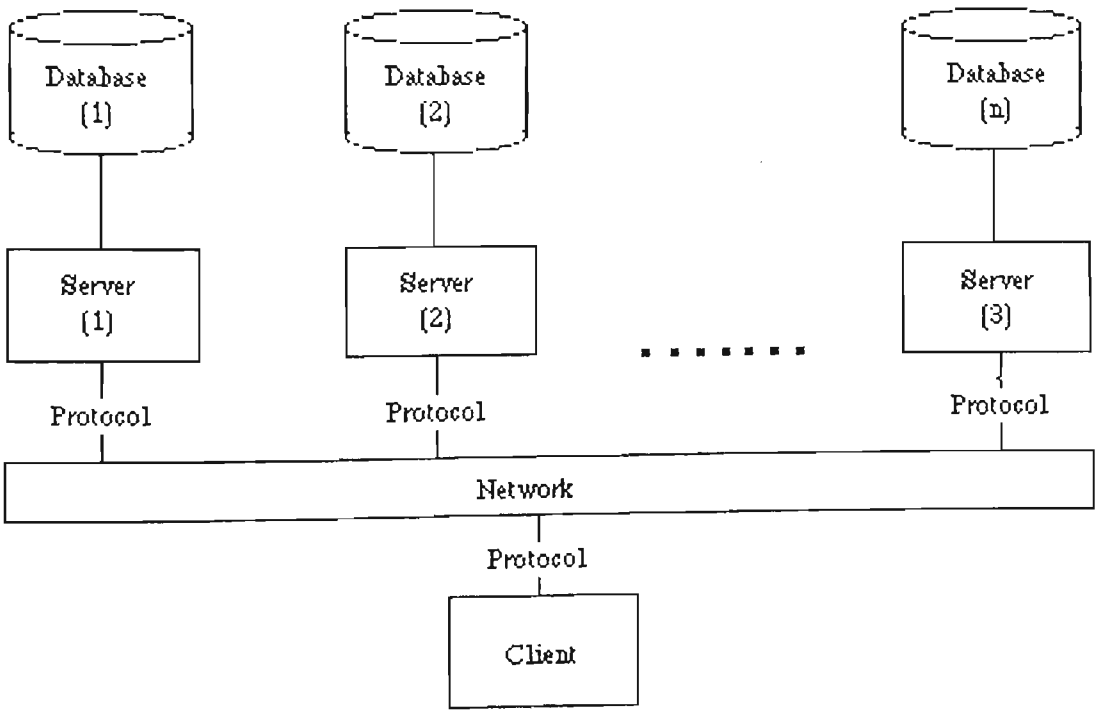


Figure 4.1 The Environment of a HDDS.

4.1.2 Physical Disconnection

A physical disconnection in this prototype system will be handled in the following circumstances.

Assumption 1 : The disconnection occurs after the login process has completed. This means that links between a client and servers must be available at the beginning of the login process.

Assumption 2 : Although a connection is closed, the client and server programs are not terminated. Then the client and the server can reconnect and communicate to each other with their previous status preserved.

Assumption 3 : When a physical disconnection occurs, the client program can check, do the internal close connection, and later try to connect back to the same server.

Assumption 4 : No two physical disconnections occur at the same time. This means that if the connection between the client and Server1 is closed then the connection between the client and Server2 must be available, or vice versa.

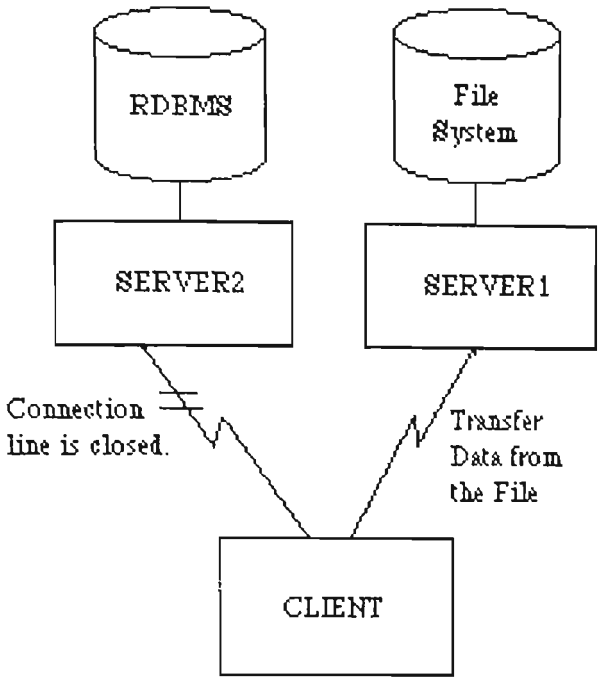


Figure 4.2 Physical Disonnection between the Client and Server2.

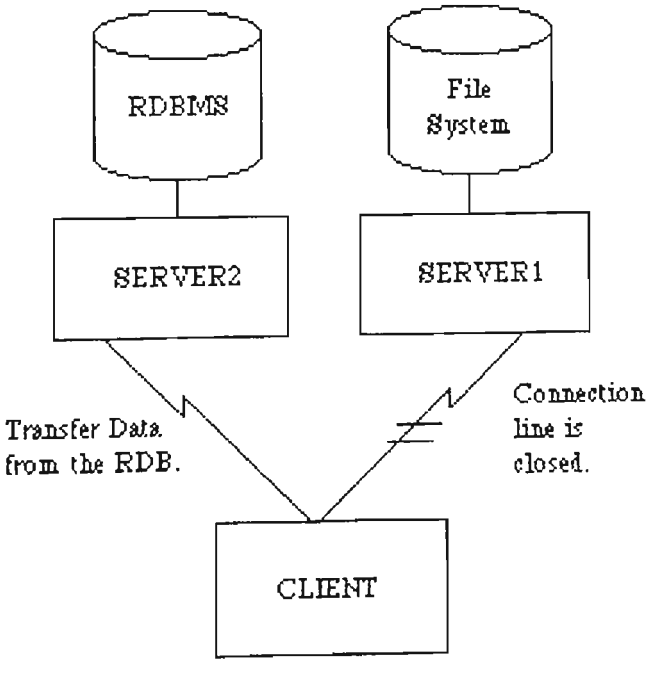


Figure 4.3 Physical Disconnection between the Client and Server1.

4.1.3 Characteristics of Data

As mentioned in the first chapter, our prototype data will be stored in two servers: a file system on Server1 and a relational database (RDB) on Server2. These data satisfy the following assumptions.

Assumption 1: Data on the RDB and data on the file systems are related.

Assumption 2: Data on both systems have the same key values to support search and merge operations.

Assumption 3: The mapping relation between data and key values on the file system is a one-to-one correspondence.

Assumption 4: Data in file systems on Server1 with the same key are related.

Assumption 5: The life interval, or the time stamp value, is a field of the data.

Assumption 6: The timestamp of the created data schema is stored for each table at the data source, as described in Figure 3.2.

4.1.4 Characteristics of This Prototype File System

A file system consists of two subfiles: an index file, and a data file. The index file contains the data definition of the data file, key values and offset of data in the data file. A key value plays the same role as the key value in the RDB. An offset gives the location of the data in the data file corresponding to a key value. The data format is stored at beginning of an index file and has the information listed below:

1. the maximum number of fields which will be sent to a client (assuming fixed number of fields);

2. the name of each field in one record;

3. the type of each field such as integer, floating point, or string; and

4. the length of data of each field.

```
Struct File_index
{
   char       Country[15];
   long int   Offset;
}
```

A: The format of normal File_index which does not contain the format of data in the other file.

```
struct  File_index
{
   unsign int   noField;
   char         FieldName[15];
   char         FieldType[12];
   unsign int   FieldLenght;
   DataStruct   DataKey;
}
struct  Key_index
{
   char         Country[15];
   long int     Offset;
} DataStruct;
```

B : A format of a File_index which contains the format of data in the other file.

Figure 4.4 The Format of Record in the File System.

4.2 System Implementation

The implementation will be based on the design in Chapter 3. To simulate a HDDS, two data resources were created, file systems stored in Server1, and a relational database system on Server2. The data manipulation program on the Server1 was a simple retrieval code written in C and the data management system on the Server2 was the ORACLE RDBMS. Figure 4.5 shows the system architecture of the simulated HDDS with its set of software tools.

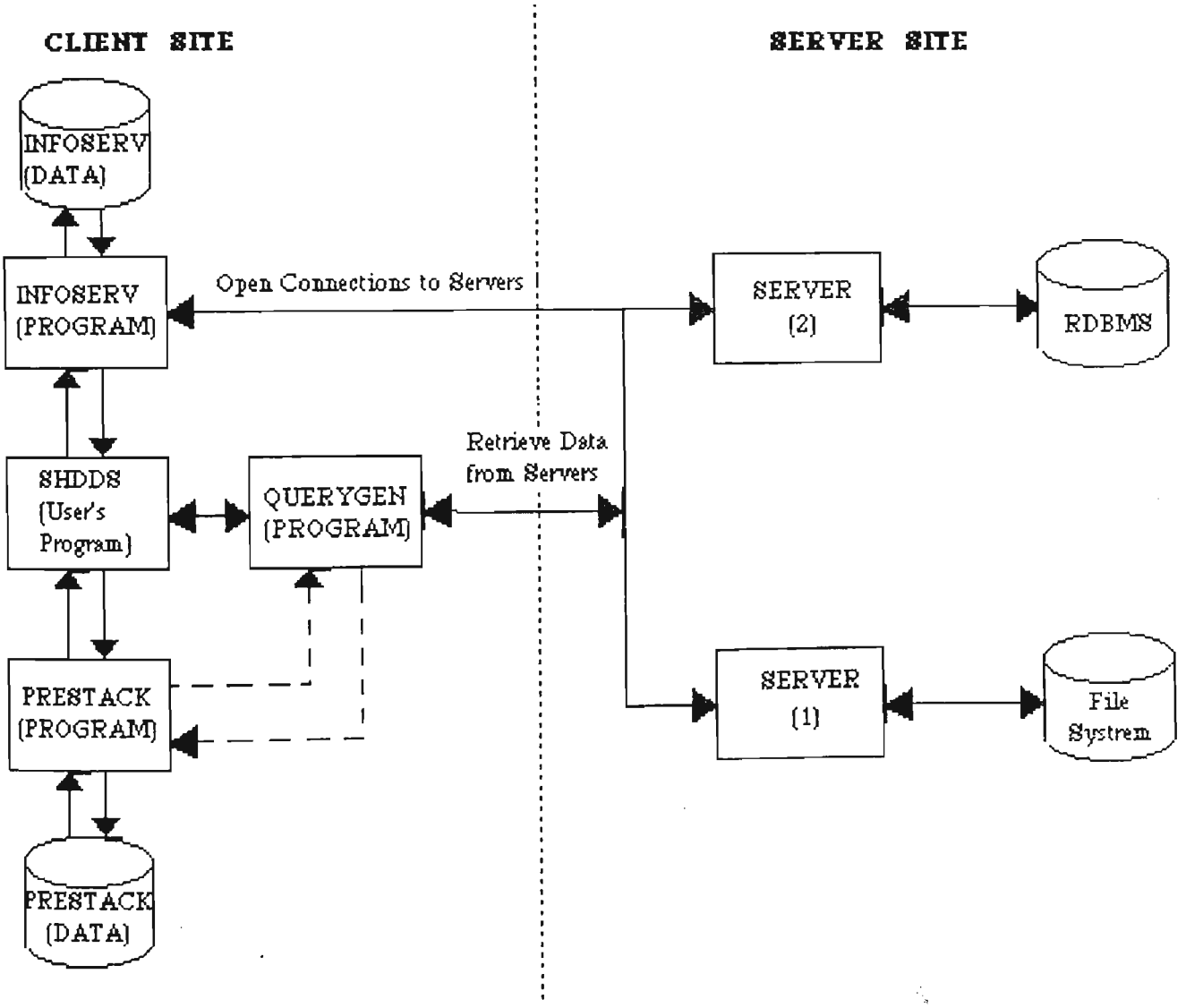


Figure 4.5 The System Design Architecture.

4.2 System Implementation

The implementation will be based on the design in Chapter 3. To simulate a HDDS, two data resources were created, file systems stored in Server1, and a relational database system on Server2. The data manipulation program on the Server1 was a simple retrieval code written in C and the data management system on the Server2 was the ORACLE RDBMS. Figure 4.5 shows the system architecture of the simulated HDDS with its set of software tools.

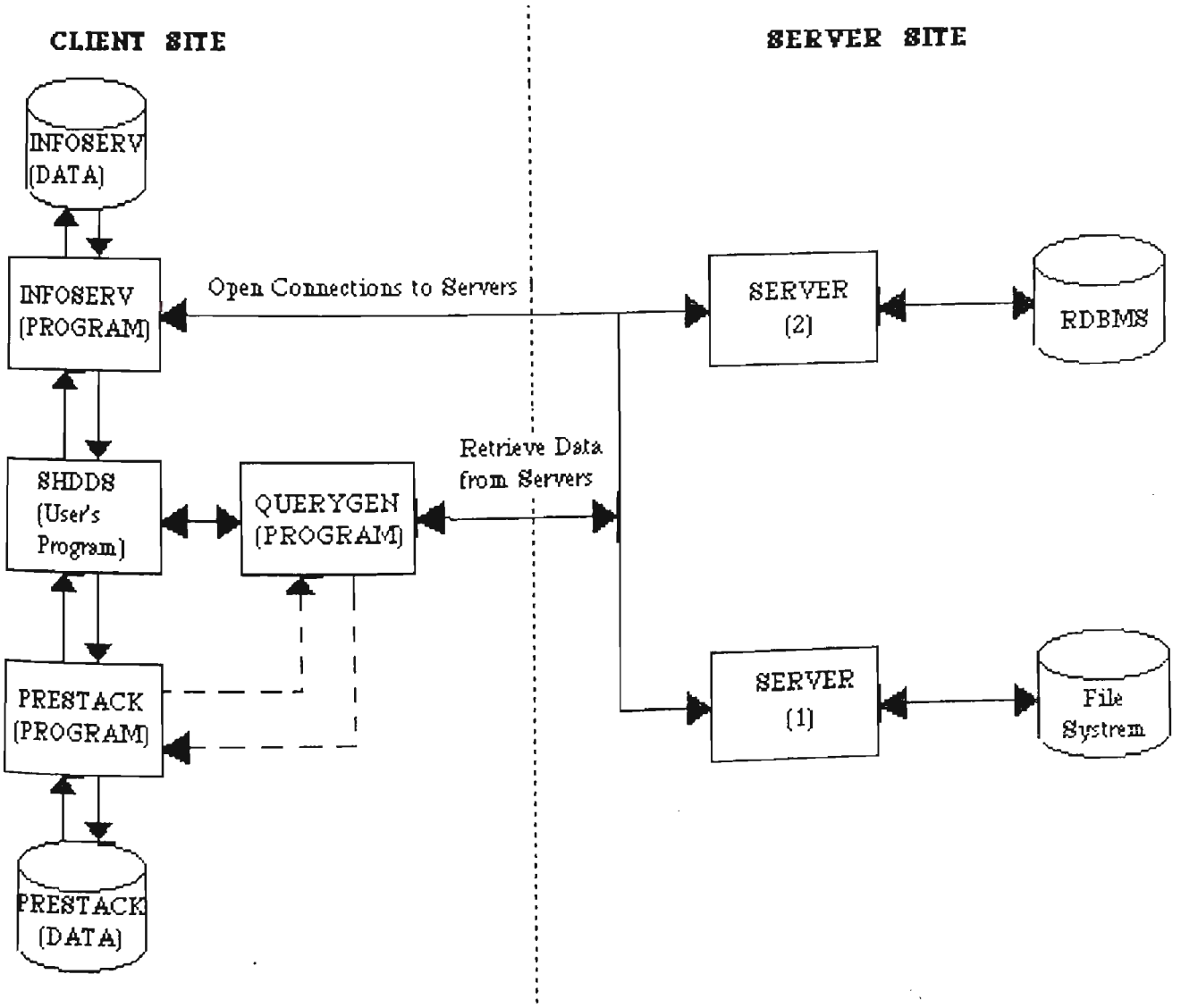


Figure 4.5 The System Design Architecture.

4.2.1 Client System and Programs

The prototype client is a Macintosh running HyperCard. Our client application is written using HyperTalk scripts which has functions to read data from both servers. The data in the RDB is accessed by Hyper*SQL statements which are embedded in HyperTalk scripts. Hyper*SQL has an Oracle driver to access the RDB. It also controls errors which may occur during the process and informs users. The HyperTalk script and Hyper*SQL treat all data as character strings, although it may contain numeric values. HyperCard can convert freely between strings and numeric fields.

There are two important modules in the client program. The first consists of three HyperCard stacks, which will perform discrete functions, which is called a CSI. Each stack is defined as a subsystem of the system software to support a client query. The second module is a user interface program, which allows users to enter a query and presents an outcome of the query.

The subsystem programs, INFOSERV, QUERYGEN, and PRESTACK are written in HyperTalk scripts. INFOSERV is a subsystem which keeps all information about available servers and services. QUERYGEN is a subsystem which has functions to generate query commands which are suitable for each server - depending on the information stored in INFOSERV. PRESTACK is a stack that stores all information retrieved from remote repositories. PRESTACK can also be used as a secondary source when a physical disconnection occurs, or the query had been re-issued while the timestamp in the PRESTACK is equal to the timestamp at a remote database. If a user believes that an incorrect data value has been presented and some approximation is needed, information from the PRESTACK may be used in the approximation routine.

A user interface program, called SHDDS uses the embedded commands of INFORSERV and QUERYGEN to request data from remote sources. The embedded commands will perform all retrieval functions, relieving the user of the need to determine the location of the data requested, which will be provided by INFOSERV and QUERYGEN subsystems.

SHDDS: User Interface Program

A user uses the SHDDS user interface program to enter a query which consists of variable names, resource name, and a condition. The resource name will be treated as a table name or file name, as appropriate, in subsequent processing. SHDDS has many standard commands to call suitable routines from INFOSERV, QUERYGEN, and PRESTACK. Standard commands reduce the difficulties in controlling transaction accesses to different locations. The data manipulation is performed by the user interface program which can use values passed from standard commands to present data, performing merges if required.

In presenting data to a user, the consistency of data from different resources must be taken into account, using the timestamp at each location [RUSINKIEWICZ et al, 1991]. If data from distinct resources have different timestamps then those data should not be merged. However, if there is a field of a database that has been updated, and this field does not occur in any related database, the merging of data from different repositories, including the updated database, can be performed if and only if the timestamps of all required records are consistent. The SHDDS will receive an indication from the CSI whether the retrieved data is consistent or not.

INFOSERV: Subsystem of Server Information

INFOSERV is the first subsystem which will attempt to satisfy a user requirement. The location of each variable requested must be validated by INFOSERV. The information provided by each server is recorded as a card in a stack each of which will be updated automatically if the timestamp of the created data schema has been changed. The functions of INFOSERV can be defined as follows:

1. checks the "interested_area" entered by the user to indicate that the given field is a database name or a file name.

2. checks the location of every variable requested, whether it is stored in the database or the file;

3. opens links between the client and servers;

4. determines if there has been a change of data schemas by comparing the timestamps stored by INFOSERV with the timestamp on each server;

5. updates and notifies the user whenever a change of service information has been found.

All the processes above can be performed using the information stored by INFOSERV, which will contain an ORACLE table name details, or a file name and details, depending on the type of server. The standard commands of the INFOSERV module which will be embedded in the SHDDS and other two subsystems are described below.

do_close_connect

This command is used to close the connection between a client and Server1.

do_disconnect_sql

Similar to the do_close_connect command, it is used to closed the connection between a client and Server2.

findLocation

The location of required variables will be found by this procedure. After all related data resources have been found by findResource, every variable in the query will be checked for its location, using those resources. The result of this search will be a table name of a database, or a file name, which contains the required variable, and the variable name at the remote site.

findResource

The findResource command defines the type of data resource: a database or a file system. It uses the table name entered by the user to determine the source. The result of this function will be the type of a resource (database or file), and the name of area of interest (table name or file name).

FLCompare

This command is used to check the timestamp of data definition of the required file, including the data schema itself, which is stored in a card of INFOSERV stack.

get_format

This command is used by a QUERYGEN routine to retrieve the data format of a file if there is a change of data format.

open_file_unix

This command is called by QUERYGEN when the connection between the client and the Server1 is established.

openDatabase

This command is called by QUERYGEN when the connection between the client and the Server2 is established. The timestamp of the data definition will be checked, and if the timestamp stored in the PRESTACK is different from the remote source, then the new data definition will be retrieved and recorded in a new card of the INFOSERV stack.

openLink

The link between a client and servers will be specified after resources are defined. This function will open connections to all servers, depending on the data resources on each server. If the data are stored in a database, then the command to open a connection will be generated. After that, the access right will be checked by the server. If the data are recorded in a file system, then the asking-for-connection command of MacTCP toolkit will be sent to the server.

sql_error

This command is used to check the process of a SQL statement, and it detects errors such as when a connection line is closed or the data schema has been altered. Some significant errors are given below.

00902 invalid datatype

00903 invalid table name

00904 invalid column name

00910 specified length too large for CHAR column

00932 inconsistent datatypes

00942 table or view does not exist

01034 ORACLE is not available

01040 This version of ORACLE does not match the mounted system

01454 cannot convert column into numeric datatype

01455 converting column overflows integer datatype

01457 converting column overflows decimal datatype

02017 integer value required

02267 column type incompatible with referenced column type

06100 connection error

06402 unexpected end-of-file

06408 bad message type

06410 network read error

06412 bad read length

06413 unexpected end-of -file.

QUERYGEN: Subsystem of Query Generator

We have assumed that there are various DBMSs and file systems, and that the data retrieval methods from distinct sources are different. QUERYGEN is a subsystem which has functions to generate query statements suitable for each server. When locations of variables have been found, QUERYGEN will use them to create query statements. For example, in our prototype, if data are stored in the database system then an SQL statement will be generated; otherwise, the MacTCP toolkit will be used to retrieve data from the server.

During the retrieval process, it is possible that a connection line between the client and a server will close. In this situation, QUERYGEN returns a flag to allow the user application program to decide on the next step. When a disconnection has occurred, and the retrieval process from every resource has not finished, QUERYGEN will keep checking and trying to reconnect to the lost server.

There are two standard commands which will be embedded into the user program to control the retrieval process from diverse resources, createCommand and retrieve_info.

createCommand

This command will generate a SQL statement to retrieve data from a database system. The variable names of data in the database are indicated by INFOSERV where the table name is defined using the area of interest in the query statement.

FLestimate

The FLestimate is called when a user wants to find an approximate data value, either in place of a result, or when the result of a query cannot be obtained.

retrieve_info

This command will send a SQL statement to a DBMS server, or call the routine using the MacTCP toolkit commands to retrieve data from a file. During the retrieval period, the status of connection lines will be monitored until the retrieval process is finished. The command returns the retrieval status. If the retrieval succeeds, the returned value will be zero. Otherwise, it will indicate the type of an error, such as a changed format, or a disconnection, and the server which generated the error.

PRESTACK: Subsystem of Preserved Data

The concepts of implementing a PRESTACK are as follows.

1. The workstation will create a PRESTACK when the system is used, for the first time, by each user.

2. Each time the user retrieves data from the original resource, PRESTACK will update by adding new data that is not already held, or creating a new card if the data schema of the original repository has been changed, or replacing the data in PRESTACK with the new value and keeping the previous value on the historical card of the same table name.

3. PRESTACK will not record incomplete information that is obtained before a physical disconnection occurred, or data obtained from an approximation.

PRESTACK is created in the form of a stack. PRESTACK will be a backup of some information from the original database and file systems, containing the results of recent user queries. PRESTACK contains records which are called *preserve stack cards (PSC)* which contain the following details:

1. the name of the database table or file from which the item was retrieved;

2. the update time of data in the original data sources;

3. the update time of a data format in the original data sources;

4. all conditions of user queries such as COUNTRY = "UK";

5. the header of retrieved data which is the list of variable names; and

6. retrieved data values.

PRESTACK will be used when a connection line is closed, in the following situations illustrated in Figure 4.6 and Figure 4.7.

1. The connection fails while retrieving the information from the RDBMS.

2. The connection fails while retrieving the format of a file, or data from a file.

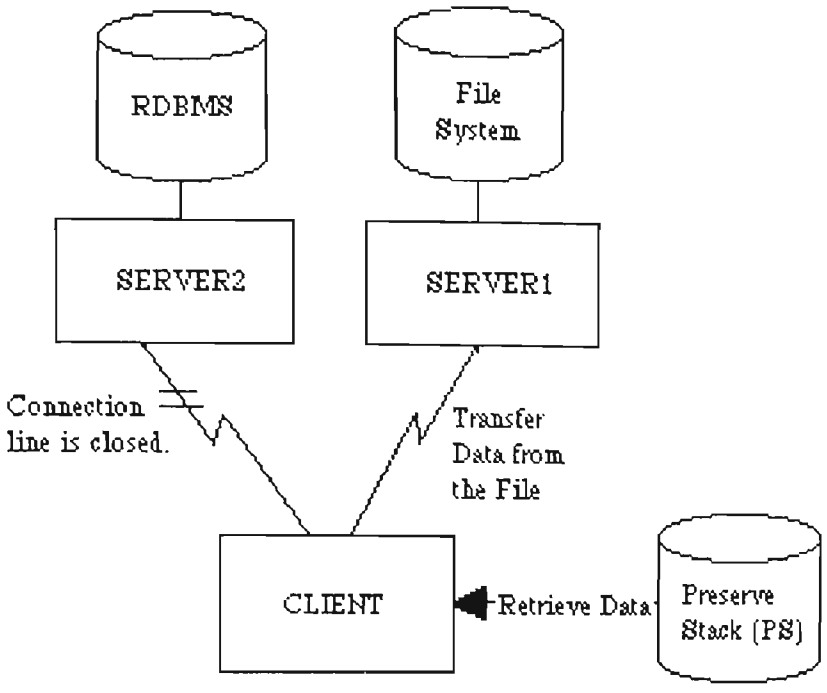


Figure 4.6 Using PRESTACK to Retrieve data When no Physical Connection between the Client and Server2.

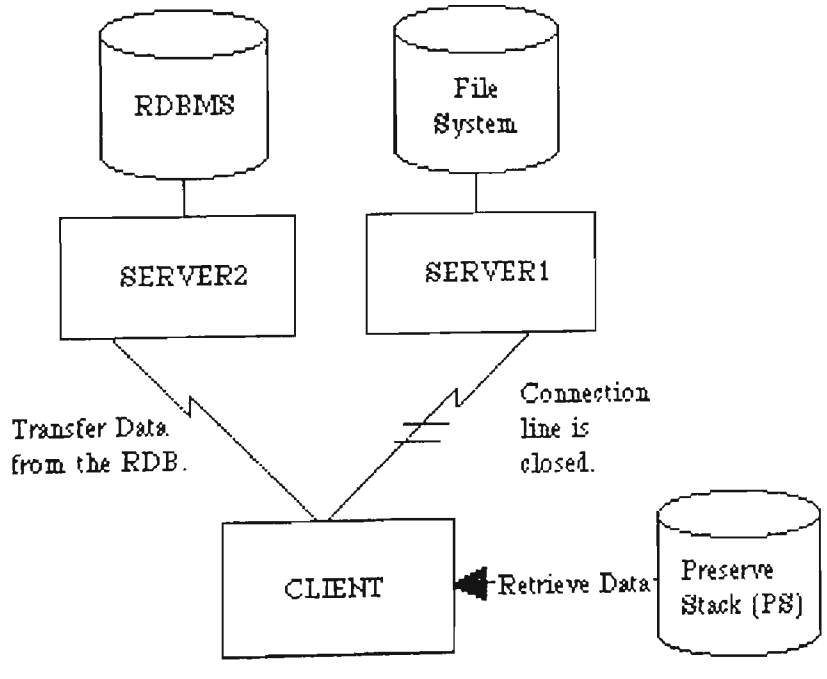


Figure 4.7 Using PRESTACK to Retrieve data When no Physical Connection between the Client and Server1.

The retrieval method on PRESTACK is as follows.

1. If the PSC contains the same query as the current user query, the data can be retrieved from PRESTACK using the table name or the file name and the field names of the current query to search for the required data.

2. If the key has never been retrieved, then stop searching PRESTACK; the client program shows data from the part of the query already processed.

Some commands of PRESTACK which relate to two subsystems and the user interface programs are as follows.

do_copy_log

This command is embedded into the SHDDS to make a copy of data retrieved from original resources in the PSC. do_copy_log has two functions: create a new card, and update a previous card.

The creation of a new card will be performed when data items retrieved from the new source have not been recorded in PRESTACK, or the timestamp which is a label of the entire table or the file has been changed. Another possibility is that the card of the database table or the file will be updated by adding new values or replacing an existing data value by one with a later timestamp. In that case, the previous data value will be saved in the historical card of the same table name in PRESTACK.

findLine

The findLine command is used to find the line containing the required name in a group of data fields which is recorded in the card.

retrieve_log and rtvAPPXDB

These routines retrieve a previous data item from a card in PRESTACK when a user needs the previous value in an approximation routine.

rtvDBMS and rtvFILE

These routines are called from the SHDDS and QUERYGEN modules to retrieve data from the PSC, rather than the DBMS or the file server respectively. These will be invoked when a connection is lost, or the required data item is already in the PSC. The latter assumption will be checked by QUERYGEN.

setUpdateLog

The setUpdateLog command is called by INFOSERV to check whether the update time of data in PRESTACK is the same as the original data.

We now consider some possibilities for approximating a value. There may be relationships between data from different repositories, as illustrated in Table 4.1 and Table 4.2.

YEAR	POPULATION_of_COUNTRY	INCREMENT_RATE
1985	25,000,000	
1986	28,000,000	8%
1987	29,400,000	5%
1988	31,458,000	7%
1989	34,603,800	10%

Table 4.1 Interpolate Population Using Time.

YEAR	NO_OF_PRODUCT	NO_OF_SALE_PRODUCT	SALE_RATE
1985	250,000	200,000	80%
1986	285,000	57,000	20%
1987	270,000	135,000	50%
1988	330,000	313,500	95%
1989	245,000	171,500	70%

Table 4.2 Choice of Interpolation of Sales Against Time or Production .

An approximate value for a numeric data item can be obtained by *interpolation.* Interpolation involves the selection of a function $p(x)$ from a given class of functions, often polynomials, in such a way that the graph of $y = p(x)$ passes through a finite set of given data points [ATKINSON,1989]. After fitting a function, it can be used to predict the value of y, given a value of x. In a table with several numerical fields, there is considerable choice in the interpolating variable and the functional form chosen.

In our prototype, the file systems[25] on Server1 are related, so some form of interpolation may well be appropriate. The example of the relationship between the two files is shown in Table 4.3.

[25] The data on the file systems are assumed to have the relationship with each other, as defined in Section B.1

Country	Year	Agriculture_Data	Alimentaires_Data
USSR	1985	109.83996	110.339996
USSR	1986	117.269997	118.660004
USSR	1987	116.139999	117.88999

Both fields depend on the climate

Table 4.3 An Example of Relationship between Fields in Two Files.

Table 4.4 gives an example of the possibility.

Country	Year	Agriculture_Data	Alimentaires_Data
UK	1985	109.760002	120.779999
UK	1986	110.099998	132.470001
UK	1987	108.919998	%$##%$^#@$

X and Y values which are used to create a function.

X value which is used to approximate the Y value, X_i

Y value which is needed to be approximated, Yi.

Table 4.4 The Values used to Create the Functional Relation between Two Files.

There are many methods of interpolation such as Polynomial Interpolation, Newton Divided Differences, Lagrange Polynomial, and Hermite Interpolation. The method used in this thesis is Newton Divided Differences[26].

Linear interpolation yields

$$y_i = p(x_i) = f(x_0) + \frac{f(x_1) - f(x_0)}{(x_1 - x_0)} * (x_i - x_0)$$

when the x_i : value used to approximate y_i,

x_0, x_1 : values which are in the same years as y_0 and y_1,

$f(x_0), f(x_1)$: values of the production from the same file as y_i, matched with the year of x_0 and x_1 .

If an approximation is required for non-numeric data, or numeric data for which no rationale exists for interpolation, the data in PRESTACK will be presented to the user with its timestamp. The data in PRESTACK will be searched by country name and the field name. If there is a data item for that country name and field name available then it will be presented to the user as an approximate value as illustrated in Table 4.5.

26 The details of this method can be found in any text books of the numerical analysis methods.

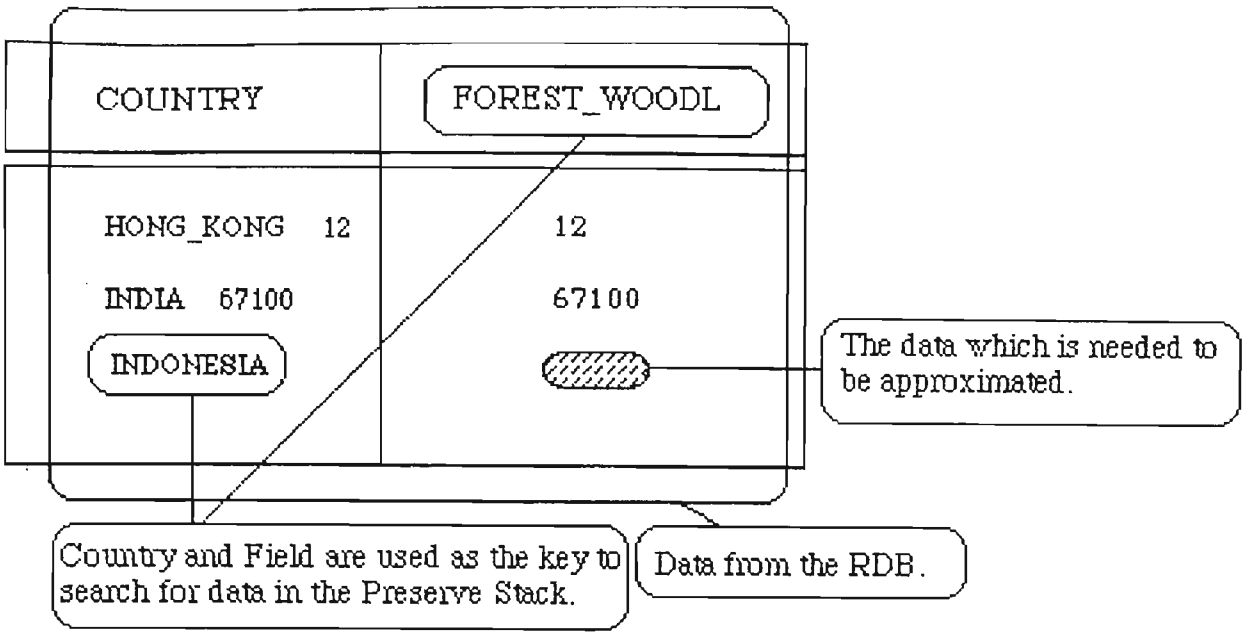


Table 4.5 An Example of Finding the Key to Retrieve an Approximate Data from the RDBMS.

4.2.2 Server Systems

There are two servers which store related data. The first server uses a file system, and the second server stores data in a RDB. When the connection is closed between Server2 and a client, the RDBMS will perform recovery on the RDB. In such case the client application need not consider the effect of disconnection on the RDB. When a user retrieves data from each server, each server has to control all security functions. To achieve data security, the following functions are required of each server.

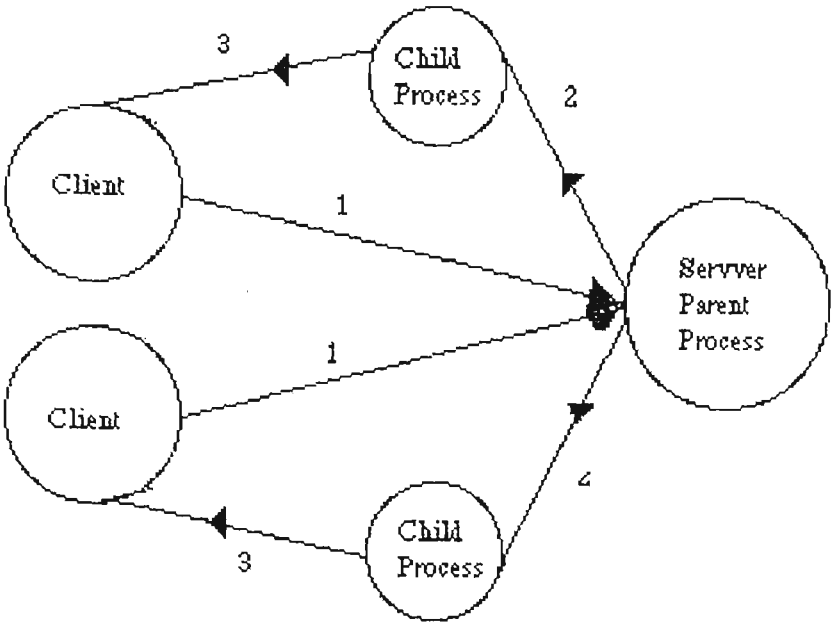
1. The server must register itself to INFOSERV on each client system.

2. The server's address, data source names and types (the type of data resource, such as an RDBMS or a file system), and location of data definitions, either of the whole database, or the user view the DBA is willing to grant, must be provided to INFOSERV when the server registers.

3. Whenever the location of a data definition, the server address, or the type of data source, is changed the system administrators of INFOSERV clients must be notified to update their information and QUERYGEN routines.

If the servers adhere to these requirements completely, the client programs will be able service all user requirements. The information on servers in INFOSERV can be updated automatically, once the location of data definition has been communicated.

After the client issues a query statement, the RDBMS on Server2 will manipulate the retrieval process and send the result or an error code back to the client. If disconnection occurs, the client program can continue with reduced functionality. To cope with the problem of resynchronising messages between the client and Server1, after a disconnection has occurred, the Server1 software uses the parent/child model. The UNIX system call *fork* allows a process to split into two duplicated processes. The main process is called a *parent* process and the duplicated process is called a *child* process. Whenever a client asks for a connection, the parent on Server1 will generate a child process and passes the client connect off to the child process. After a client finishes its task and terminates the connection, the child process will be killed. Using this method, sending and receiving commands on both systems will be in the correct sequence. The connection between a client and the Server2 is illustrated in Figure 4.8.



1 Client asks for the connection.
2 Server splits the process to be parent and child processes.
3 Child responds to the client.

Figure 4.8 Connections between a Client and Server2.

4.2.3 Protocol

The protocol which is used to transmit data between a client and servers in this environment is TCP/IP. The transmitted data is guaranteed by the protocol to be correct, in order, and not duplicated. We chose TCP/IP because it is the most commonly used protocol, and is available for our prototype on UNIX and Apple Macs.

4.3 Evaluation

Our design uses standard commands embedded into a user application program, SHDDS, allowing the retrieval process to be managed by INFOSERV, QUERYGEN, and

PRESTACK without interference from users. Aspects of the design can be summarised as follows.

1. Ease of User Use

Because the application program uses embedded commands to obtain data values from the remote servers, it need not manage any retrieval processes or supply different retrieval commands for different servers. Users need not be concerned with the various data models that exist in the HDDS. Facilities are provided for a user to retrieve information about the servers, and to check the data definitions on a server before issuing a query. Users are able to decide their own presentation format for the retrieved data .

2. Monitoring changes on the server

The design includes the facility to check for changes on the servers and keep users aware of the current state of information distributed across the servers.

3. Presenting Consistent Information.

Consistency of data, based on timestamps, will be checked. Whenever the timestamp values from different places are different, the CSI will determine these data to be inconsistent, and present each separately to users. The CSI can present some information to users even if there is a communication failure, by keeping a copy of previously obtained data in the PRESTACK. If asked by the user, it may be able to estimate data from other values it holds, using some approximation.

4. Low Cost Access

By storing the access costs of each server, the system is able to choose the lowest access cost server for retrieving data. Some of this data may be retrieved from the local database, in which the results of previous retrievals are recorded.

5. Maintenance of information on servers

The information stored by INFOSERV and PRESTACK will be updated by each subsystem automatically. However, when a new server is ready to contribute data, its information has to be registered in INFOSERV by the server administrator. Thus maintenance of the system will be a joint responsibility of the server and client system administrators.

1. *Adding a new server, or new services.*

When a new server system is added, the server administrator has to register all details of its services to allow INFOSERV to show users the new data retrieval possibilities.

2. *Adding a DML for a new database.*

When a new database is added, its DML must be considered. If the new database employs a DML already available, no alteration in QUERYGEN is required. Otherwise, the new DML has to be added into QUERYGEN. Adding a new database system may affect the data-model transformer of the PRESTACK. To store data from the new data model into the PRESTACK, the new data model has to be transformed to a local data model that would be available in the PRESTACK.

6. Security Control

The first check of user access authority is a local check performed at the client site. The second check is performed at server sites. These processes (two-step checking) guarantee that the user is an authorised user who can access both PRESTACK and remote databases.

It is possible that during the linking process between the client and a server, a communication failure occurs. In order to serve the user during the communication failure, the retrieval process will be performed on the PRESTACK. Thus, the access check at the client can prevent an unauthorised user from obtaining data during the communication failure in spite of the fact that remote checking cannot be done.

7. Speed

We used interpreted HyperTalk to implement the CSI propotype, so the speed of this program is slow compared with a program running objectcode. However, later versions of Hypertalk can be compiled, which should improve execution speed for HyperTalk scripts.

4.4 Summary

Our prototype is a simulated HDDS containing one client and two servers, Server1 providing data from files, and Server2 providing data from the ORACLE RDBMS. The connection protocol is TCP/IP.

Our prototype system software consists of three subsystems. The first subsystem, INFOSERV, verifies the required variables, establishes connections to servers, automatically checking and updating of information of available services. INFOSERV reports changes of

data schema at a remote server to users as they are detected. Once a user issues a query; all required variables will be verified; the locations of servers which provide such value are determined; and links to servers established.

The second subsystem QUERYGEN, which controls all retrieval routines, is activated when the location of each server has been established from INFOSERV. The functions of QUERYGEN are separated into three parts. The first is to generate a suitable statement suite for each required server, relieving users of knowing details of DMLs to access different servers. The second part is to retrieve data from available servers, attempting to continue even if a physical disconnection occurs. The last function of the QUERYGEN module is to determine the consistency of data received from various repositories by comparing the timestamp of each record, either explicit, or implicit from the timestamp labels of tables and files.

PRESTACK is the last subsystem which serves users while a disconnection continues; it is also used as a local data source to provides some current data to users more cheaply than by remote access.

Chapter 5

Conclusions and Further Study

This chapter will present conclusions concerning the system software design discussed in Chapter 3, and the implementation described in Chapter 4. We describe what we have achieved, and some possible further extensions to the design and implementation.

5.1 Implications of the Demonstration

In this thesis the timestamp mechanism is proposed for solving the problems of a change at a server system. The timestamp is also a heuristic AI technique to detect the inconsistency of data when a local database is implemented.

1. The timestamp can be used to identify the change of a data schema at a server site, or to determine a change of data value at a server. A user is informed if a timestamp inconsistency of data in the HDDS is detected.

2. Our prototype system contains three parts: a client system two servers, and the TCP/IP protocol. An algorithm is also implemented to hide the differences of various data manipulation languages.

3. Our design includes a local database at the client site, which gives the possibility on continuation in a disconnection situation. The system software is able to present some possible data to the user during the disconnection period. Information about available services can be given to users when needed.

5.2 Further Study

Our prototype system has been implemented in HyperCard. To complete this design in real applications, further work would be required in the areas of compiler, protocol, and data models of databases as follows.

5.2.1 Compiler

In order to use standard commands embedded into user applications, a technique of compiling and linking between the user software and the CSI has to be developed. This technique will depend on the language which is used to implement the system software. Thus, the consideration of finding a suitable language to implement the software has to take place before finding the technique of compiling and linking between users' application and software. However, the compiler for a natural language is recommended.

5.2.2 Protocol

In order to connect and transmit data between a client and servers, some efficient protocol is required. Although this thesis uses TCP/IP, there is a possible that other protocols might be more suitable than it. This is so because the TCP/IP is fairly low level (may be difficult to handle) and less flexible from the user's point of view. However, the protocol

which is used in this software is based on the assumption that it can run over various types of computer systems and maintain efficiency in transferring data.

5.2.3 Data Model of Database Systems

The data model of each ISDB and PDB should be taken into account for the real world applications because the objectives and concepts of a design to overcome problems in each organisation are different. For example, the objective to develop an efficiency hypertext or multimedia software.

5.3 Summary

The research reported in this thesis addresses the retrieval of information from independent sources of data, potentially on a global basis, over an extended period of time. The objective was to give the individual user, the researcher or retriever of the information, a set of mechanisms for continuing their work in this environment with the certainty that the data retrieved maintained its consistency.

This problem is one that has not been addressed by current wide area retrieval systems as is shown in Chapter 2.

A series of solutions and their corresponding mechanisms have been devised during the research to handle a wide variety of problems as they became apparent. These mechanisms handle communication link failure, changes in data values, changes in database schema and the augmentation of file systems to permit any form of consistency checking.

A major result has been the recognition that a source of information can be made available through the local storage of previously accessed and retrieved results. This generation

of a working local environment and its usage has been demonstrated to be theoretically achievable.

The mechanisms have all been implemented on a representative test environment to check their viability and correctness.

The overall result has been that the original objectives of the research have been achieved, and significant innovation demonstrated in the area of HDDS.

Bibliography

[AHMED et al.,1991] Ahmed, R., et al., "The Pegasus Heterogeneous Multidatabase System," IEEE Computer, Vol.24, No.12, December 1991, pp 19-27.

[APPLE,1988] Macintosh HyperCard user's guide, CA: Apple Computer, 1988.

[APPLE,1989] HyperTalk(TM) Beginner's Guide: An Introduction to Scripting, CA: Apple Computer, 1989.

[APPLE-MACTCP,1988] "Apple-MacTCP Programmer's Guide: Networking and communications Publications," Apple Computer, 1988.

[APM(1),1991] An Application Programmer's Introduction to the Architecture, Cambridge: Architecture Projects Management Limited, Release TR.017.00, Nov. 1991.

[APM(2),1991] ANSAware 3.0 Implementation Manual, Document RM.097.01, Cambridge: Architecture Projects Management Limited, February 1991.

[APM(3),1991] The System Designer's Introduction to the Architecture, Cambridge: Architecture Projects Management Limited, April 1991.

[ANDERSON & LEE,1981] Anderson,T., Lee, P.A., Fault Tolerance Principles and Practices, Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[ARONSON,1994] Aronson,L., HTML: Manual of style, 1st.ed., California:Ziff-Davis Press.

[ATKINSON,1989] Atkenson, K. E., An Introduction to Numerical Analysis, 2nd.ed., New York: John Wiley & Sons, 1989.

[BARGHOUTI & KAISER,1991] Barghouti, N.S., Kaiser, G.E., "Concurrency Control in Advanced Database Applications," ACM Computing Surveys, Vol.23, NO.3, September 1991, pp 269-317.

[BATINI et al.,1986] Batini, C., Lenzerini, M., Navathe, S.B., "A comparative Analysis of Methodologies for Database Schema Integration," ACM Computing Surveys, Vol.18, NO.4, December 1986, pp 324-364.

[BEAUCHAMP,1987] Beauchamp, K.G.,Computer Communications, Berkshire:Van Nostrand Reinhold (UK) Co.Ltd.,1987

[BERNERS-LEE et al.,1994] Berners-Lee,T., et al., "A Secret. The World Wide Web," Communication of ACM, Vol.37, No.8, pp. 76-82, 1994.

[BERSHAD & LEVY,1988] Bershad, B.N., Levy, H.M., "A Remote Computation Facility for a Heterogeneous Environment," IEEE Computer, Vol.21, No.5, May 1988, pp. 50-54.

[BOBROWSKI,1992] Bobrowski, S., ORACLE(R) RDBMS: Database Administrator's Guide,Vol.2.,Version 7.0: Developer's Release Documentation, USA:ORACLE Corporation, May 1992.

[BREITBART,1990] Breitbart, Y., "Multidatabase Interoprability," ACM SIGMOD record, Vol.19, No.3, Sept.1990, pp 53-60.

[BREITBART et al.,1992] Breitbart, Y., et al., "Overview of Multidatabase Transaction Management," The VLDB Journal, Vol.1, No.2, Oct.1992, pp.181-240.

[CERI & PELAGATTI,1985] Ceri, S., Pelagatti, G., Distributed database, principles & systems, McGraw-Hill International, 1985.

[CHEU,1988] Cheu, D., SQL Language: Reference Manual ORACLE RDBMS Version 6.0, USA:ORACLE Corporation, November 1988.

[CHUNG,1990] Chung, Chin-Wan, "DATAPLEX: An Access to Heterogeneous Distributed Database," Communications of the ACM, Vol.33, No.1, January 1990, pp. 70-80.

[CLIFTON,1987] Clifton, Carl S., What every engineer should know about data communications, Marcel Debber, Inc., New York and Basel, 1987, pp 61.

[COMER,1988] Comer, D., Internetworking with TCP/IP, Principles Protocols and Architecture, Prentice Hall, 1988.

[DAYAL&HWANG,1984] Dayal, U., Hwang, H.-Y., "View Definition and Generalization for Database Integration in a Multidatabase System" in Hurson, A.R., Bright, M.W., Pakzad, S.H. (Ed.): "Multidatabase Systems: An Advanced Solution for Global Information Sharing," Washington:IEEE Computer Society Press, 1994.

[DAVIDSON,1988] Davidson, J., An introduction to TCP/IP, Springer-Verlag, 1988.

[DEEN et al.,1987] Deen, S.M., Amin, R.R.,Taylor, M.C., "Data Integration in Distributed Databases," IEEE trans. on Software Eng., Vol.SE-13, No.7, July 1987, pp 806-864.

[DEMURJIAN&HSIAO,1988] Demurjian, S.A., Hsiao, D.K., "The multi-lingual database system," Proc. 3rd. Int.Conf. Data Engineering, Los Angeles, CA, Feb. 1987.

[DREW et al.,1992] Drew, P., et al., "A Toolkit for the Incremental Implementation of Heterogeneous Database management Systems," The VLDB Journal, Vol.1, No.2, October 1992, pp 241-284.

[FAO(1),1986] "FAO Production Yearbook," Rome: Food and Agriculture Organization of the United Nations, Vol.40, 1986, pp 9-13.

[FAO(2),1987] "FAO Production Yearbook," Rome: Food and Agriculture Organization of the United Nations, Vol.41, 1987, pp 53-57.

[FAO(3),1988] "FAO Production Yearbook," Rome: Food and Agriculture Organization of the United Nations, Vol.42, 1988, pp 53-57, 84-92.

[GARCIA,1982] Garcia-Molina, J., "Reliability Issues for fully Replicated Distributed Databases," IEEE Computer, Vol. 16, No. 9, September 1982, pp 34-42.

[GOLDING,1992] Golding, R.A., "A Weak-Consistency Architecture for Distributed Information Services", Computer Systems, Vol. 5, No. 9, Fall 1992, pp 379-405.

[GRAY,1981] Gray, J.N., "The Transaction Concept: Virtue and Limitations," VLDB, September 1981,pp 144-154.

[GROFF & WEINBERG,1990] Groff, J.R., Weinberg, P.N., Using SQL, Berkeley: McGraw-Hill, 1990.

[HAMMER&McLEOD,1980] Hammer, M.,McLeod, D., "On Database Management System Architecture," in Hurson, A.R., Bright, M.W., Pakzad, S.H. (Ed.): "Multidatabase Systems: An Advanced Solution for Global Information Sharing," Washington:IEEE Computer Society Press, 1994.

[HSIAO,1991] Hsiao, David K., "Database and Database Systems in the 21st Century" in Mesirov, J.P.(Ed.):"Very Large Scale Computation in the 21st. Century," Society for Industrial and Applied Mathematics(SIAM), Vermont:Capital City Press, 1991.

[HSIAO & KAMEL, 1989] Hsiao, David K., Kamel, Magdi N., "Heterogeneous Database: Proferations, Issues, and Solutions," Washington:IEEE Trans. on Knowledge and Data Eng., Vol.1, No.1, March 1989, pp. 45-62.

[JABLONSKI et al.,1990] Jablonski, S., et al., "Implementation of a Distributed Data Management System for Technical Applications - A Feasibility Study," Information Systems, Vol.15, No.2, pp 247-256.

[KAHLE,1989] Kahle, B., "Wide Area Information Server Concepts," TMC Tec Report TMC202, Version 4.0, November 1989.

[KAHLE(1),1991] Kahle, B., "An Information System for Corporate Users: Wide Area Information Servers," TMC Tec Report TMC199, Version 3, April 1991.

[KAHLE(2),1991] Kahle, B., "Release 1.0," Ester Dyson, April 1991.

[KAHLE(3),1991] Kahle, B., "Roles of Electronic Publishing on Campus," TMC, Version 0.2, September 1991.

[KAHLE(4),1991] Kahle, B., "Overview of Wide Area Information Servers," Apr. 1991.

[KAHLE et al.,1992] Kahle, B., et al.,"Interfaces for Wide Area Information Servers," TMC, Apple Computer, NSF Network Service Center, Version 0.8, January 1992.

[KIM et al.,1990] Kim, J., et al., "Design and Implementation of a Temporal Query Language with Abstract Time," Information System, Vol.15,No.3, pp.349-357.

[KLAHOLD et al.,1986] Klahold, P., et al.,"A general model for version management in databases," Proc.12th Int.Conf. on Very Large Databases, pp. 319-327.

[LEFFLER et al.,1989] Leffler, S. J., et al., The design and implementation of the 4.3 BSD UNIX Operating system., Reading, Mass.: Addison-Wesley, 1989.

[LINDEN,1992] Linden, B., SQL Language Reference Manual Version 7.0: Developer's Release Documentation, CA.:ORACLE Corporation, May 1992.

[LISKOV & SCHEIFLER,1982] Liskov, B., Scheifler, R. "Guardians and Actions : Linguistic Support for Robust, Distributed Programs," Proc. of the Ninth Symposium on Principles of Programming Languages, January 1982, pp 7-19.

[LITWIN & ABDELLATIF,1986] Litwin,W., Abdellatif, A., "Multidatabase Interoperability," IEEE Computer, Vol.19. No.12, December 1986, pp 10-18.

[MAIER,1983] Maier, D., The Theory of Relational Databases, London:Pitman Publishing Limited, 1983.

[MARTIN et al.,1991] Martin, B.E., et al., "An Object-Based Taxonomy for Distributed Computing Systems," IEEE Computer, Vol.24, No.8, August 1991, pp 17-27.

[MARZULLO et al.,1991] Marzullo, K., et al., "Tools for Distributed Application Management," IEEE Computer, Vol.24, No.8, August 1991, pp 42-51.

[MOSS,1982] Moss, J.E.B., "Nested Transactions and Reliable Distributed Computing", Proc. of the second Symposium on Reliability in Distributed Software and Database Systems, July 1982.

[NICKOLAS,1992] Nickolas, P., Private Communication.

[NOTKIN et al.,1987] Notkin, D., et al., "Heterogeneous Computing Environments: report on the ACM SIGOPS Workshop on Accommodating Heterogeneity*," Communications of the ACM, Vol.30, No.2, February 1987, pp. 132-140.

[NISO,1988] National Information Standards Organization, Z39.50-1988 Information retrieval Service Definition and Protocol Specifications for Library applications, New Brunswick: Transaction Publishers, January 1988.

[OC(1),1988] Oracle for Macintosh: Networking, Version 1.0, Belmont: Oracle Corporation, 1988.

[OC(2),1988] SQL*NET TCP/IP: User Guide, Version 1.0, Belmont: Oracle Corporation,1988.

[PAPAZOGLOU,1991] Papazoglou, M.P., "Framework for Interconnecting Distributed Information Systems," DBIS'91: Database and Information Systems Conference, Sydney, February 1991.

[RAM,1991] Ram, S., "Heterogeneous Distributed Database Systems," IEEE Computer, Vol.24, No.12, December 1991, pp 7-10.

[REDDY et al.,1988] Reddy,M.P., Prasad, B.E., Reddy, P.G., "Query Processing in Heterogeneous Distributed Database Management Systems," in Gupta, A. (Ed.): "Integration of Information Systems :Bridging Heterogeneous Databases," New York: IEEE Computer Sociaty Press, 1989.

[RUSINKIEWICZ et al,1991] Rusinkiewicz, M., Sheth, A., Karabatis, G., "Specifying Interdatabase Dependencies in a Multidatabase Environment," IEEE Computer, Vol.24, No.12, December 1991, pp 46-53.

[SCHLICHTER & MILLER,1988] Schlichter, J.H., Miller, L.J., " FolioPub: A Publication Management System," IEEE Computer, Vol.21, No.1, Jan. 1988, pp.61-69.

[SHETH & LARSON,1990] Sheth, A.P., Larson, J.A., "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases," ACM Computing Surveys, Vol.22, No.3, September 1990, pp 183-235.

[SIEWIOREK & SCHWARZ,1982] Siewiorek, D., Schwarz, R., The Theory and Practice of Reliable System design, Digital Press, 1982.

[SINGHAL & CASAVANT,1991] Singhal, M.,Casavant, T.L., "Distributed Computing Systems," IEEE Computer, Vol.24, No.8, August 1991, pp 12-15.

[SKEEN & STONEBRAKER,1983] Skeen, D. ,Stonebraker, M., "A Formal Model of Crash Recovery in a distributed System," IEEE Transactions on Software Engineering, Vol.SE-9, No.3, May 1983.

[SMITH & OMAN,1990] Smith, D.B., Oman, P.W., "Software Tools In Context," IEEE software, May 1990, pg. 15-19.

[STANKOVIC,1985] Stankovic, J., "A Reliable Distributed System Software," IEEE computer society order No.570, IEEE Catalog No.EHO230-3, 1985, pg 3-5.

[STALLINGS,1991] Stallings, W., Data and Computer Communications, 3rd ed., New York: Maxwell Macmillan International Editions, 1991.

[STEIN,1991] Stein, R.M.,"Browsing Through Terabytes," BYTE, May 1991, pp 157-164.

[STROM & YEMINI,1984] Strom, R.E., Yemini, Shaula, "Optimistic Recovery : An Asynchronous Approach To Fault-Tolerance in Distributed Systems," Proc. of the Fourteenth International Conference on Fault-Tolerant Computing, June 1984.

[SUN,1988] Network Programming, Australia: SUN Microsystem, Revision A, 1988.

[SVOBODOVA,1984] Svodobova, L., "File Servers for Network-Based Distributed Systems," Computing Surveys, Vol.16, No.4, December 1984, pp 353-398.

[TAKAGI,1979] Takagi, A., "Concurrent and Reliable Update of Distributed Databases," Massachusetts Institute of Technology, November 1979.

[THOMAS et al.,1990] Thomas, G., et al., "Heterogeneous Distributed Database Systems For Production Use," ACM Computing Surveys, Vol.22, No.3, September 1990, pp. 246-249.

[TOMLINSON,1991] Tomlinson, B., "An oldesh (release 1.1 vintage): Introduction to CORBA," www.acl.lanl.gov/sunrise/DistComp/Objects/CORBAtalk/CORBAtalk.html.

[VERHOFSTAD,1978] Verhofstad, J.S.M., "Recovery Techniques for Database Systems," Computing Surveys, Vol.10, No.2, June 1978, pp 167-195.

[WIEDERHOLD et al.,1992] Wiederhold, G., et al., "Toward Mega Programming," Communication of the ACM, Vol.35, No.11, November 1992, pg. 89-98.

[WINKLER & KAMINS,1990] Winkler, D., Kamins, S., HYPERTALK 2.0: THE BOOK, New York: Bantam Books, 1990.

[WITTIE & VAN TILBORG,1980] Wittie, L., Van Tilborg, A. M., "MICROS, A Distributed Operating System for Micronet, A Reconfigurable Network Computer," IEEE Transactions on Computers, Vol.C-29, No.12, December 1980.

Appendix A

A Presented Paper in

The International Conference on Information Systems and Management of Data,

New Delhi, October 6-8, 1993

Presentation of Consistent Information from Independent Databases

Pattarasinee Bhattarakosol

Dept. of Mathematics

Faculty of Science

Chulalongkorn University

Payathai Rd. Bangkok 10330

Thailand

fscipat@chulkn.chula.ac.th

Abstract

A Heterogeneous Distributed Database System (HDDS) is a system that contains various kinds of database systems (DBS) distributed over a network; each of which retain their autonomy when participating in the HDDS. The problem domain of this paper is based on an uncontrolled environment in which a number of conditions hold. Firstly, users do not have authority to control any changes, and the communication within the network is unstable. Additionally, no duplicated databases exist, data values are based on time, data relationships are visible only from the users' point of view, and a system's repositories include file systems.

The problem of ensuring users that presented information is consistent is discussed and a method is searched for that treats file systems as sharable sources. Furthermore, the system software should be able to provide users with up-to-date service information. It should also provide some consistency information when communication fails, and a mechanism to approximate data when needed. The available software packages such as DATAPLEX, WAIS, and ANSA do not provide facilities in supporting all requirements under the above defined circumstances. Besides, these system do not include file systems as sharable repositories. Therefore, the design of Computer Software Interface (CSI) has been proposed. The idea of using either data update time or data valid time to determine data consistency has been implemented. The concept of upgrading characteristics of file systems to be similar to the DBS is used; thus data in file systems can be shared. Registering addresses of service definitions enables the CSI to check and maintain service details kept in a local database. Consequently, the CSI can inform users when service details have been changed. A combination of logging mechanism and the implementation of a duplicated database is developed to serve users in case of communication failure occurs. Using all the discussed methods, the CSI is able to provide and guarantee consistency of data to users in the aforementioned environment.

Keywords: Heterogeneous Distributed Database Systems, Data Consistency, Independent Databases, Communication Failure.

Introduction

Currently, there are many system software implemented for the HDDS. Most of the system software provide full rights for users in the system to be able to access data such as update data, delete data, add new data, or read data. Software such as DATAPLEX [CHUNG,1990] , Pegasus [AHMED et al.,1991], manage all types of transactions such as retrieve, update etc. Some software such as the Wide Area Information Servers (WAIS) [KAHLE,1989] provides only the retrieval transaction which is entered by a user using a WAIS user interface [KAHLE et al.,1992]. The other kind of system software is like ANSA [APM(a),1991]. It provides commands that are embedded into the user's application and interact with the data holder system to gain access. The user need not deal with all difficulties in handling different access methods. These software are running under the assumption that the system is stable, and there is no error between transmitted data, or between transaction processes.

One operation that is performed by users is the retrieval operation. The retrieval process aims to retrieve relevant information for users. The information which is considered in this research are related to a certain period. The retrieval process that retrieves data based on a certain period refers as a *sustained task.* For example, information about the amount of production from a handicraft factory within the year 1989, or information about Agriculture production of Indonesia in the year 1990.

The environment of software approach in this research is based on the assumption that the communication within the network is unstable and all data repositories belong to different owners. Therefore, end users have no priority to control data in the HDDS. Furthermore, there is no duplicated database in the system. This research also focusses on a HDDS that

contains related data which are separated and stored in different data repositories (databases or files); the assembly of consistent data has to be presented to users.

As mentioned previously, data are distributed over the HDDS and the assembly of data will be presented to users. This does not imply that data are stored in a specific format in order to serve the grouping of data based on users' requirements. The grouping of data, based on the users' view is grouped from independent data formats and independent owners' objectives controlled by distinct owners. Thus, data in the HDDS are related with each other by the users' point of view whereas they are independent from an owners' point of view. As a consequence of relationships between data in the users' view, performing a sustained task will enable users to assemble their environment.

For example, a user in the Department of Agriculture of a company in JAPAN performs a sustained task to obtain Agricultural information from the Agricultural Department in USA, CANADA, UK, and FRANCE. The result from the process presented to the user in JAPAN is a composite table of each country based on the same year, as shown in Figure 1. The only person with specific view of assembled data is the user, in this example is the user in JAPAN. However, the real outcome of the sustained task is a yearly report data from each country.

According to uncontrolled environment, users must be able to trust all received data from the HDDS. Nevertheless, the mentioned system software cannot guarantee the correctness and consistency of data under the defined circumstances. Consequently, the user cannot be ensured be the system software that obtained data are consistent and correct as they must be. This is a central problem faced by an user of a HDDS software system.

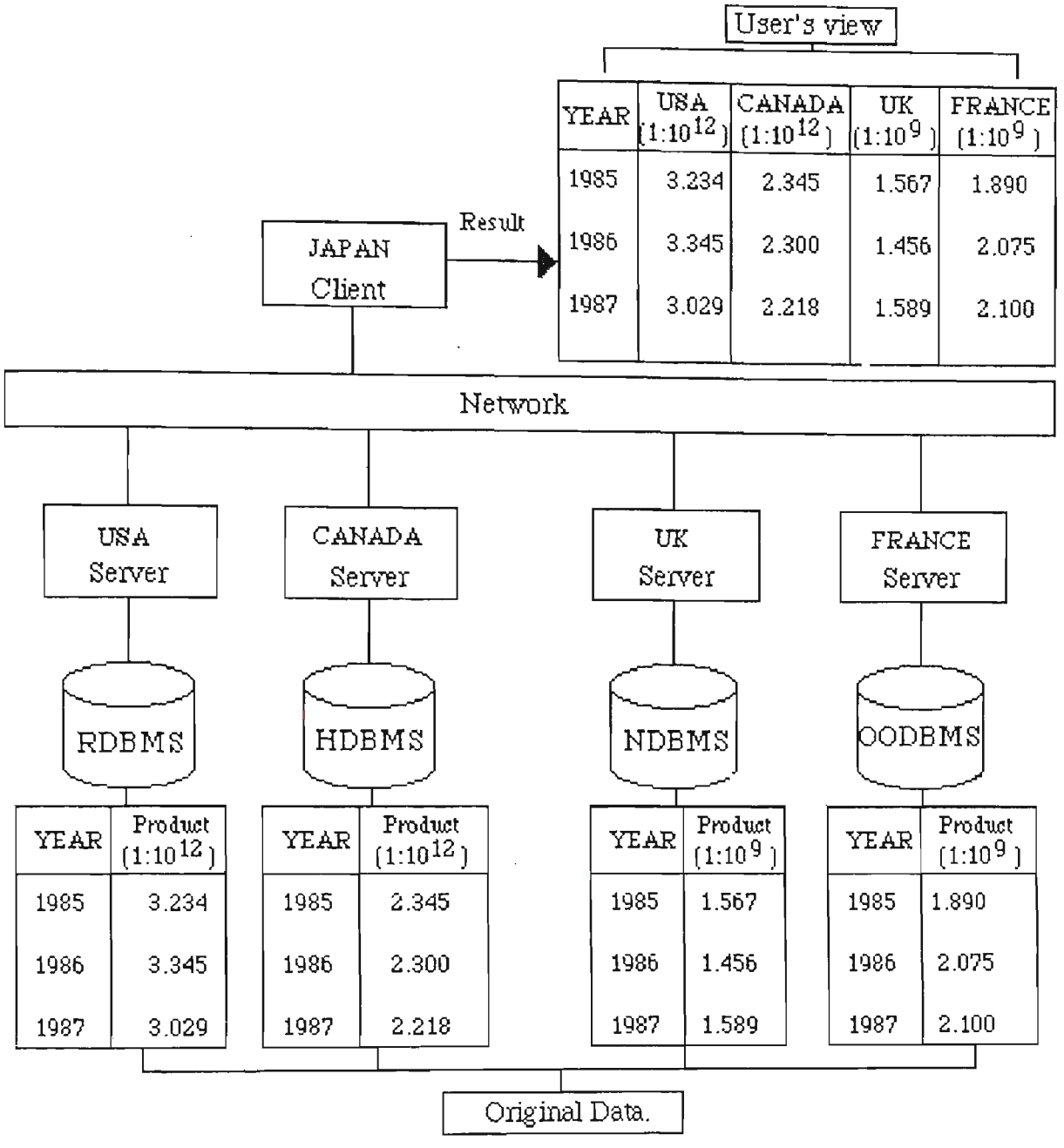


Figure 1 An Example of Sustained Task and the Result.

This paper proposes mechanisms to be implemented in the system software to control and serve users in obtaining data under the above defined circumstances; the software design will project on the retrieval process of users' queries. Additionally, the designed software should be able to inform users whenever a change of available services arises. So, users could have access to up-to-date services and information. Moreover, the objective to use

file systems as repositories in the HDDS has been considered. The method guarantees the consistency of data has been implemented. Therefore, users in the HDDS are able to be ensured by the system software that the received data is consistent.

System Environment:

This research focusses on the environment wherein remote users are working under the uncontrolled environment. This is due to the unstable communication between the client and servers. Furthermore, users cannot control any changes which are performed by database/file owners in the HDDS. However, data in the HDDS are related with each other under some users' constraints.

As an assumption that users will works under and uncontrolled environment of the HDDS, the configuration of a component node in a HDDS may constitute various kinds of hardware, data sources and protocols. Access to data stored in the HDDS may be accomplished by many applications. It is possible that there are multiple users who wish to access data simultaneously with different objectives. The data manipulation on data sources can be accomplished any time by an owner or a system administrator.

Typically, there are many distinct database systems which are installed to serve particular requirements. Each individual system in the HDDS is characterised by a specific Data Manipulation Language (DML). This prevents the use of the DML of any individual data sources as a global DML. Furthermore, each data source will have a data schema that pertains to local data only. The above mentioned factors necessitate customised data access methods for individual database systems.

Apart from a database system, a file system does not maintain any semantics about the stored data and does not have a DML. There are three primitive file structures: a sequential

file, an index sequential file, and a random file. Each structure has a unique method to access data in a file. However, many applications in various computer languages are able to access the same file. The programmers have to know the file structure and the current data definition before writing any application that uses the file. The problem manifests itself whenever a change of data definition of the file occurs without changing the previous application program.

The last component that has to be considered in a HDDS is the communication protocol. As a result of having various kinds of computer systems in a network, it is possible that the protocol on each system be different. There are many protocols to transmit data such as TCP/IP [DAVIDSON,1988], Z39.50, WAIS protocol etc. Each such protocol is implemented under a particular objective, such as, Z39.50 is implemented for retrieving data and transforming information between two computers [NISO,1988]. TCP/IP is a protocol that transmits data in predefined format and guarantees correctness of transferred data.

Under the circumstances that there are various kinds of data repositories which may or may not contain the data schema, and various kinds of protocols exist in the HDDS, there are many problems to be considered before designing and implementing a system software which manages presentation of data to users. The most important aspect of presenting data to users relates to consistency of data from component databases because the retrieval result is able to lead users to make right/wrong decision in any users' processes. The next section describes the meaning of data consistency.

Data Consistency

The data consistency in this research is based on the characteristic of interdependent data [RUSINKIEWICZ et al.,1991]. Table 1 illustrates the meaning of data consistency. Table

1 (b), data are separated to store in a file system and a database system. The consistency of data can be determined by the time that data are available in the system, such as data in the file system are consistent with data in the database system because they occur in the same year, 1985 and 1986. If an update occurs on the database, Table 1 (c), data on the file system will not consistent with data on database because data in the file system are data of year 1985 and 1986 while data on the database system are data of year 1985 and 1987.

To achieve the objective of presenting of interdependent data consistently, time values of the constituent data are used. Many aspects of time arise in the real world, three possible aspects of time are described as follows [KIM et al.,1990] [KLAHOLD et al.,1986]:

1. time of realisation or data update time;

2. time of storage into the database; and

3. time of validity.

The data update time is the time at which data in the database is updated and is recorded in the database. For example, data in the database has been updated at 12-03-86, then the data update time is 12-03-86. The data update time may be referred as a timestamp. The valid time is the time at which new data value becomes active or data are considered as up-to-date data. For example, if data in a file will count to be valid during 10-03-85 to 10-03-87 then the data valid time is the interval of (10-03-85, 10-03-87).

When interdependent data are distributed over the world, it is difficult to use only the data update time to consider the consistency. As a consequence of having different time zones, the time in different time areas cannot be compared directly. It is reasonable to implement valid time without any time conversion [KIM et al.,1990]. There are two primary

mechanisms to represent the time value. These may be defined as absolute time and abstract time.

Country	Year	Agricultures	Cereales
INDIA	1985	123.54	121.60
	1986	123.18	122.10

Table 1 (a) Original Data before Separation.

Country	Year	Agricultures
INDIA	1985	123.54
	1986	123.18

Data in the Database.

Country	Cereales
INDIA	121.60
	122.10

Data in the File.

Table 1 (b) Separated Data stored on a Database and a File.

Country	Year	Agricultures
INDIA	1985	123.54
	1987	145.09

Data in the Database.

Country	Cereales
INDIA	121.60
	122.10

Data in the File.

Table 1 (c) Data after Individual Update.

Table 1 An Example Illustrates the Meaning of Data Consistency.

The absolute time is the primary method to represent the time value which is of the form YYYY/MM/DD [KIM et al.,1990] or DD-MM-YYYY [RUSINKIEWICZ et al.,1991], where YYYY refers to year, MM refers to month, and DD refers to day. The abstract time is used in the time-concerning event [KIM et al.,1990]. The abstract time uses some special characters to define the valid time such as exclamation mark (!) and at-sign (@) etc [RUSINKIEWICZ et al.,1991]. For example, *25-Aug-86!* means 'after August 25, 1986'.

Regarding Figure 1,*year* is the data valid time unit. Then the problem relates to consistency of data will not occur after data in the file contain *year* as a key value. Therefore, the user will not receive the incorrect data after individual update data on each system occurs. However, it is possible that the data valid time has not been implemented as a key value of the original data. Thus, the only one method that can be applied to determined data consistency for the entire system is to use the data update time value. Therefore, data consistency can be determined by either the data update time or the data valid time.

Problem Domain

Regarding to the defined circumstances above, the problem domain in this research is described as follows;

1. to find methods which are able to use a file system as a sharable data source, not just databases;

2. to inform users when any change occurs in the HDDS;

3. to guarantee to users that the received data is consistent;

4. to provide consistent information while a communication failure occurs; and

5. to provide approximation routines when the user expects that a received data value is not reasonable.

System Analysis and Design

Consider the system software such as DATAPLEX, WAIS, and ANSA. There are many deficiencies of system software' s services due to the above circumstances. Firstly, the mentioned system software cannot inform users when any alteration occur in the system. This is because the mentioned system software cannot check and update the server's information. Secondly, it cannot ensure users that the received data are consistent information although merged data is presented to users. Thirdly, it is not able to provide consistent data to users while a communication failure occurs and no duplicated database exists. Fourthly, it does not provide any approximation routines when an unreasonable data arises in the system. Lastly, it is not able to access data in the file because it does not consider a file system to be a repository in the HDDS.

To achieve an efficient system software that can protect and solve the above problems, a system software called a *Computer Software Interface (CSI)* has been developed. The CSI consists of three subsystems: an Information Server System (ISS), a Query Generator System (QGS), and Preserve Data System (PDS). The ISS contains a local database called an Information Server Database (ISDB) while the PDS contains a local database named a Preserve Database (PDB). Figure 2 illustrates the system design environment.

By the characteristics of a HDDS which consist of various kinds of data schemas and Data Manipulation Languages (DML), it is not possible to implement every data model and DML into one application program. Thus, every system software has set up its own language

for users. The characteristic of the software language must be that it be user friendly, which enables users to issue their requirements either in a form of query language such as SQL, or embedded commands such as ANSAware commands. These commands will be interpreted into server statements by applying the method of data-language translator. In this research, the idea of using standard commands embedded into a user's application program has been presented in order to reduce the difficulty in handling various types of DMLs. The following sections describe functions and methods to be used in each subsystem of the CSI.

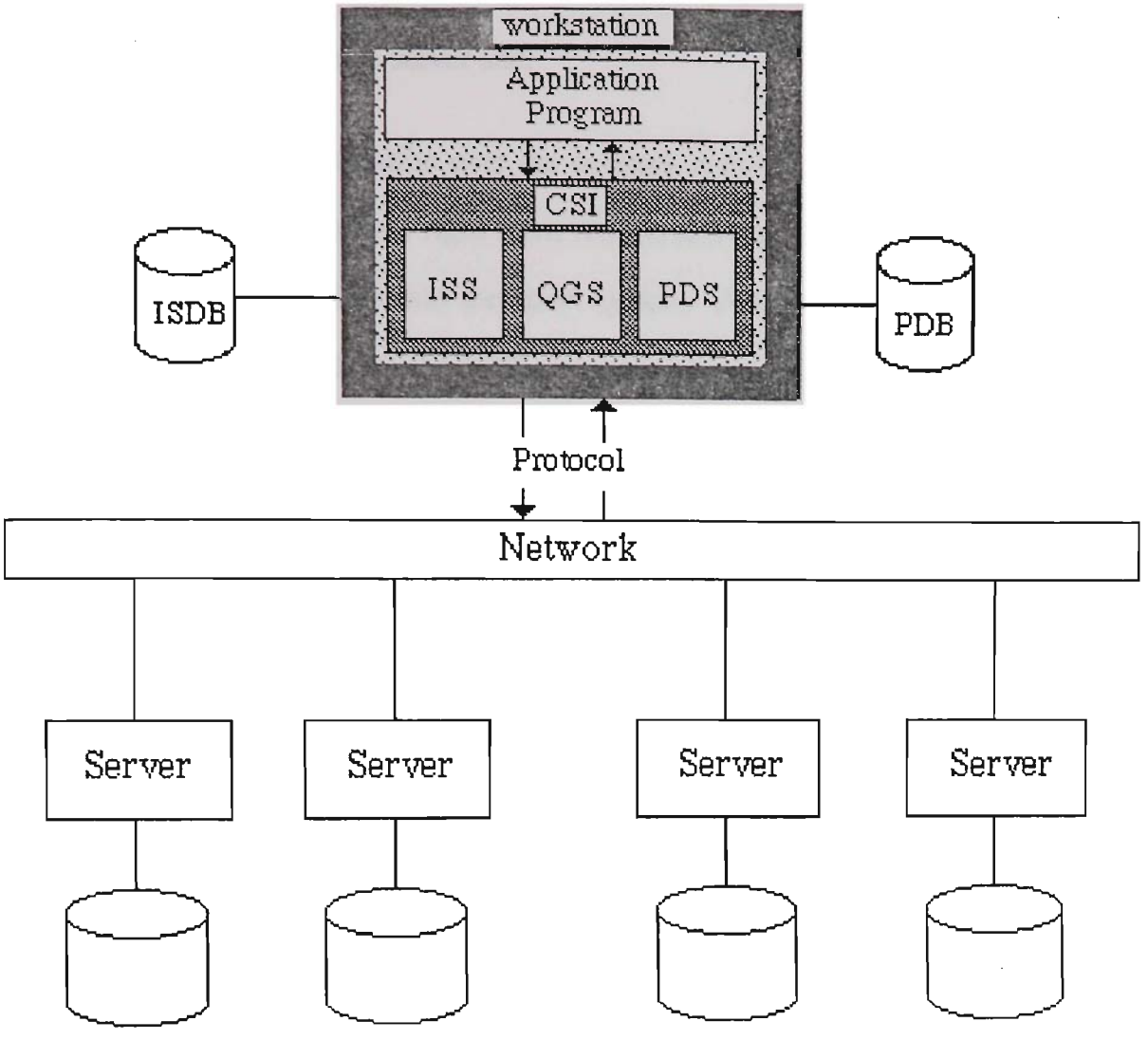


Figure 2 The Design Environment of a CSI over a HDDS.

Information Server System (ISS)

The functions of the ISS are to check user's authority, to define the required variables' addresses, to check for any changes of servers' services, to make a decision to retrieve data from the lowest cost and fastest servers, to open and close connections between the client and defined servers. The Information Server System (ISS) is implemented as an Information Server DataBase (ISDB) Manager. The implementation of the ISS is proposed

to support basic requirements before the retrieval process starts. Additionally, the ISS maintains data in the ISDB by updating information whenever any change which relates to available services occurs at a server site.

The fact that the communication over the network is unstable, it is possible that the user will not be checked for access right from remote servers. The ISS will check the user's right whenever the user logs into the client system. After the user passes the check, the user is able to gain data from the PDB even while the communication between the client and a server is broken. Thus, the data security will be achieved though the remote checking cannot be done.

In order to manage semantic heterogeneity which deals with the different meaning of data in the HDDS, the ISDB has been presented, using hierarchical data model. The function of the ISDB is to store information of available services in the HDDS, including variable names or attribute names or significant words. Once a user issues a query to retrieve data, the required variables must be verified and addresses of these values are defined by using information in the ISDB; the user's quiry consists of required data name, interested area, and conditions.

Based on the objective that information in the ISDB must be updated automatically, therefore, at the beginning of granting services, the server must register all information about granting services into the ISDB. The information stored in the ISDB includes:

1. addresses of sharable databases or files;

2. names of sharable tables or files or services;

3. locations of data schemas, data schemas, and data models of databases or files;

4. meaning of each defined data; and,

5. all access costs.

The idea of storing locations of available services' definitions and service's model into the ISDB is presented in order to maintaining information in the ISDB by the ISS. Furthermore, the concept of upgrading the characteristic of a file to be similar to a database is also proposed by storing definition of data in some accessable storage such as at the beginning of the data file. Therefore, the data schema of records will be available in an accessible storage. Thus, information about stored data in a file can be recorded into the ISDB as the information of data of a database system. Consequently, the ISS is able to check and update services' details in the ISDB without waiting for a server administrator. Moreover, users are able to check the available data definitions before issuing a query and will be informed when a change of a service occurs.

The method that the ISS uses to select the suitable servers is to consider the meaning of data (interpreted using the interested area entered by the user), access costs and data update time value. The server which has the lowest access cost will be selected. However, according to the consideration of using the PDB as a repository in the system, the data update time will be used as a value to determine whether the PDB is suitable to be a server for the user's query or not. The data update time value from every server will be compared with data update time value of each server stored in the PDB. Whenever the data update time value in the PDB is equal to the update time value from a remote server, the ISS will choose the PDB as a repository to retrieve the required information.

Query Generator System (QGS)

A function of the QGS is to generate suitable commands for every data source in the HDDS. The QGS also controls the retrieval process while communication between the client and a server fails. Furthermore, the QGS contains an approximation routine when an approximated value is required. The language generator routine in QGS is developed from the concept of data-language translator will generate a suitable statement for accessing each server, after determining data addresses and data models. The query extraction similarly to the concept of Distributed Query Generator of DATAPLEX [CHUNG,1990] may be required when the user's query is very complicated. The retrieval process will perform using extracted queries, each of which are sent to the suitable servers.

The method to generate retrieval commands for each server is based on the result from the ISS. Therefore, the QGS is active if and only if the ISS is able to obtain details of required services. Referring to the outcome of the ISS stated in the previous section, the command generator uses the address of required data, and type of the repository as a pointer that points to the command generator routine. Each generator routine will consist of retrieval command(s) which are suitable for each type of data holder. For example, if command generator named Oracle_command is implemented for generating SQL statements of Oracle DBMS then this routine will be used to generate retrieval command for every address which implements Oracle DBMSs.

Due to the uncontrolled environment, it is possible that an unreasonable data arises in the system. Whenever the user suspects that a retrieved data is incorrect, data might be required. In the user's application program, the approximation command will be embedded. The approximation routine will be a part of the QGS. The QGS will determine

the estimated method: calculation by mathematical method, or using data in the PDB. This can be done using the meaning of data defined in the ISS.

If the estimated value can be obtained from the calculation, the QGS will retrieve other related data to create the mathematical model and perform the necessary calculation. The result of calculation will be returned to the user's application. On the other hand, if the estimated value cannot be achieved by calculation, the QGS will send a message to the PDS to retrieve data from the PDB.

Preserve Data System (PDS)

The PDS supports the retrieval process while communication failure occurs after the QGS sends a message to the PDS. Additionally, it is used to support an approximation routine of the QGS when the user receives an unreasonable data. The technique used in implementing the PDS is the combination of the logging mechanism and duplicating database in the network. The PDS is implemented under the combination of the above methods by implementing a local database and updating this local database whenever a change occurs on the server site. The local database which cooperates with the PDS is called a *Preserve DataBase (PDB),* using a relational data model; the PDS is implemented as the database management system of the PDB. Therefore, the CSI is able to serve users when communication failure occurs by using data stored in the PDB.

The data in the PDB are obtained from outcomes of users' queries after the retrieval process is completed. As a result of query's extraction, the PDB will be a subset of remote data holders which contains all significant data in users' interested area. As a consequence of recording data from the HDDS into the PDB, the data-model transformation may be required due to the various data models in the HDDS. The information which is stored in the PDB is listed below.

1. The data source address and name correspond to the stored data.

2. The update time of current data with respect to the update time of the remote source; this value is issued by the server.

3. The create time of the current data schema with respect to the create time of the remote resource; this value is issued by the server.

4. The retrieval constraints which users use to limit the area of interest (they are under the same update time).

5. A list of variables or column names or significant words which is the same as the data definition of the original database.

6. The data values correspond to either variable names, or column names, or significant words.

As a consequence of being a subset of primary data sources, it is possible that the PDB be treated as a local data holder. This leads to the assumption that results of some user's queries can be achieved from the PDB. Additionally, the system software can present data from the PDB when communication between the client and a server fails.

Whenever an approximation required from the PDS, the PDS searches for the previous value with respect to the address, source name, and data name. The result of this research could either be a success or a failure. The search is successful if the required data was stored in the PDB. Otherwise, and error message is generated. The final solution from the PDS will be sent to the QGS; the QGS will pass this result to the user's application.

Evaluation of a CSI

The evaluation of the designed CSI can be summarised as follows.

1. **User Friendly Program.** The designed CSI is a user friendly program because users use embedded commands to obtain required value from the HDDS. Users need not manage any retrieval processes or deal with the difficulties in interpreting suitable commands for different servers. Furthermore, users need not bother with the various data models that exist in the HDDS. The designed CSI also provides the facility to the user in retrieving services' information in order to check the available data definitions before the user issues the query. Besides, users also are able to set their own presentation form for data after received the result from the QGS, including the consistency flag.

2. **High efficiency.** The designed CSI has high efficiency in retrieving data from various data repositories, including file systems because it is able to generate a suitable command for each server. Furthermore, it is able to check and inform a change of service which might affect users' requirements.

3. **Present Consistent Information.** Referring to the definition of consistent data, the consistency of data will be based on two time values: data update time and data valid time. The designed CSI is able to check the consistency of data by applying this rule. Whenever the time values from different servers are different, the CSI will determine these data to be inconsistent data. In such a case data from each server is presented separately to users. Additionally, the CSI can present some information to users even if there is a communication failure. The CSI obtains data, which is presented to users, by implementing a local database called a Preserve Database (PDB). Consequently, some approximated values can be obtained from the PDB. Besides, the information of

available services in the information server database will be up-to-date due to the ability in checking available services from servers has been implemented.

4. Low Cost Access. By storing every access cost of every server in the CSI, the designed CSI is able to choose the lowest access cost server for retrieving data. Furthermore, some consistent data may be retrieved from the PDB which is a local database of the system.

5. Maintenance Problems. The maintenance problems relate to many factors such as the growth rate of the HDDS, the change of services, the change of server system, etc. According to the design of the CSI, there are three main subsystems to manage all retrieval functions. Every subsystem has to maintain up-to-date information and retrieval routines in order to perform correct process with respect to user's requirements. Based on the proposed method, the maintenance routine of the stored information in the ISDB and PDB will be performed by each subsystem automatically. However, when a new server and services occur in the HDDS, their information have to be registered by the server administrator. Thus, under the following circumstances the maintenance of the ISS, QGS and PDS will be a responsibility of the server and client system administrators.

1. *A new server, and new services exist in the HDDS.*

Whenever a new server system arises in the HDDS, the server administrator has to register all details of its services to initialise the services in the CSI. Then, ISDB will be updated and ready to grant this new information to users.

2. *A new database is installed in the HDDS.*

the new DML has to be added into the QGS. Furthermore, the information of this database include services' details have to be registered into the ISDB. The effect of adding a new database system does affect the data-model transformer of the local database or PDB. In order to store data which are implemented by the new data model into the PDB, the new data model has to be transformed to a local data model that would be available in the local database. Finally, the new transformation model has to be added into the routine of the PDS.

6. Security Control. The security control will be performed by two systems. The first checking of a user's authority is the local check - preformed at the client site. The second control is performed at server sites. These processes (two-step checking) guarantee that the user is an authorised user who can access both local database and remote databases. It is possible that during the linking process between the client and a server, the communication failure occurs. In order to serve the user during the communication failure, the retrieval process will be performed on the PDB. Thus, the access check at the client can prevent an unauthorised user from obtaining data during the communication failure in spite of the fact that remote checking cannot be done.

Conclusions

This paper presents the design CSI which enables users to receive consistent information under an uncontrolled environment due to users are not the owner of required repositories, and because the communication system is unstable. The central problem is to find a method to ensure users that received data from any repositories is consistent. The other problem are such as finding methods that can present up-to-date service information to users when any changes arise in the system, present consistent information during the communication failure occurs, or support approximation routine when approximation is

needed. The present CSI consists of three subsystems: Information Server System (ISS), Query Generator System (QGS), and Preserve Data System (PDS). The idea of upgrading the characteristic of a file system to be similar to a database system has been proposed in order to use the file system as a sharable repository in the HDDS. The method to determine consistency of data is based on either data update time value or data valid time value. The concept of registering addresses of all available services' definitions provides the efficiency of the CSI to check, maintain service information, and be able to inform users when any changes occur. The combination between logging mechanism with duplicated database has been developed in order to serve users when communication failure occurs. After implementing all methods mentioned above, the CSI is able to present consistent information to users in any situations under an uncontrolled environment. The CSI is able to inform all changes of the system to users; it also provides an approximation routine to estimate a value when data is expected as an incorrect data or an unreasonable data.

References

[AHMED et al.,1991] Ahmed, R., et al., "The Pegasus Heterogeneous Multidatabase System," IEEE Computer, Vol.24, No.12, December 1991, pp 19-27.

[APM(a),1991] ANSAware 3.0 Implementation Manual, Document RM.097.01, Cambridge: Architecture Projects Management Limited, February 1991.

[APM(b),1991] The System Designer's Introduction to the Architecture, Cambridge: Architecture Projects Management Limited, April 1991.

[CHUNG,1990] Chung, Chin-Wan, "DATAPLEX: An Access to Heterogeneous Distributed Database," Communications of the ACM, Vol.33, No.1, January 1990, pp. 70-80.

[DAVIDSON,1988] Davidson, J., An introduction to TCP/IP, Springer-Verlag, 1988.

[KAHLE,1989] Kahle, B., "Wide Area Information Server Concepts," TMC Tec Report TMC202, Version 4.0, November 1989.

[KAHLE et al.,1992] Kahle, B., et al.,"Interfaces for Wide Area Information Servers," TMC, Apple Computer, NSF Network Service Center, Version 0.8, January 1992.

[KIM et al.,1990] Kim, J., et al., "Design and Implementation of a Temporal Query Language with Abstract Time," Information System, Vol.15,No.3, pp.349-357.

[KLAHOLD et al.,1986] Klahold, P., et al.,"A general model for version management in databases," Proc.12th Int.Conf. on Very Large Databases, pp. 319-327.

[MAIER,1983] Maier, D., The Theory of Relational Databases, London:Pitman Publishing Limited, 1983.

[NISO,1988] National Information Standards Organization, Z39.50-1988 Information retrieval Service Definition and Protocol Specifications for Library applications, New Brunswick: Transaction Publishers, January 1988.

[RUSINKIEWICZ et al,1991] Rusinkiewicz, M., Sheth, A., Karabatis, G., "Specifying Interdatabase Dependencies in a Multidatabase Environment," IEEE Computer, Vol.24, No.12, December 1991, pp 46-53.