A MESSAGE-BASED REMOTE DATABASE ACCESS FACILITY

by

NEIL KARL KOORLAND

B.Sc. University of Cape Town, 1982.

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

August, 1985

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date 6/9/85

DE-6 (3/81)

Abstract

We present a design for a remote database access facility, which uses a message system as its communication medium. Adopting a message-based design offers a number of advantages over more conventional connection-oriented architectures for those remote database applications where the interaction between user and DBMS involves a single query followed by a single result.

By complying with the CCITT X.400 Recommendations on Message Handling Systems, the design allows for networkwide access to remote DBMSs that is independent of the nature of DBMSs being accessed, the systems on which they reside and the network over which they are accessed.

An initial implementation using the EAN distributed message system, developed at the University of British Columbia, is described as a means of demonstrating the design's feasibility.

Acknowledgement

Both my supervisor, Dr. Paul Gilmore, and the architect of EAN, Gerry Neufeld, have given valuable advice and guidance in the formulation of a message-based design. Ed Sadowski's M.Sc. thesis work on a message-based File Transfer Facility also proved a useful source of ideas for both the design and the implementation.

I am also grateful to the backroom EAN programmers John Demco, Brent Hilpert and Rick Sample for their help in using and debugging the EAN software, and to Barry Brachman for help in debugging earlier drafts of this thesis.

The financial assistance of the National Science and Engineering Research Council of Canada is gratefully acknowledged.

Table of Contents

List of Figures

CHAPTER 1

Introduction

1.1. Motivation

Electronic message systems[1] have undergone extensive development over the past decade. They have matured to the extent where their use is pervasive and the emphasis in their design has progressed from the provision of rudimentary services to the addition of sophisticated functionality. This maturation is evidenced by the emergence of international standards for their design; most notably the CCITT X.400 Recommendations on Message Handling Systems [CCITT83a].

Although by far their most common application, the use of message systems has not been restricted to the domain of interpersonal messaging (electronic mail). The message paradigm has much to recommend its use in other applications and to date, the broader potential of message systems has been exploited to achieve, *inter alia*, File Transfer [SADOW84], Videotex Transmission and Forms Processing [VALLE84]. In recognition of this wider applicability, the CCITT has ensured that the X.400 recommendations provide a design framework for message systems in general and not only for interpersonal messaging.

This thesis is motivated, in part, by the success enjoyed by this investigation into the generality of message systems and extends their application to the accessing of

[1]The terms Electronic Message System, Computer-based Message System, Message Handling System and Message System are used interchangeably in the literature to refer to any system which supports the electronic transmission of a message from source to destination in an asynchronous store and forward fashion.

remote databases[2]. It proposes a design for a remote database access facility which uses a message system as its communication medium; i.e. a system in which queries are submitted to remote databases, and their results returned as the contents of messages. The aim is to present a design framework which can provide convenient access to multiple remote databases, possibly residing on different host systems, and managed by heterogeneous Database Management Systems (DBMSs).

The thesis has also been motivated by the existence of an operational implementation of the X.400 model with which to work. The implementation, called EAN, has been developed by the Distributed Systems Research Group (DSRG) at the University of British Columbia and is currently being used between several sites in Canada and Europe primarily for interpersonal messaging, and to a lesser extent, for file transfer. EAN has been chosen as the message system of the CDN network (CanaDiaN research network) [NEUF83] [GILMO84]. It is envisaged that the CDN network will constitute a primary communication and file transfer medium for the Canadian research community.

A natural outgrowth of this work has been to use EAN as a vehicle for an examination of the wider applicability of message systems. The ability to access remote databases is of particular use to the research community and incorporating such a facility into the EAN system would be a significant enhancement. This is especially true in light of the fact that the design proposed here is well suited to those situations, typical of research organisations, where databases reside on host systems not normally accessible to external users.

[2]A database application is said to be *remote* if a database is accessed over a network.

1.2. Proposed Design

Because of the store-and-forward nature of a message system, accessing a remote database in this manner is inherently *connectionless*; meaning that no direct connection (or virtual circuit) is established between the user and the database.

As such, it differs fundamentally from the conventional *connection-based* or "online" approach to remote database applications where a network connection is established between user and database and maintained for the duration of the user/database session. The various implications of this difference are discussed fully in the thesis.

At the outset it should be stated that this message-based approach is not intended as an alternative to *all* connection-based remote database applications. A connectionless facility could not satisfy the real-time or fast-response requirements of many applications since a message system cannot guarantee a sufficiently small delay for messages, containing queries and results, to propagate through the system.

However, there are a number of applications that involve single or batch queries which do not require real-time responses and for which a connectionless design would not only suffice, but would in fact be more suitable and economical. It is for these applications that the connectionless design, presented in this thesis, is intended.

As stated above, a key design objective is to make the design as system independent as possible. Stated another way, the design should not be dependent on the nature of the DBMS's being accessed, the hosts on which they reside or even the networks over which they are accessed. This objective is attained by framing the design in compliance with the CCITT X.400 recommendations which, as part of the ISO Reference Model for Open Systems Interconnection [CCITT82], is intended for use in a heterogeneous network environment.

1.3. Implementation

The proposed design has been implemented using much of the existing EAN message system software to access an INGRES relational DBMS [STONE76] running under UNIX 4.2Bsd[3]. This implementation was carried out both as a means of demonstrating the feasibility of the design, and as a way to illustrate the ease with which it can be implemented on top of an existing X.400-based message system.

1.4. Thesis Organisation

Chapter 2 motivates a message-based design by first surveying recent work in the areas of message systems and remote database applications and then discussing the features of a message system which make it a desirable alternative to certain types of connection-based remote database applications.

Chapter 3 presents a design for a message-based remote database access facility which complies with the CCITT X.400 recommendations. The design allows for network-wide access to heterogeneous database management systems (DBMSs) and does not entail the modification of DBMSs being accessed.

Chapter 4 is an overview of a first implementation of the design using the EAN distributed message system.

Chapter 5 presents a number of examples of the implementation in operation and also evaluates the design with respect to various criteria. The chapter concludes with a discussion of issues which are not addressed in the thesis and which could be the subject of further study.

---

[3]Unix is a trademark of AT&T Bell Laboratories.

CHAPTER 2

Survey and Design Motivation

2.1. Currently Available Remote Database Facilities

Accessing remote databases[1] is one of the most commonplace network applications and takes place within a wide variety of operational settings. Without describing all them in detail it is useful to identify three classes of systems that are currently available. Figure 2.1 illustrates the examples discussed below.

2.1.1. Virtual Terminal Access

At the most rudimentary level there are *virtual terminal* facilities, which allow users at a local site to access the remote site, on which the database resides, via a *remote login*. By furnishing appropriate authorisation (usually a userid/password) over the network, users interact with the remote database in the same way as if they were signed on directly (see Figure 2.1a). The fact that the database is being accessed over a network, is transparent to the user. In a recent survey it was estimated that over 2400 databases containing information on a wide range of topics are available to the general public in this fashion [LISAN80]. The same type of direct virtual terminal access is also in common use by organisations accessing private databases.

2.1.2. Value-added Network Services

A major problem with the virtual terminal service is the necessity to negotiate a different login procedure for each different host. This is a major inconvenience to users of

[1]Where its context does not lead to any ambiguity, the term database will be used to refer to a database itself as well as the database management system (DBMS) which controls it.

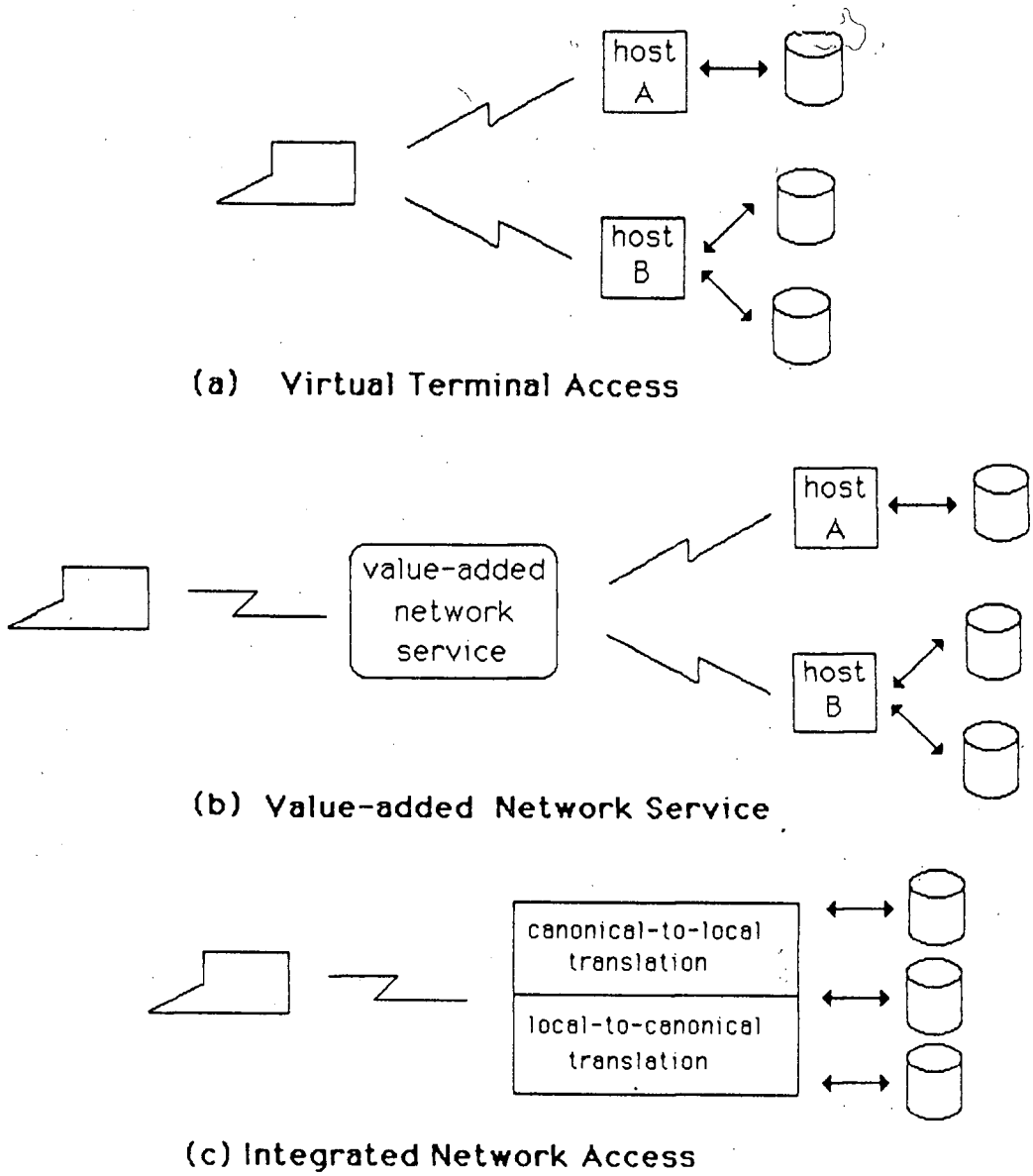

Figure 2.1 Systems Currently in Use

multiple databases residing on heterogeneous hosts.

This problem has led to the development of *value-added network services* such as

that offered by Telecom Canada's iNet 2000[2] [CUNNI83] [SOLOS84]. Value-added services act as a gateway between the user and multiple remote databases. Users login with the value-added service which, on receiving a request for access to a particular database, performs the requisite access procedure on behalf of, and transparent to, the user. In this way, users only have to go through one access procedure regardless of the number of DBMSs accessed. This configuration is shown in Figure 2.1b.

2.1.3. Integrated Network Access

A completely separate and more complex issue, of course, is the problem of contending with heterogeneous data models and query languages. iNet avoids this problem altogether by requiring users to be familiar with each different data model and query language. It only assumes responsibility for suppressing differences in system access procedures.

Recently two experimental systems, the Network Virtual Data Manager [WANG83] and MULTIBASE [SMITH81], have gone a step further in attempting to provide integrated access to heterogeneous remote databases that accounts for differences in data models and query languages as well as DBMS invocation procedures.

Both systems employ the notion of an intermediate canonical database system with a unified global (networkwide) schema and a single high-level query language. Using a Data Transfer Protocol (DTP), queries in canonical form are mapped into the representation appropriate to the target DBMS with the inverse mapping done to the result of the query (see Figure 2.1c).

To the extent that these systems are able to accommodate a number of different data models and query languages they constitute a useful contribution to the problem of

[2] iNet 2000 is a trademark of Telecom Canada.

integrated networkwide database access. However, not all data models and query languages can be mapped into canonical forms of the kind specified by NVDM and MULTIBASE and the problem of finding a truly universal canonical query language, into which all others can be mapped, is still under investigation.

2.2. The Connection-based Model

Regardless of the degree of transparency with which access to remote databases is facilitated by any of the systems described above, they all share a common feature, namely they are *connection-based* or *message-based*. As will be seen, the design presented in this thesis is *connectionless*. To appreciate the implications of this difference, the nature and inherent limitations of connection-based systems first needs to be examined.

A connection-based application is characterised by three distinct phases : connection establishment, data transfer, and connection release. These are shown schematically in Figure 2.2.

The connection-based approach is an integral part of the the International Standards Organisation (ISO) Reference Model for Open Systems Interconnections (OSI)[3] [CCITT82], at least insofar as its original formulation is concerned, and is well suited to stream-oriented applications where a series of related data units must be transferred between communicating entities. This includes many remote database applications which are interactive in nature and which require fast-response.

The fact that a connection must first be established, before the transfer of data can take place, implies that there has to be some form of prior negotiation between the

[3]A thorough discussion of issues relating to the widely embraced OSI model is beyond the scope of this thesis. The model is, however, relevant to the material presented in the thesis and therefore the reader should have a reasonable understanding of the various elements constituting the model; particularly layer 7, the application layer.

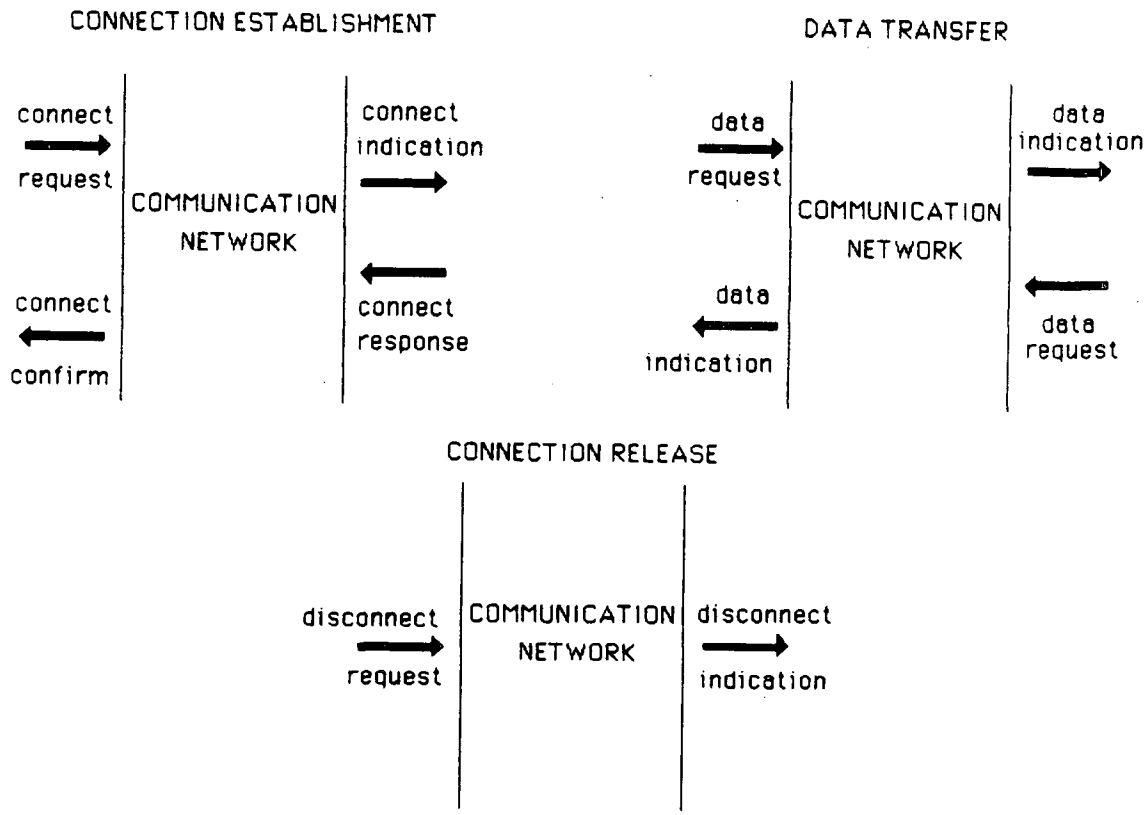

Figure 2.2 **The Connection-based Model**

communicating entities. In the case of connection-based remote database applications this is achieved by means of a remote login procedure. Because of this, the connection-based model has the following drawbacks:

- Prior to accessing a remote database, users have to have registered as users of the remote site in order to obtain access authorisation and this can involve a substantial delay if the database is being accessed for the first time.
- While issuing userid/passwords to every remote user might suit the needs of commercial database vendors whose sole function is to facilitate access to their data-

bases, it is clearly unacceptable for most other organisations because of the administrative and security problems it would create. Such is the case, for example, when a research organisation wishes to make available certain information from a database in their system to external users. In order to do this, the system administrator has to maintain a record of external users permitted access to the system, a task that might be prohibitively time-consuming. Although this problem is often solved by allowing access to the system by all remote users through a single "guest" account, one is still faced with serious security problems if, as is most likely, it is impossible to prevent the guest user from using other facilities on the system.

- The connection-based model dictates that the connection be maintained for the entire duration of the user/database interaction even during times when no data is transferred. This can be costly over long distances.
- It might not be possible to establish a direct connection, between user and remote DBMS, because of physical constraints imposed by the network. For example, either the local or the remote system might not be part of a switched network or they might belong to different networks and the gateway between them cannot support a virtual circuit.

2.3. The Connectionless or Message-based Model

Having discussed the problems associated with adopting the connection-orientated approach to remote database applications we now describe the connectionless or message-based model of data communication and how it might be exploited in remote database applications.

Unlike the three-phase connection-orientated approach, the connectionless data model involves the transmission of a single independent data unit from source to

destination without prior negotiation and subsequent connection release. This is illustrated in Figure 2.3 . At the application level these data units are called *messages* and the application processes, which cooperate to achieve connectionless data transmission, are collectively known as a *Message Handling System.*

Intimately associated with the concept of connectionless data transmission is the notion of a *store-and-forward* network; i.e. the notion that a message is built as a complete data unit at its source and propagated through intermediary nodes until it reaches its ultimate destination. Each node along the way assumes responsibility for the message for the duration of its possession and is not obliged to pass it on within any time limit.

2.3.1. Suitable Applications for Connectionless Transmission

Chapin [CHAPI82] [CHAPI83] has identified a number of applications for which connectionless data transmission is better suited than connection-based transmission. These include :

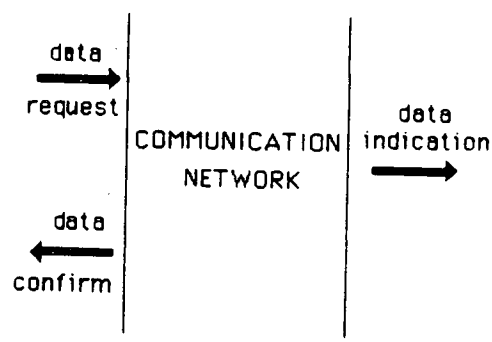

Figure 2.3 The Connectionless Model

Inward data collection – the periodic sampling of a number of remote data sources. Instead of having to poll each remote site for data, the sampler is sent data as the content of a message.

Broadcast and multicast communication – the dissemination of a single message to a number remote destinations.

Office information exchange – store-and-forward transmission of multimedia documents.

Request-Response application – here a server process associated with a remote resource is responsible for processing requests for the resource, which are submitted to it as messages by request sources, and returning response messages to each request source. In these applications, the typical interaction between source and server involves a single request followed by a single response.

2.3.2. Relevance to Remote Database Applications

It is the last example which is of particular interest here, since many remote database applications are of the request-response type in that they only involve a single query followed by a single result. Directory services, transaction-based systems and batch query applications are all good examples of request-response type remote database applications.

Figure 2.4 shows remote database access modelled as a request-response application. At present, even those remote database applications which are more suited to this message-based approach, are implemented in a connection-based environment. As such, they are susceptible to the problems, discussed earlier, that are associated with connection-based systems. We suggest these problems could largely be avoided if,

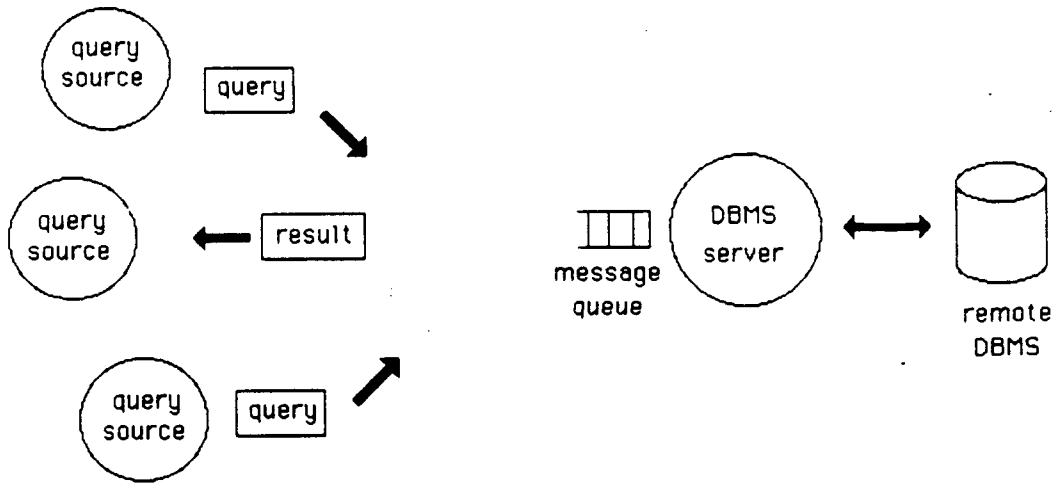

Figure 2.4 Message-based Remote Database Access

instead, they were implemented in accordance with the message-based approach. Specifically, implementing them in a message-based environment would have the following advantages :

- Since all queries are submitted to a remote DBMS indirectly through a server process operating at the remote site, users only have to direct their queries to the server process without the need to login at the remote system. There is no need for them to register as users at the remote site prior to accessing the DBMS.
- Since external users only have access to the services offered by the DBMS server process, and not any other resources on the remote system, administrative and security considerations are greatly simplified. Naturally, the server process is free to screen and reject any queries it considers to be unauthorised.

- Data transmission costs are reduced. Since there is no need for maintaining a connection between users and remote sites, only the cost of transferring messages, containing queries or results, is incurred.
- Results returned to the user as messages, can be stored, retrieved, edited, forwarded to other users, and combined with interpersonal messages.
- Potential accessibility to remote databases is greatly increased as queries, destined for sites unreachable via a direct connection, can be relayed by the message system through intermediary sites which lie on a path to the target site.

It should be emphasised, however, that because of the inherent store-and-forward nature of message systems, a connectionless design is clearly only appropriate for remote database applications which are request-response oriented. Those applications which involve a protracted interaction between user and DBMS and which require fast response times are not appropriate for the message-based design presented here. While this certainly limits the applicability of the message-based design presented in the next chapter, the fact that there are many request-response type remote database applications, warrants an investigation into the feasibility of a message-based design.

CHAPTER 3

A Message-based Design

3.1. Design Issues

Having discussed the desirability of a message-based remote database access facility for request-response applications, we now examine the issues which have to be addressed in designing such a system. It is useful to divide these issues into two main categories which can then be dealt with separately.

Firstly, there are those issues which concern the message system itself, regardless of its application. These include :

- **Message submission and extraction.** This concerns the nature of the interface between the sender and the message system, when a message is submitted for delivery or when a reply has been received. In most message systems this interface embodies some notion of a "mailbox" or a "mailing slot".
- **Message transfer.** The manner in which messages are transferred reliably from originator to recipient in a store-and-forward fashion.
- **Message format.** All messages contain not only data but also control information such as a description of the data, the source and destination address, and the quality of service to be applied to the message. A message must therefore conform to an agreed upon structure in order for all entities handling the message in the system to interpret this information correctly.

Secondly, there are those issues which relate to how the message system can be used to facilitate remote database access in particular. The issues that have to be dealt

with here are :

- **The user interface.** How does the user formulate and dispatch a query or receive its result as a message, and what kind of functionality should be provided in assisting the user in doing this.
- **DBMS interface.** How are queries submitted to the DBMS for processing once they have been delivered and how results are submitted to the message system for return to the user.
- **Security and administration.** While all users of the message system have the ability to send queries to a remote DBMS, there should be a mechanism that ensures that only messages sent from authorised users are actually processed. It should also be possible to carry out various administrative tasks such as the collection of usage statistics and accounting information.

Because of the complexity of these issues, designing a message-based remote database access facility from scratch would be a formidable problem. Fortunately, a design for a message system which adequately addresses most of them is already in existence in the form of the CCITT X.400 recommendations on Message Handling Systems proposed in 1983 as an international standard[1].

The fact that the X.400 recommendations are intended as an international standard for message systems, makes their adoption in this thesis, as a basis for a message-based remote database facility, a logical design decision and one which reduces the design task to manageable proportions.

[1]In actual fact, the CCITT recommendations consist of a set of eight related but separate proposals, each dealing with a specific aspect of message systems. Where context does not lead to any ambiguity, they are referred to collectively here by the first recommendation, X.400.

3.2. The CCITT X.400 Recommendations on Message Handling Systems

The X.400 recommendations define a distributed architecture for a message handling system. As shown in a simplified form in Figure 3.1, users send messages to each other via their respective *User Agents (UAs)*.

As the name implies, a UA acts as a server process on behalf of the user and is responsible for the proper formulation, management, submission and reception of messages.

The use of the term "mailbox" has been avoided in the CCITT recommendations because it would lead to various ambiguities. But for the sake of clarification, the UA can be loosely thought of as the user's mailbox. An important point to note here is that although a "user" is a person in an interpersonal messaging environment, there is nothing implicitly or explicitly contained in the X.400 specifications which restricts the

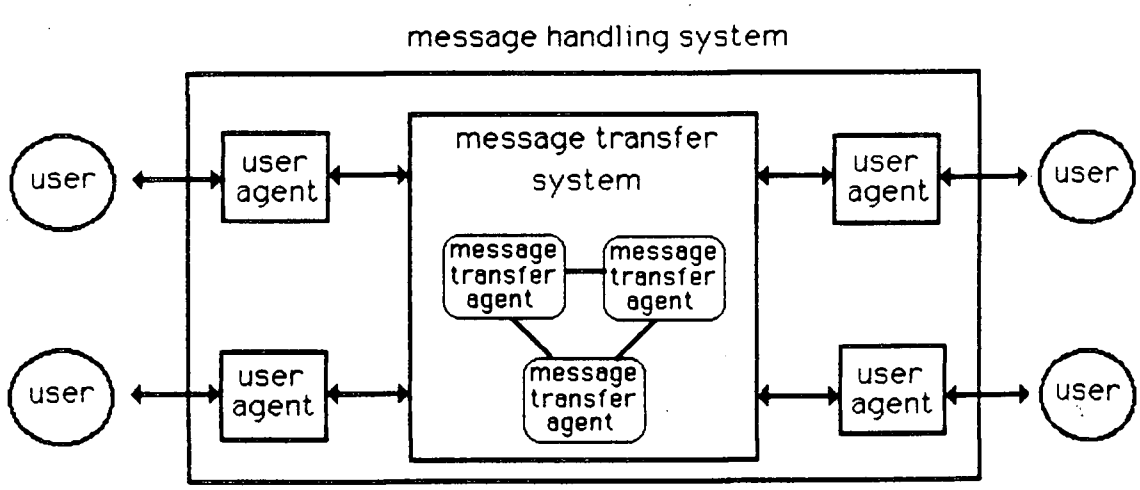

Figure 3.1 CCITT X.400 Model for Message Handling Systems

nature of the user. A user can equally be, for example, an automated process.

UAs communicate with each other through the *Message Transfer System (MTS)* which consists of nodes called *Message Transfer Agents (MTAs)*. There is a many-to-one relationship between UAs and MTAs; i.e. each UA is associated with only one MTA but each MTA can be associated with more than one UA. Messages propagate through the MTS, from the MTA associated with the originating UA to the MTA associated with the recipient UA, in a store-and-forward fashion. As is the case with all store-and-forward systems, no guarantee is given as to the speed with which messages are delivered, although it is possible to specify that a message should not be delivered beyond a certain time.

In the terminology of the OSI reference model, UAs are said to reside at the *User Agent Layer (UAL)* which exists above the *Message Transfer Layer (MTL)* where communication between MTAs take place. Both the UAL and the MTL reside at the Application layer of the OSI model. Messages are submitted to, and extracted from, the MTS at the interface between the UAL and the MTL. The clear delineation of responsibilities between the MTS and UAs is an important feature of the X.400 model because it allows for both their logical and physical separation.

As far as naming and addressing are concerned, X.400 specifies a complete scheme for all entities in the message system. Briefly, each UA must be associated with at least one *Originator/Recipient (O/R) Name* which uniquely identifies the UA in the message handling system. An O/R Name which in some way implies the physical address of the UA (although not the route that should be taken to get there), is known as an *Originator/Recipient (O/R) Address*. An O/R name is *hierarchical* as opposed to *flat*, in order to allow for distributed control.

The CCITT recommendations also specify the format of a message in detail. As shown in Figure 3.2, a message consists of an *envelope* and its *content* both encoded and structured according to a given *protocol*. In the case of the envelope, the protocol used is specified in recommendation X.411 [CCITT83c].

Information on the envelope, such as the recipient's O/R Name, is used exclusively by the MTS to deliver the message from its source to its destination. This information is of course common to all message-based applications and the protocol is therefore used for all messages regardless of the application being supported.

The message content, on the other hand, is used exclusively by UAs and remains transparent to the MTS. The intention here is that a different content structure protocol can be defined for each different application that best reflects the nature of the application at hand.

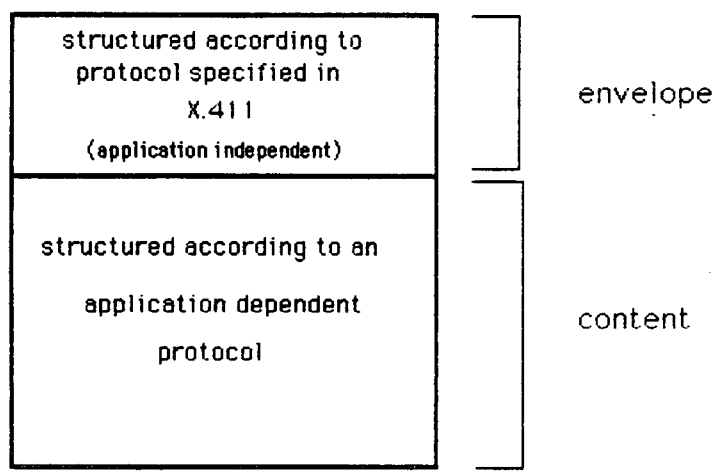

Figure 3.2 CCITT X.400 Message Structure

Each UA is therefore thought of as belonging to a *class* of UAs where the UAs comprising a particular class communicate according to their own protocol. However, they all share the same envelope protocol and MTS. To date, only a protocol for interpersonal messaging (X.420) has been included in the CCITT recommendations [CCITT83d] and is given in Appendix A.

3.3. Facilitating Remote Database Access

Accommodating remote database access, within the X.400 architectural framework is straightforward. Three components of the X.400 model must be exploited.

Firstly, each user must have access to a UA capable of structuring a query as the contents of a message according to a given protocol and submitting it to the MTS for delivery to the remote DBMS. The UA is also responsible for receiving messages containing results from the MTS on behalf of the user. It can provide an arbitrary degree of functionality in performing this task. For example it can provide sophisticated query/result storage facilities and an interface to UAs which provides interpersonal messaging.

Secondly, UAs are required to act as server processes on behalf of remote DBMSs. Their task is to accept messages containing queries from the MTS and submit it to the DBMS for processing. Once the query has been processed, the UA must obtain the result of the query and submit it to the MTS for delivery back to the user's UA. For the sake of clarification, these UAs will be referred to as *Database User Agents (DBUAs)* in order to distinguish them from the UAs associated with each user.

Strictly speaking, the X.400 model requires that each remote DBMS be associated with a different DBUA. However, there are certain advantages to having only one DBUA acting on behalf of a number of DBMSs, not the least of which is the decreased

implementation overhead incurred in situations where a large number of DBMSs reside on one host. In this thesis, therefore, it is assumed that one DBUA can act on behalf of more than one DBMS.

Figure 3.3 illustrates the relationship of UAs and DBUAs to each other and to the message system. In the terminology of X.400, UAs and DBUAs using the MTS in this way are said to constitute a class of UAs which cooperate to support remote database access.

Thirdly a protocol appropriate for remote database applications must be defined. The protocol should specify the structure of messages containing queries as well as messages containing results. The protocol elements which define the structure of queries must make it possible for the remote DBUA to find and extract the information it needs in order to invoke the appropriate DBMS correctly as well as locating that part of the

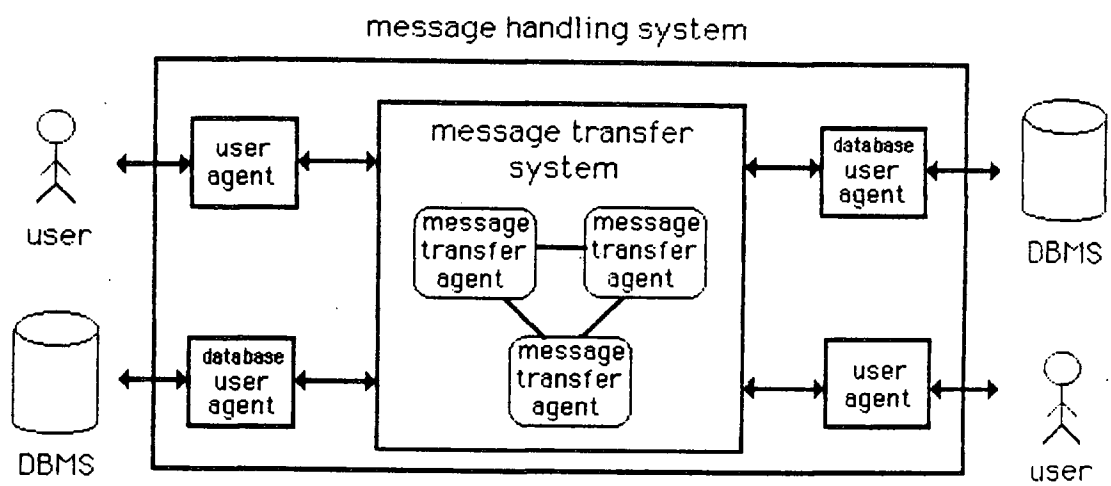

Figure 3.3 Using X.400 as a Remote Database Access Facility

content which constitutes the actual query to be passed onto the DBMS. Similarly, the protocol must define the structure of a result so that the user's UA is able to find and extract a result from a message on its arrival. The next section discusses in greater detail the protocol chosen in this design.

3.4. A Protocol for Remote Database Applications

The query-result protocol proposed here is shown below using the notation specified in CCITT recommendation X.409 [CCITT83b]. The protocol is hereafter referred to as the *Database Access Protocol.*

```
MODULE
DBAPDU ::= CHOICE {
                [0] IMPLICIT Query,
                [1] IMPLICIT Result }
Query   ::= SEQUENCE {
                [0] IMPLICIT QueryHead,
                [1] IMPLICIT QueryBody }
QueryHead ::= SET {
  Recipient DBUA  [0] IMPLICIT ORName
  Recipient DBMS  [1] IMPLICIT ORName
  Originating User [2] IMPLICIT ORName
  QueryID         [3] IMPLICIT STRING
  Userid          [4] IMPLICIT STRING OPTIONAL
  Password        [5] IMPLICIT Encrypted OPTIONAL
  InvokeCommand   [6] IMPLICIT STRING OPTIONAL }
QueryBody ::= [0] IMPLICIT IA5Text  -- Plaintext
Result ::= SEQUENCE {
                [0] IMPLICIT ResultHead
                [1] IMPLICIT ResultBody }
ResultHead ::= SET {
  Originating DBUA [0] IMPLICIT ORName
  Originating DBMS [1] IMPLICIT ORName
  Recipient User   [2] IMPLICIT ORName
  QueryID          [3] IMPLICIT STRING
  Error Condition  [4] IMPLICIT INTEGER (NoError (0),
                                         NoDBMS (1),
                                         UnavailableDBMS (2),
                                         NoAuthorization (3),
                                         BadInvocation (4) ... } }
ResultBody ::= [0] IMPLICIT IA5Text  -- Plaintext
END
```

In the terminology of CCITT X.400, a message is made up of data structures called *protocol data units (PDUs)* each of which may consist of other PDUs.

Each PDU in a message is identified by an integer code, the interpretation of which is dependent on the data unit's context in the message (denoted by the keyword *IMPLICIT*). A message is therefore hierarchical in structure. Using integer codes instead of character strings to identify protocol elements makes the protocol language independent[2].

According to the notation above, the content of a message in a database application is known as a *Database Application Protocol Data Unit (DBAPDU)*. A DBAPDU is either a *Query* or a *Result*. This alternation is denoted by the keyword *CHOICE*. Queries consist of a *QueryHead* and a *QueryBody*. The keyword *SEQUENCE* specifies that the QueryHead must precede the QueryBody.

A QueryHead consists of a number of fields or elements some of them mandatory and some of them optional (denoted by *OPTIONAL*). The keyword *SET* specifies that the ordering of elements within the QueryHead is immaterial and can differ from message to message. Included here are the O/R Name of the DBUA that must receive the query, the DBMS for which the Query is intended, and the user which originated the query. Each QueryHead also has a QueryID which is used to uniquely identify the query. This is only used for reference and administrative purposes.

The three optional QueryHead fields are used by the DBUA to invoke the appropriate DBMS. Their inclusion in the Database Access Protocol requires a word of explanation. In general, some combination of userid, password and user-supplied parameters must be passed to a DBMS when it is invoked. A DBMS uses this information both as a

[2] The specifications *O/RName, STRING, Encrypted, IA5Text* and *INTEGER*, are general purpose protocol elements whose detailed composition is specified in X.411.

means of determining the user/DBMS interaction environment (e.g. which database(s) the user requires access to), and as a user-authorisation mechanism. But since different DBMSs differ with respect to precisely which of these it requires, it is necessary to provide sufficient flexibility to include some, all or none of these within the message structure. This flexibility is achieved by allowing the user to determine which of the three optional fields are included in the query message. It was felt that these three fields were probably sufficient for most DBMSs and host operating systems. However, it is a simple matter to extend the protocol with additional fields should a need for them arise as a result of the peculiarities of certain DBMSs not anticipated here.

A QueryBody contains the actual query. Results consist of a *ResultHead*, used by the UA in referencing a previously submitted query, and a *ResultBody*, which contains the actual result.

According to this protocol, neither the UA which submits a query, nor the DBUA, which receives it, is expected to have any knowledge of the data models or query languages of the DBMSs on whose behalf it operates. Both the UA and the DBUA treat the QueryBody and the ResultBody as a byte-stream without any associated semantics. It is up to the DBMS to interpret the query and the user to interpret the result. Only the control information contained in the QueryHead and ResultHead is interpreted by the UA and DBUA.

In this way, neither the UA nor the DBUA is dependent on the nature of individual DBMSs thereby making the system extendible and easy to implement. Under this protocol, it is entirely up to the user to ensure that queries submitted by their UA are valid for the target DBMS. This is the same policy adopted in connection-oriented systems such as iNet 2000.

Using the X.400 architecture in this way has a threefold advantage. Firstly, it provides *generality* in the sense that X.400 conforms to the OSI reference model thereby facilitating its use in network environments comprising heterogeneous systems. Secondly, it is an *extendible* design in that it is easy to add DBMSs to the system. All that is required is the establishment of a DBUA at the site on which the DBMS resides or, if a DBUA already exists at the site, the updating of that DBUA. Thirdly, it is *compatible* in that no change is required to the software comprising remote DBMSs, since a DBMS perceives the DBUA as a conventional user of the DBMS.

CHAPTER 4

Implementation

4.1. Implementation Objective

This chapter describes an implementation of the design discussed in the previous chapter. The design has been implemented more as a means of demonstrating its feasibility in an operational setting, than to provide a fully functional and reliable remote database access facility of the type that would be required by the user community.

Specifically, it was felt that the ability to access at least one remote DBMS using a message-based system would be sufficient to demonstrate its feasibility.

The setting of a modest objective was mainly a result of limited time and computing resources. At the outset of the implementation access was only available to two host machines (a DEC VAX 11/750 and a SUN Workstation[1]) both running under the same operating system (UNIX 4.2Bsd) and one DBMS (an INGRES relational DBMS).

4.2. Use of the EAN Distributed Message System

The task of implementing the design was greatly simplified by the existence of the EAN Distributed Message System. As mentioned earlier, the EAN message system is currently in use at several test sites in Canada and Europe for interpersonal messaging. EAN was developed by the Distributed Systems Research Group (DSRG) at the University of British Columbia over a period of approximately four years.

[1]SUN Workstation is a trademark of SUN Microsystems Inc.

The entire system is written in *C* and has been designed as a portable application system with all host operating system dependencies identified and isolated in the code. To date EAN runs under MTS, UNIX 4.2Bsd and VMS on a variety of machines.

As an implementation of the X.400 recommendations for interpersonal messaging, those elements of the EAN message system, which specifically support interpersonal messaging (i.e. the User Agent Layer) have been separated from the those elements which provide generalised message transfer (i.e. the Message Transfer Layer)[2].

This means that EAN's Message Transfer System can be used in its entirety, and without any modification, to support other message-based applications. As mentioned in the previous chapter, this generality is one of the primary design principles underpinning the X.400 model. In fact, EAN's MTS has already been used to provide a store-and-forward file transfer facility [SADOW84].

Needless to say, the implementation of the underlying MTS constitutes the major software engineering component of any message-based system. Most of the design and implementation efforts of the DSRG, in developing EAN, have been focussed on the provision of an extensive and reliable MTS with a relatively minor effort being required for the implementation of the User Agent Layer. Therefore, the ability to utilise an already existing MTS, in providing remote database access, represents a substantial reduction in implementation effort.

All that is required, then, is the provision of a class of UAs and DBUAs capable of transferring queries and results according to the Database Access Protocol described in the previous chapter. Each user must have access to a UA and each DBMS must be

[2]At present, this separation is achieved through program module linkage. However, it could be achieved by structuring the UAs of the UAL and the MTAs of the MTL as separate processes (possibly residing on different machines), and having them communicate via some appropriate interprocess communication mechanism. It is intended that EAN will eventually employ this structure.

associated with a DBUA. Considering the widespread usage enjoyed by message systems for interpersonal messaging, this simplification serves to strengthen the case for a message-based remote database access facility as a desirable enhancement of the capabilities of existing message systems.

Those aspects of EAN's MTS which bear directly on the implementation of the remote database access facility will be discussed here, where appropriate. However, the MTS can, for the most part, be regarded as a reliable transfer service for transferring messages between UAs and DBUAs with the exact nature of its operation being of no concern for the purposes of this thesis.

While UAs and DBUAs are considered as belonging to the same class, by virtue of the fact that they use a mutually understood protocol to support a particular application, their responsibilities differ fundamentally. This in turn, means that their respective implementations bear little resemblance to each other, apart from the manner in which they use EAN's MTS to send and receive query/result messages. Their implementation will therefore be described independently in the following discussion.

4.3. The Remote Database User Agent

4.3.1. A Hybrid User Agent

Implicit in the X.400 recommendations is the requirement that a UA should only support one type of application, i.e. if a number of message-based applications are supported in compliance with the X.400 model, then the user has to access a separate UA for each of those applications. Strictly speaking, the remote database UA should be unrelated to EAN's interpersonal UA and the remote database UA is presented as a separate entity in the previous chapter.

Notwithstanding this, it was decided to incorporate the functions of the remote database UA as an enhancement to the existing interpersonal UA; i.e. to merge the two UAs into one and to allow a user to send queries and receive results along with interpersonal messages. This violation of the X.400 model was considered to be acceptable because of the greatly simplified implementation that it allowed for.

4.3.2. Use of Existing Software for the User Agent

As in the case of the MTS, much of the existing EAN software was used to implement the remote database access UA. This was possible because the existing interpersonal UA contains many of the functions required by the remote database UA. Although a remote database UA differs from a UA in an interpersonal messaging application in the nature of messages that are sent and received as well as the type of services offered to the user, they both have the task of assisting the user in the preparation (construction) of messages as well as providing message storage capabilities.

4.3.3. Implementation Overview of the Remote Database UA

The existing EAN interpersonal UA is a large application program (approximately 12000 lines of C code) and provides the user with all the interpersonal messaging services stipulated in the X.400 model (recommendation X.420 [CCITT83d]). In addition it also provides sophisticated message storage and retrieval facility and user profile definition[3].

Users interact with the interpersonal UA using the User Command Interface (UCI) which consists of 28 commands, each an implementation of a specific interpersonal mes-

[3]The X.420 recommendations only specify the composition of an interpersonal message and not the nature of the user interface. The features offered by the user interface can incorporate an arbitrary degree of sophistication. In the case of EAN's UA, the large program size is a result of the inclusion of extensive functionality such as user profile definition and a message folder abstraction.

saging service[4]. Incorporating a remote database UA into the EAN UA was a simple matter of adding a new command to the UCI which implements the remote database UA. This command has been called the *QUERY* command.

4.3.4. Formulating a Query

The content of an interpersonal message consists of a ***heading*** and a ***body***. The heading, like the envelope, contains various delivery parameters which are interpreted in the appropriate manner by interpersonal UAs. The body of the message contains the actual text of the interpersonal message.

Since the QUERY command is merely part of the interpersonal UA, its task is to structure the body of an interpersonal message according to the Database Access Protocol. If the remote database UA had been implemented as a seperate program form the interpersonal UA, the entire message content would be structured according to the Database Access Protocol (as shown in Figure 4.1a). This complication is the result of combining the two UAs. Figure 4.2b shows the structure of an ***interpersonal*** message which contains a query.

The *Network User Address* of the recipient DBMS as well as the three optional protocol elements of the QueryHead are taken from arguments supplied in the QUERY command[5]. The user has the option of giving the NUA of the recipient DBMS either as a full NUA or as an alias which is then interpreted from the user's profile definition. Optional fields are only included in message structure if they are given as arguments to the QUERY command.

---

[4]For a more detailed description of the EAN User Agent see [DADOU84].

[5] In EAN O/R Names are in fact O/R Addresses and are known as Network User Addresses (NUAs).

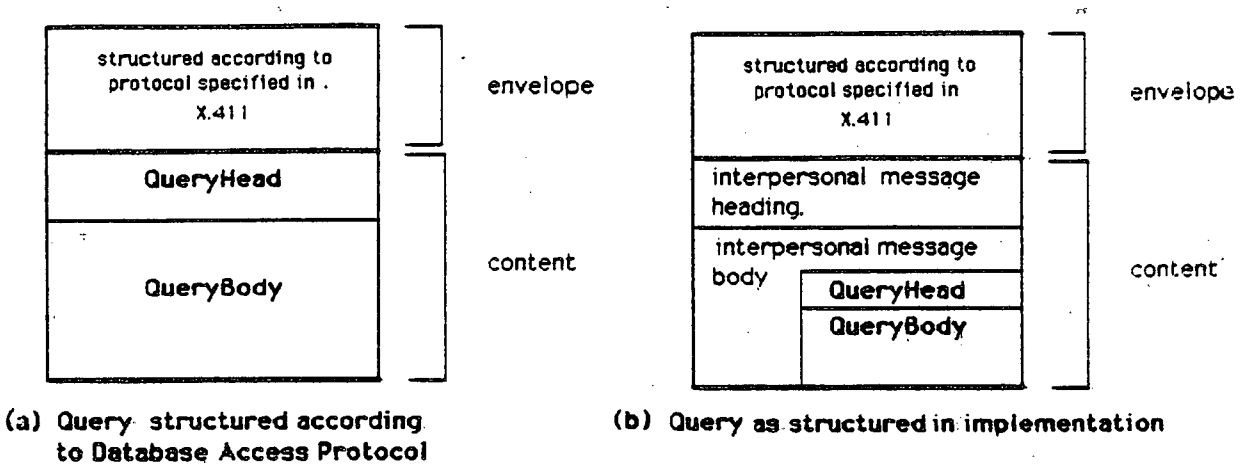

Figure 4.1 Message structure of a query

The rest of the protocol elements are generated automatically. The NUA of the recipient DBUA is formed by substituting the user name part of the recipient DBMS's NUA with the reserved user name *dbagent*. The hierarchical naming and addressing scheme used by the message system guarantees that the resultant NUA can be unambiguously determined as the NUA of the DBUA responsible for that DBMS.

The QueryID field is made unique by concatenating the NUA of the originating UA with a timestamp. The intended use of the QueryID is for accounting and query processing status reporting which have not been included in this first implementation. The QueryID has been included for the sake of completeness.

The QueryBody is prompted for in the same way as the text of an interpersonal message and the user can include an already existing text file instead of having to type the entire query. This is useful for queries submitted periodically with little or no

alteration or for sending the same query to a number of DBMSs.

A useful feature to provide in a remote database access facility is the ability to keep a record of a query submitted to the MTS for later reference. For example, if a result is received at some later stage stating that there was a syntax error in the original query, it would be useful for the user to have the ability to access the original query for examination. The EAN UA already includes a sophisticated interpersonal message storage/retrieval facility and this implementation has extended it to handle queries (and results) as well.

The heading part of the message content is also generated automatically and consists of a *TO* field (the NUA of the recipient DBUA), a *FROM* field (the NUA of the originating UA). Many of the fields in the message envelope, the content heading and the query heading are therefore the same. This redundancy is more apparent than real, since the fields are all interpreted at logically distinct levels in the message system. This is a feature of layered systems in general, but is considered more as an asset than a liability because of the increased flexibility it allows.

Once the entire message content is constructed it is added to an envelope and submitted to the MTS for delivery to the recipient DBUA. Once the MTS accepts responsibility for the message, the QUERY command exits and the user is free to use any other service offered by the UA.

Incorporating the QUERY command into the EAN UA resulted in an additional 400 lines of C code, a minor addition relative to the overall size of the UA. Keeping the QUERY command small was possible because all the message construction and MTS interface routines are already included in the UA.

In EAN, messages are constructed by linking together basic data structures called *ENODEs*. Each ENODE corresponds to a X.400 protocol element with each ENODE containing the context sensitive integer code identifying which protocol element it is. Routines for constructing specific ENODEs such as an O/R Name or a character STRING as well as general purpose routines for linking ENODEs together or locating a particular ENODE in the message structure were already part of the EAN interpersonal UA.

4.3.5. Processing a Result

A result of a previously submitted query is delivered to the UA of the originating user[6]. The UA is responsible for allowing the user to manipulate the result in the same way that it provides various services to the user for processing incoming interpersonal messages. As mentioned above, the result will also contain the original query. Since remote database access service is built into the interpersonal UA, it is necessary to treat results in exactly the same manner as interpersonal messages; so as to avoid having the user confronted with two different types of messages which require differential treatment. Treating results and interpersonal messages identically, allows the user to combine results with interpersonal messages for forwarding to other users. It also means that existing UA software can be used to store, retrieve, view, edit and print results.

However, if the result is returned to the UA in the format specified by the Database Access Protocol, it will be misinterpreted by the UA because it would contain protocol elements, in the body part of a message, which are not defined in the X.420 interpersonal protocol.

---

[6]In actual fact, the MTS retains the result (message) until the UA explicitly requests the MTS for any messages that might have arrived for the UA. Once a message is delivered to the UA, the MTS is no longer responsible for it.

Therefore, in the interest of uniform treatment of results and interpersonal messages, it was decided not to encode results according to the Database Access Protocol but instead, to have results returned as normal interpersonal messages (i.e. containing only interpersonal protocol elements in the body part of the message content). Similarly queries and results stored for later reference in the UA message file system have to be stored in interpersonal message format and not in Database Access Protocol Format.

If a separate UA was used to implement remote database access, then naturally those elements of the Database Access Protocol pertaining to a Result, would be used. This is the intention of the design and would be implemented were sufficient functionality required of the remote database UA to warrant a separate UA.

4.4. The Database User Agent

The DBUA has been implemented as a special EAN UA with the reserved NUA dbagent@*host* where *host* is a qualifier which uniquely determines the O/R Name of a particular DBUA. Here again, the use of existing EAN software contributed to a simple implementation the different aspects of which are discussed below.

4.4.1. Invoking the DBUA

Unlike the UA, which is invoked manually by the user, the DBUA is a server process which has to be invoked automatically upon the arrival of a message (query). Fortunately, the EAN message system already provides a mechanism for doing this. It is possible to arrange for the MTA responsible for a DBUA to wake up a program on arrival of a message and this feature is used to invoke the DBUA automatically.

To avoid possible problems with concurrency, the first thing done by a DBUA on being awaken, is to check that no other DBUA is already active. If another DBUA is

active then it exits immediately. This ensures that only one DBUA is active at a time for any given DBUA NUA. A DBUA remains awake as long as messages arrive; i.e. queries which arrive while a DBUA is already active, will be processed on a FIFO basis by the active DBUA.

4.4.2. Query Validation

The DBUA has to ensure that incoming messages are valid with respect to two criteria. Firstly, all messages have to be checked to see that they are in fact queries. Since a DBUA has a NUA (a well-known one at that), there is nothing stopping someone mistakenly sending an interpersonal, or any other kind of message, to the DBUA. To prevent this, the body part of the content of incoming messages is examined and if it does not contain a QueryHead, it is rejected and an appropriate error message is returned to the originating UA (the NUA of which is obtained from the *TO* field of the content heading)[7].

Secondly, the DBUA must check to see that the QueryHead contains sufficient information for the appropriate DBMS to be invoked correctly. To begin with the NUA of the recipient DBMS, which appears in the QueryHead, must be checked against a table containing all the NUAs of DBMSs on whose behalf the DBUA acts. If no match is found then a result is returned to the originating UA indicating that a DBMS with that NUA does not exist.

If the NUA for that DBMS does appear in the file, then the DBUA has to invoke the DBMS using the information contained in the three optional QueryHead fields. This means that individual DBUAs, unlike UAs, will differ (in detail if not in overall struc-

[7]Another possibility here, is that if an incoming message is a *status report* on a message submitted previously by the DBUA but which could not be delivered by the MTS for some reason. In this case, the status report is ignored.

ture) from each other since different DBMS (or identical DBMSs residing on different host operating systems) have to be invoked differently. This system dependency is isolated in the DBUA implementation by confining the invocation to a separate program module. This allows for simple addition (or deletion) of DBMSs. If a DBMS has to be added to the DBUA, then all that is needed is the addition of a program module which can invoke the DBMS in the correct way. Needless to say, each DBMS is invoked by the DBUA as if the originating user had invoked it. This ensures that only authorised users access the DBMS. Once the DBMS is invoked, the DBUA is responsible for passing the QueryBody to the DBMS in such a way that the DBMS perceives the DBUA as a conventional user.

4.4.3. Returning a Result

A DBUA has to construct a result containing both the original query and the result obtained from the interaction with the DBMS. This is done using the same message construction routines as are used in the UA. As has already been explained, the result does not conform to the Database Access Protocol in this implementation. Instead, a conventional interpersonal message is constructed with the query and the result placed as plaintext in the body of the message content. The content heading is built to contain the delivery parameters necessary to ensure that the result is returned to the originating UA. Once the message content has been built, it is added to an envelope and submitted to the MTS.

CHAPTER 5

Results and Evaluation

4.5. Results of a Typical Session

This implementation of the remote database access facility has been demonstrated to work in a number of test examples, the transcript of which is given in Appendix C. In these examples queries formulated at a UA at one EAN site *dsrg.ubc.cdn* were submitted to an INGRES relational database residing on another EAN site *cs.ubc.cdn*. Although these results realise the objectives of a first implementation, the functionality of both the UA and the DBUA would have to be improved before they could be used as reliably as the existing EAN UA.

4.6. Response Time The fact that the two EAN sites used in these examples are part of the same local network, meant that the time taken for queries to be processed was very short (30 - 50 seconds).

However, since a message-based remote database access facility is primarily intended for use over long-haul networks where the delay would be considerably longer, this low response time is not considered to be a significant system performance measure. However, since a message-based design is not intended for fast-response applications, a longer response time could not be regarded as a drawback of the system.

4.7. Drawbacks of a Hybrid UA

Deviating from the CCITT X.400 model in combining the functions of a remote database UA and an interpersonal UA into one entity has resulted in a somewhat "quick

and dirty" implementation. Results do not conform to the Database Access Protocol and queries and results are stored in interpersonal message format. If a fully functional remote database access facility had to be implemented, a cleaner implementation could only be achieved by ensuring that the remote database UA was kept as a separate entity.

4.8. Evaluating the Design

As illustrated in Figure 5.1, the problem of networkwide database access can be characterised by identifying the three dimensions of heterogeneity in which DBMSs differ from each other.

There are differences in the way a DBMS is invoked (dependent on the host operating system), differences in the nature of the user/DBMS interaction (dependent on the nature of the data model and query language used), and differences in the location of

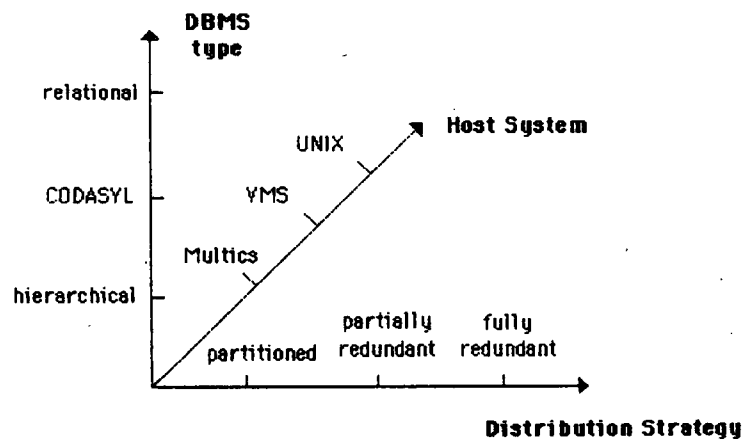

Figure 5.1 Dimensions of variations in DBMSs [WANG83]

data in the DBMS (dependent on the data distribution strategy used by the DBMS).

As was discussed in Chapter 2, these variations have given rise to a number of connection-based systems whose sophistication varies according to the degree with which these differences are overcome. Less sophisticated systems, such as iNet, only deal with differences in system dependent invocation procedures while more sophisticated systems, such as Multibase, accommodate different data models, query languages and distributed data. Similarly, the degree of sophistication with which a message-based facility provides access to remote databases can be measured by its ability to accommodate the differences mentioned above.

The design proposed here accommodates both differences in DBMS invocation as well as differences in Data Models and Query Languages. In providing this generalisation, however, a certain onus is placed on users to be aware of the peculiarities of the DBMS they wish to access.

In the case of remote DBMS invocation, users are required to know which of the three optional fields must be included in the QueryHead. This is an inevitable result of the intractibility of designing a DBUA capable of generalised DBMS invocation, since for the most part a DBMS is invoked differently each time it is used (e.g. many DBMS require the name of a particular database as part of the system command which invokes it).

As far as allowing different Data Models and Query languages is concerned, neither the UA nor the DBUA assume responsibility for ensuring the syntactic or semantic validity of queries and so users have to be entirely familiar with the query language of DBMSs they access. Here it must be stated that a more ambitious design would relieve users of this requirement. It would have been possible to allow users to submit queries in

a canonical query language, such as is used in systems like MULTIBASE. The Query-Body would then be structured according to this canonical form instead of plaintext and each DBUA would be responsible for translating the structured canonical query into the query language of the recipient DBMS. It would also, of course, be responsible for the inverse translation so that results are always returned to originating UAs in a uniform format.

This approach was not adopted because it would have resulted in a far more extensive implementation (both the UA and the DBUA would have to enlarged substantially, while detracting from the primary goal of demonstrating the feasibility of a message-based design.

In addition, the present design does *not* address the problem of providing access to distributed DBMSs (i.e. partitioned databases managed by multiple controllers and located at multiple sites). It was felt that the complexity of dealing with distributed DBMSs would unduly complicate the protocol design as well as the overall system architecture. For instance, one would have to contend with the problem of deciding which controller (or controllers) should be sent queries, as well as collating partial results. These, and many other issues, relating to distributed databases are not fully understood and are still the subject of active research. Trying to deal with these issues within the context of this thesis would result in a design of far lesser simplicity and ease of implementation.

However the fact that a distributed message system is used as the communication medium suggests that it might be suitable for distributed database applications as well. This would be a subject for further investigation.

4.9. A More Extensive Implementation

It would have been desirable to implement the design over more host operating systems and DBMSs as well as over a wider range of EAN sites, but this was not possible due to the lack of system resources.

4.10. Conclusions

We have given the motivation and presented a design for a message-based remote database access facility which is well suited to request-response type applications. The design complies with the CCITT X.400 international recommendations for message handling systems.

A first implementation of the design has demonstrated that it is both feasible and simple to implement on top of an existing message system. Refining the Database Access Protocol to allow for data model/query language independence, and exploring the potential of a message-based design in distributed database applications are both areas for further study.

References

[CCITT82] CCITT Provisional Recommendation X.200, "Reference Model of Open Systems Interconnection for CCITT Applications", Sept 1982.

[CCITT83a] CCITT Final Draft Recommendation X.400, "Message Handling Systems:System Model-Service Elements", Dec 1983.

[CCITT83b] CCITT Final Draft Recommendation X.409, "Message Handling Systems:Presentation Transfer Syntax and Notation", Dec 1983.
[CCITT83c] CCITT Final Draft Recommendation X.411, "Message Handling Systems:Message Transfer Layer", Dec 1983.

[CCITT83d] CCITT Final Draft Recommendation X.420, "Message Handling Systems:Interpersonal Messaging User Agent Layer", Dec 1983.

[CHAPI82] A.L. Chapin, "Connectionless Data Transmission", *Comp. Comm. Rev.*, Vol 12, no. 2, April 1982, pp 21-61.

[CHAPI83] A.L. Chapin, "Connections and Connectionless Data Transmission", *Proc. IEEE*, Vol 71, no. 12, Dec 1983, pp 1365-1371.

[CUNNI83] I. Cunningham, J. Raiswell, "Gateway to Online Information Services", *Telesis*, vol 1. 1983, pp 2-7.

[DADOU84] N. Dadoun, "EAN User Agent Implementation Notes", *CPSC 530 Term Paper*, Univ. of Brit. Col., April 1984.

[DAY83] J. Day, H. Zimmermann, "The OSI Reference Model", *Proc. IEEE*, Vol 71, no. 12, Dec 1983, pp 1334-1340.

[GILMO84] P.C. Gilmore, "EAN and CDNnet: A Progress Report", *Information Pamphlet*, Dept. of Comp. Sc., Univ. of Brit. Col., Nov 1984.

[LISAN80] S. Lisanti, "The Online Search", *Byte*, Dec 1980, pp 215-230.

[NEUFE83] G.W. Neufeld, "EAN : A Distributed Message Handling System", *Proc CIPS 1983 Conf.*, Ottawa, May 1983, pp. 144-149.

[NEUFE84] G.W. Neufeld, "The EAN Distributed Message System User Manual", Distributed Systems Research Group, Dept. of Comp. Sc., Univ. of Brit. Col., 1984.

[SADOW84] E.R. Sadowski, "The Efficacy of a Store-and-Forward File Transfer System", *M.Sc. Thesis*, Univ. of British Columbia, April 1984.

[SMITH81] J.M. Smith et al., "Multibase - Integrating Heterogeneous Distributed Database Systems", *AFIPS Conf. Proc.*, 1981 National Computer Conference, Vol 50, pp 487-99.

[STONE76] M. Stonebraker et al., "The Design and Implementation of INGRES", *ACM Trans. Database Syst., Vol 1, no 3, Sept 1976, pp 189-222.*

[SOLOS84] A.G. Solosky, "iNet 2000 : Gateway to the Information Marketplace", CIPS Session 84 Proc., Calgary, May 1984, pp 477-78.

[VALLE84] J. Vallee, *Computer Message Systems*, McGraw-Hill, (1984), 163 pp.

[WANG83] P.S.-C. Wang, S.R. Kimbleton, " An Application Protocol for Networkwide Database Access", *J. of Telecommunication Networks*, Vol 2, no 3, Fall 1983, pp 285-294.

Appendix A

CCITT X.420 Specification

CCITT X.420 specifies the service elements and protocol structure of the Interpersonal User Agent Layer. The protocol used to encode an interpersonal message is called *P2* and is given here according to the notation specified by CCITT X.409 [CCITT409].

The essential elements of P2 state that a message, in an interpersonal messaging application, is either a *user message* or a *status report* giving status information about a previously submitted message.

As can be seen, X.420 provides substantial flexibility in the type of data that can be included in the body of a message.

The converted EncodedInformationTypes component conveys the new EncodedInformationTypes if conversion took place on the IM-UAPDU.

```
P2 DEFINITIONS ::=
BEGIN

-- P2 makes use of types defined in the following modules:
--      P1:   X.411, Section 3.4
--      P3:   X.411, Section 4.3
--      SFD:  this Recommendation, Section 5
--      Sa:   S.a, Section 4.1

UAPDU                       ::= CHOICE {
                                    [0] IMPLICIT IM-UAPDU,
                                    [1] IMPLICIT SR-UAPDU}

-- IP-message UAPDU

IM-UAPDU                    ::= SEQUENCE {Heading, Body}

-- heading

Heading                     ::= SET {
    IPMessageId,
    originator            [0] IMPLICIT ORDescriptor OPTIONAL,
    authorizingUsers      [1] IMPLICIT SEQUENCE OF ORDescriptor OPTIONAL,
      -- only if not the originator
    primaryRecipients     [2] IMPLICIT SEQUENCE OF Recipient OPTIONAL,
    copyRecipients        [3] IMPLICIT SEQUENCE OF Recipient OPTIONAL,
    blindCopyRecipients   [4] IMPLICIT SEQUENCE OF Recipient OPTIONAL,
    inReplyTo             [5] IMPLICIT IPMessageId OPTIONAL,
      -- omitted if not in reply to a previous message
    obsoletes             [6] IMPLICIT SEQUENCE OF IPMessageId OPTIONAL,
    crossReferences       [7] IMPLICIT SEQUENCE OF IPMessageId OPTIONAL,
    subject               [8] CHOICE {S61String} OPTIONAL,
    expiryDate            [9] IMPLICIT Time OPTIONAL, -- if omitted, expiry date is never
    replyBy              [10] IMPLICIT Time OPTIONAL,
    replyToUsers         [11] IMPLICIT SEQUENCE OF ORDescriptor OPTIONAL,
      -- each O/R descriptor must contain an O/R name
    importance           [12] IMPLICIT INTEGER {low(0), normal(1), high(2)} DEFAULT normal,
    sensitivity          [13] IMPLICIT INTEGER {personal(1), private(2), companyConfidential(3)}
                                OPTIONAL,
    autoforwarded        [14] IMPLICIT BOOLEAN DEFAULT FALSE
      -- indicates that the forwarded message body part(s) were autoforwarded -- }

IPMessageId                 ::= [APPLICATION 11] IMPLICIT SET {
                                    ORName OPTIONAL,
                                    PrintableString}

ORName                      ::= P1.ORName

-- P2 definitions to be continued
```

Figure 3/X.420. Formal Definition of IM-UAPDU (Part 1 of 3)

```
-- P2 definitions continued

ORDescriptor        ::= SET {-- at least one of the first two members must be present
                            ORName OPTIONAL,
                            freeformName [0] IMPLICIT S61String OPTIONAL,
                            telephoneNumber [1] IMPLICIT PrintableString OPTIONAL}

Recipient           ::= SET {
                            [0] IMPLICIT ORDescriptor,
                            reportRequest [1] IMPLICIT BIT STRING {
                                receiptNotification(0),
                                nonReceiptNotification(1),
                                returnIPMessage(2)} DEFAULT {},
                                -- if requested, the O/R descriptor must contain
                                --   an O/R name
                            replyRequest [2] IMPLICIT BOOLEAN DEFAULT FALSE
                                -- if true, the O/R descriptor must contain
                                --   an O/R name --}

-- body

Body                ::= SEQUENCE OF BodyPart

BodyPart            ::= CHOICE {
                            [0]  IMPLICIT IA5Text
                            [1]  IMPLICIT TLX,
                            [2]  IMPLICIT Voice,
                            [3]  IMPLICIT G3Fax,
                            [4]  IMPLICIT TIF0,
                            [5]  IMPLICIT TTX,
                            [6]  IMPLICIT Videotex,
                            [7]  NationallyDefined,
                            [8]  IMPLICIT Encrypted,
                            [9]  IMPLICIT ForwardedIPMessage,
                            [10] IMPLICIT SFD,
                            [11] IMPLICIT T1F1}

-- body part types

IA5Text             ::= SEQUENCE {
                            SET {repertoire [0] IMPLICIT INTEGER {ia5(5), ita2(2)}
                                DEFAULT ia5
                                -- additional members of this Set
                                -- are a possible future extension --},
                            IA5String}

TLX                 ::= for further study

Voice               ::= SEQUENCE {
                            SET, -- members are for further study
                            BIT STRING}

-- P2 definitions to be continued
```

Figure 4/X.420. Formal Definition of IM-UAPDU (Part 2 of 3)

```
-- P2 definitions continued

G3Fax                   ::= SEQUENCE {
                                SET {
                                    numberOfPages [0] IMPLICIT INTEGER OPTIONAL,
                                    [1] IMPLICIT P1.G3NonBasicParams OPTIONAL},
                                SEQUENCE OF BIT STRING}

TIF0                    ::= SaDocument

SaDocument              ::= SEQUENCE OF Sa.ProtocolElement

TTX                     ::= SEQUENCE {
                                SET {
                                    numberOfPages [0] IMPLICIT INTEGER OPTIONAL,
                                    telexCompatible [1] IMPLICIT BOOLEAN DEFAULT FALSE,
                                    [2] IMPLICIT P1.TeletexNonBasicParams OPTIONAL},
                                SEQUENCE OF S61String}

Videotex                ::= SEQUENCE {
                                SET, -- members are for further study
                                S100String}

NationallyDefined       ::= ANY

Encrypted               ::= SEQUENCE {
                                SET, -- members are for further study
                                BIT STRING}

ForwardedIPMessage      ::= SEQUENCE {
                                SET {
                                    delivery [0] IMPLICIT Time OPTIONAL,
                                    [1] IMPLICIT DeliveryInformation OPTIONAL},
                                IM-UAPDU}

DeliveryInformation     ::= P3. DeliverEnvelope
                            -- This merely reuses a data type definition, and does not
                            -- imply that the information was ever carried in P3.

SFD                     ::= SFD.Document

-- note that SFD and SaDocument use the same space of application-wide tags
--  which is different from that used for other MHS protocols

TIF1                    ::= SaDocument

-- P2 definitions to be continued
```

Figure 5/X.420. Formal Definition of IM-UAPDU (Part 3 of 3)

```
-- P2 definitions continued

-- IPM-status-report UAPDU

SR-UAPDU                ::= SET {
                                [0] CHOICE {
                                    nonReceipt      [0] IMPLICIT NonReceiptInformation,
                                    receipt         [1] IMPLICIT ReceiptInformation},
                                reported            IPMessageId,
                                actualRecipient     [1] IMPLICIT ORDescriptor OPTIONAL,
                                intendedRecipient   [2] IMPLICIT ORDescriptor OPTIONAL,
                                    -- only present if not actual recipient
                                    -- the O/R descriptor must contain an O/R name
                                converted           P1.EncodedInformationTypes OPTIONAL}

NonReceiptInformation   ::= SET {
                                reason [0] IMPLICIT
                                    INTEGER {uaeInitiatedDiscard(0), autoForwarded(1)},
                                nonReceiptQualifier [1] IMPLICIT
                                    INTEGER {expired(0), obsoleted(1),
                                        subscriptionTerminated(2)} OPTIONAL,
                                comments [2] IMPLICIT PrintableString OPTIONAL,
                                    -- on auto-forward
                                returned [3] IMPLICIT IM-UAPDU OPTIONAL}

ReceiptInformation      ::= SET {
                                receipt [0] IMPLICIT Time,
                                typeOfReceipt [1] IMPLICIT INTEGER {
                                    explicit(0),automatic(1)} DEFAULT explicit,
                                [2] IMPLICIT P1.SupplementaryInformation OPTIONAL}

END -- of P2 definitions
```

Figure 6/X.420. Formal Definition of SR-UAPDU

Appendix B

User and Database User Agent Installation

The following are instructions for the installation of the UA and the DBUA on UNIX 4.2Bsd systems.

The User Agent

The User Agent consists of existing EAN software routines (located in the EAN source directory - usually ~ *ean/src*) and a number of other routines used to provide the additional *QUERY* command. These routines are presently located in ~ *koorland/dba/dba* on *ubc-cs* along with a *Makefile*.

In order to compile the user agent, assuming pathnames in *#include* statements are all accurate, the command *make myua* should be issued. On successful compilation, the file *myua* will exist and be similar to the EAN UA except for the additional QUERY command.

The Database User Agent

All the routines for the DBUA are located in the same directory and presently support an INGRES DBMS. If additional DBMS are added to the DBUA then the code has to be modified to accommodate each new DBMS (the sections of the code that have to be modified are documented in the body of the code).

The DBUA also uses a number of existing EAN routines which have to be present before it can be compiled. To compile the DBUA, *make dbagent*.

Each DBUA must be associated with a O/R Name (Address) *dbagent@site* and its local MTA must know the name of the DBUA program. This can be arranged by the EAN system administrator.

Appendix C

Test Examples

The following is an annotated transcript of some examples of the remote database access facility in use. In these examples a UA with the NUA *koorland@dsrg.ubc.cdn* is used to submit queries to an INGRES relational DBMS with the NUA *ingres@cs.ubc.cdn* and under the control of the DBUA *dbagent@cs.ubc.cdn.*

In the following, annotations are given in Italic, user input in Roman face, and system responses in Bold face. Familiarity with the use of the EAN interpersonal UA would be helpful in following these examples.

EXAMPLE 1 - formulate a query to ingres@cs.ubc.cdn and invoke it with command "/user etc ..." which specifies the "demo" database

> query ingres@cs.ubc.cdn command="/user/ingres/bin/ingres -s demo"

Enter query (".." to end -- "break" to abort)

print parts
\go
\quit

****** query sent**

accept the result from the MTS

> accept
Accepting new messages: .

inbox:
3 NU dbagent@cs.ubc.cdn Apr 14 85 query to : ingres@cs.ubc.cdn
print the result

> print 3

Message inbox:3 - Unread
From: <dbagent@cs.ubc.cdn>
To: <koorland@dsrg.ubc.cdn>
Subject: query to : ingres@cs.ubc.cdn

a copy of the original query is returned with the result

QUERY SUBMITTED

print parts
\go
\quit

==========

QUERY RESULT

parts relation

pnum	pname	color	weight	qoh
10	byte-soap	clear	0	143
1	central processor	pink	10	1
11	card reader	gray	327	0
2	memory	gray	20	32
12	card punch	gray	427	0
3	disk drive	black	685	2
13	paper tape reader	black	107	0
4	tape drive	black	450	4
14	paper tape punch	black	147	0
5	tapes	gray	1	250
6	line printer	yellow	578	3
7	l-p paper	white	15	95
8	terminals	blue	19	15
9	terminal paper	white	2	350

EXAMPLE 2 - use EAN alias facility and command abbreviation to avoid tedious typing and submit the same query

> set alias ingres=ingres@cs.ubc.cdn
> query ingres c="/user/ingres/bin/ingres -s demo"

Enter query (".” to end -- “break” to abort)

```
print parts
\go
\quit


**** query sent
```

accept and print result

```
> accept
Accepting new messages: .

inbox:
  4 NU   dbagent@cs.ubc.cdn      Apr 14 85 query to : ingres@cs.ubc.cdn
> print 4
Message inbox:4 -  Unread
From:    <dbagent@cs.ubc.cdn>
To:      <koorland@dsrg.ubc.cdn>
Subject: query to : ingres@cs.ubc.cdn


QUERY SUBMITTED

print parts
\go
\quit

==============================
```

QUERY RESULT

parts relation

pnum	pname	color	weight	qoh
10	byte-soap	clear	0	143
1	central processor	pink	10	1
11	card reader	gray	327	0
2	memory	gray	20	32
12	card punch	gray	427	0
3	disk drive	black	685	2
13	paper tape reader	black	107	0
4	tape drive	black	450	4
14	paper tape punch	black	147	0
5	tapes	gray	1	250
6	line printer	yellow	578	3
7	l-p paper	white	15	95
8	terminals	blue	19	15
9	terminal paper	white	2	350

EXAMPLE 3 - use batch query facility

> query ingres c="/user/ingres/bin/ingres -s demo"

Enter query (".” to end -- “break” to abort)

#include batch_query

****** query sent**

> accept
Accepting new messages: .

inbox:
5 NU dbagent@cs.ubc.cdn Apr 14 85 query to : ingres@cs.ubc.cdn
> print 5
Message inbox:5 - Unread
From: <dbagent@cs.ubc.cdn>
To: <koorland@dsrg.ubc.cdn>
Subject: query to : ingres@cs.ubc.cdn

QUERY SUBMITTED

print parts
\go
\quit

==========

QUERY RESULT

parts relation

pnum	pname	color	weight	qoh
10	byte-soap	clear	0	143
1	central processor	pink	10	1
11	card reader	gray	327	0
2	memory	gray	20	32
12	card punch	gray	427	0
3	disk drive	black	685	2
13	paper tape reader	black	107	0
4	tape drive	black	450	4
14	paper tape punch	black	147	0
5	tapes	gray	1	250
6	line printer	yellow	578	3
7	l-p paper	white	15	95
8	terminals	blue	19	15
9	terminal paper	white	2	350

EXAMPLE 4 - query a non-existant DBMS

> query spires@cs.ubc.cdn

Enter query (“.” to end -- “break” to abort)

**** query sent

> accept
Accepting new messages: .

inbox:
6 NU dbagent@cs.ubc.cdn Apr 14 85 query to : spires@cs.ubc.cdn

```
> 6
Message inbox:6 - Unread
From:   <dbagent@cs.ubc.cdn>
To:     <koorland@dsrg.ubc.cdn>
Subject: query to : spires@cs.ubc.cdn

DBMS spires@cs.ubc.cdn does not exist
```

EXAMPLE 5 - Do not specify the correct optional QueryHead fields

```
> query ingres

Enter query (“.” to end -- “break” to abort)

print parts
\g
\q


**** query sent

> accept
Accepting new messages: .

inbox:
  7  NU   dbagent@cs.ubc.cdn      Apr 14 85 query to : ingres@cs.ubc.cdn
> 7
Message inbox:7 - Unread
From:   <dbagent@cs.ubc.cdn>
To:     <koorland@dsrg.ubc.cdn>
Subject: query to : ingres@cs.ubc.cdn

Insufficient or Invalid authorisation given
```

EXAMPLE 6 - specify the invocation command incorrectly

```
> query ingres c=“dfhdfhjdfshjkfdjfhj”

Enter query (“.” to end -- “break” to abort)

print parts
\go
\quit
```

****** query sent**

> accept
Accepting new messages: .

inbox:
8 NU dbagent@cs.ubc.cdn Apr 14 85 query to : ingres@cs.ubc.cdn
> 8
Message inbox:8 - Unread
From: <dbagent@cs.ubc.cdn>
To: <koorland@dsrg.ubc.cdn>
Subject: query to : ingres@cs.ubc.cdn

DBMS ingres@cs.ubc.cdn cannot be invoked

EXAMPLE 7 - use incorrect query language in QueryBody

> query ingres c="/user/ingres/bin/ingres -s demo"

Enter query (".” to end -- “break” to abort)

this is not a valid query in
EQUEL, the INGRES query language
\go
\quit

****** query sent** *the UA doesn't know the query is incorrect*

> accept
Accepting new messages: .

inbox:
10 NU dbagent@cs.ubc.cdn Apr 14 85 query to : ingres@cs.ubc.cdn
> 10
Message inbox:10 - Unread
From: <dbagent@cs.ubc.cdn>
To: <koorland@dsrg.ubc.cdn>
Subject: query to : ingres@cs.ubc.cdn

QUERY SUBMITTED

this is not a valid query in
EQUEL, the INGRES query language
\go

\quit

==========

according to the UA this result is correct and the user
is responsible for interpreting the error message from INGRES
QUERY RESULT

2600: syntax error on line 1
last symbol read was: this

> quit