分散資料庫系統的設計和建立

國科會計畫研究報告

計畫主持人：柯志昇

研究助理：林慶良・陳省隆
王秋鳳・潘佑安・張嘉祥・黃玉玲
鄭聖慶・張品煥・呂建春・劉龍騶・鄭王駿
陳志堅・朱正忠・廖久英・凌雲穎・康錫頂

計畫編號：NSC 71-0404-E001-02

執行期限：民國七十年十二月一日至七十一年十一月卅日

日期：中華民國七十二年一月
內容

一、 計畫說明 ................................................................. 1
二、 The Design and Implementation of a Distributed Database System ......................................................... 6
三、 Design and Implementation of a Heterogeneous Computer Network .......................................................... 24
四、 Enhanced Interprocess Communication Mechanisms for UNIX ... 71
五、 A Distributed Database Management System Over UNIX Environments ...................................................... 76
六、 Simulation and Automatic Implementation of Communication Protocols ................................................... 98
七、 Optimization of Query Processing in Distributed Database Systems ........................................................... 105
分散式資料庫系統的設計和建立

柯志昇

中央研究院資訊科學研究所

本研究主要在建立一分散式資料庫系統並做相關之理論研究
全部計畫工作共分下列四項：

(1) 連結三架計算機並製作必要之軟體程式，構成一區域網路。
(2) 設計並製作一關係模式資料庫管理系統之軟體程式。
(3) 設計分散式資料庫系統所需經之各種管理程式，包括協同控制
和查詢處理。
(4) 研究協同控制、查詢處理、和通訊協定之理論。

本研究計畫分為兩期，第一期自民國六十九年十二月一日至
七十年十一月三十日止（見第一期研究報告）。第二期自七十年
十二月一日至七十年十一月三十日止。本報告主要在敘述第二
期研究的過程和成果。全部系統的概要請見附件一。
計算機網路之設計與建立

1. 參與人員

柯志昇、林慶良、陳省隆、潘佑安、呂建春、康錫頂、劉龍駿

2. 研究目的

連結一架 PDP-11/70，兩架 PDP-11/23，一架 Z80 做處理機並製作必須之軟體程式以構成一區域型計算機網路。

3. 理論基礎與背景

計算機網路可以便利資訊的共用並提供訊息的快速交流。然而網路的效益和穩定性完全依賴網路通訊協定來控制。本研究乃在探討通訊協定的軟體製作方法和技術。

4. 研究內容

(1) 通訊協定的訂定。
(2) 通訊軟體程式的製作。
(3) 作業系統與通信系統之配合。

5. 研究方法

在 PDP-11/70 上裝設 RSX-11/M 作業系統，在 PDP-11/23 上裝設 UNIX 作業系統，並分別以組合語言和 C 語言製作有關之通信軟體程式，其中包括修改 RSX-11/M 和 UNIX。

6. 研究成果

（見附件二和附件三）
分散式關係模式資料庫系統之設計與建立

1. 參與人員
   柯志昇、林慶良、王秋鳳、張嘉祥、黃玉玲、鄭王駿、
   陳志堅、朱正忠、廖久英、凌雲穎。

2. 研究目的
   設計並製作一分散式關係模式資料庫管理系統。

3. 理論基礎與背景
   關係模式資料庫俱有簡單與資料獨立的特性，並具數學理論
   基礎，識別處理較易。本研究以關係模式為基礎，探討資料
   庫管理系統設計上的一些典型問題，包括資料選取方法、協
   同控制、識別處理等等。

4. 研究內容
   (1) 資料定義語言和資料處理語言之制定與處理。
   (2) 識別語言之處理。
   (3) 交易之處理與控制。
   (4) 檢索法之製作。

5. 研究方法
   以C-一語言在UNIX作業系統上製作軟體程式。

6. 研究成果
   （見附件四）
通信協定之訂定與驗証

1. 參與人員
   柯志昇、鄭聖慶

2. 研究目的
   研究計算機通信協定的訂定與驗証之方法。

3. 理論基礎與背景
   計算機通信協定係用來控制通信同步和正確傳輸的法則。通信協定的完整性對通信的效率有絕對性的影響。由於通信本身事件發生的複雜性，使得通信協定很難正確地訂定。研究一套訂定與驗証通信協定的方法刻不容緩。

4. 研究內容
   (1) 研究通信的基本特性和法則。
   (2) 研究派翠網理論在通信協定上之應用。
   (3) 製作一通信協定模擬器。

5. 研究方法
   以C－語言製作一以派翠網為基礎之通信協定模擬器做為驗証通信協定之工具。

6. 研究成果
   （見附件五）
分散式資料庫系統的詢問處理

1. 參與人員

柯志昇、張品煥。

2. 研究目的

研究分散式資料庫系統的詢問處理方法，期能使回訊時間最短或處理效率最高。

3. 理論基礎與背景

分散式資料庫系統的詢問處理具有兩大特性。其一為分散在不同地方的資料可以局部並行處理，其二為資料必須在兩地間傳輸。這兩大特性影響到詢問處理的方法和效果。本研究即在探討較好的詢問處理策略。

4. 研究內容

1. 選擇率與半聯技術的應用。
2. 多地儲存相同資料的詢問處理策略。
3. 多詢問同時處理的策略。

5. 研究方法

本研究目前純屬理論性之研究。主要方法是研習國外先進的成果報告，並加以延伸之。

6. 研究成果

（見附件六）。
The Design and Implementation of a Distributed Database System

Jyh-Sheng Ke, Ching-Liang Lin, Hsing-Lung Chen
Chiou-Feng Wang, Chia-Hsung Chang, Yow-An Pan
Chien-Chun Lu and Shyi-Ting Kang
Institute of Information Science
Academia Sinica, R.O.C.

SUMMARY

In September 1980, we initiated a project, called DDBS, to implement a distributed data base system at Institute of Information Science, Academia Sinica. This paper describes what has been learnt from the experience of implementing a local area network and a distributed data base system. Solutions for those typical issues arising from the distribution of data and processing power, include the connection of different operating systems, distributed query processing, and concurrency control, are presented.

This paper is the first of a series of companion papers on DDBS([LIN 83], [CHEN83],[CHAN83]).

Key words Distributed data base, Local area network, Query processing, Concurrency control
INTRODUCTION

Institute of Information Science, Academia Sinica, is a newly founded research organization. At this institute we have three minicomputers, one PDP-11/70, two PDP-11/23's, and a number of micro-level computers. In September 1980, we initiated the project DDBS to implement a distributed database system on top of these mini- and micro-computers. Figure 1 illustrates the network architecture of the installed system. The operating system installed in the PDP-11/70 is RSX-11/M, and the operating system installed in the two PDP-11/23's is UNIX. The project has been scheduled into three progressive stages. The first stage is to implement a relational data base management system using C language under UNIX. The second stage is to implement a local area network by connecting the three minicomputers and a Z80-based front end processor with relevant layered protocol structures. The third stage is to implement a distributed database system basing on the results of the earlier stages by tackling all issues arising from the distribution of data, including the problems of concurrency control and query processing.

This paper describes what we have learnt from the experience of converging the two diversified technologies, namely the computer network and the data base. The implementation is definitely not new research but a piece of bread and butter work designed to provide a distributed data base system for a variety of users in which the network connections and data distribution are made to be transparent to the end user.
Figure 1: Local Area Network Architecture
CONNECTING UNIX and RSX-11/M

As we mentioned before, the PDP-11/70 has been installed with RSX-11/M operating system, and the two PDP-11/23's have been installed with UNIX operating system. The physical connection among these three minicomputers are RS232C serial lines and will be replaced by a high-speed cable and its associated interface in the near future. The protocol hierarchy in the network is shown in Figure 2. DDCMP is a simulation program of DEC standard link level protocol handler which ensures correct frame transmissions. NSP is responsible for the flow control message exchange between two processes in different computers. FTP is responsible for the remote file transfer. RJE allows terminal users to use the resources of remote computer environments.

The unidirectional 'pipeline' interprocess communication mechanism of the UNIX system is not suitable for implementing the protocol hierarchy of the computer network. We have enhanced the interprocess communication capability of the UNIX system with the message-based 'port' mechanism and some relevant system calls for synchronizing asynchronous events[LU 82].

The main implementation problem when connecting RSX-11/M to a network is where to site the protocol handler. It can either be installed as part of the terminal I/O driver; or it can be run as a system task communicating with the network through some interface to the terminal I/O driver. Due to the complexity of the original RSX-11/M terminal I/O driver, it was decided that the protocol handler must be run as a background task and communicating with the network through an add-in virtual terminal driver as an interface to the terminal I/O driver. Figure 3 shows
Figure 2: Network Protocol Hierarchy
Figure 3: Standard RSX-11/M Terminal I/O

Figure 4: Modified RSX-11/M Terminal I/O
the terminal I/O driver function of the standard RSX-11/M system. Figure 4 shows the modified RSX-11/M in which three additional software modules have been incorporated to function as a virtual terminal protocol handler via which multiple terminals can be connected to the PDP-11/70 through a Z80-based concentrator. Essentially the integration of VTMON and VTDRV is the protocol handler at NSP level. All I/O requests from the application tasks will be sent to the Virtual Terminal Driver, VTDRV, which in turn will queue an AST (Asynchronous System Trap) routine in the TCB (Task Control Block) of the Virtual Terminal Monitor task, VTMON, so that VTMON will reversely issue an O/I request to VTDRV and send a message to the DDCMP task upon the execution of the AST routine. The DDCMP task will then issue relevant physical I/O requests to TTYDRV accordingly. Figure 5 shows the relationships between the application program, high-level protocol handlers (i.e. FTP and RJE), and the low-level protocol handlers. Detail of these protocol programs can be found in [CHEN83].
Figure 5: Protocol Handlers under RSX-11/M
DISTRIBUTED RELATIONAL DATA BASES

There has been a lot of debate over the inefficiency of the relational model data bases. However, since our major concerns in selecting data model were in choosing a data model with high degree data independency and strong mathematical foundation for query processing, we were convinced that the relational model is the only model which can meet our goal.

Implementation of the DBMS has been strongly influenced by INGRES[STON76] and System R[ASTR76]. The DBMS includes two subsystems, namely Relational Interface System(RIS) and Physical Storage System(PSS)(see Figure 6). The RIS provides high level, data independent facilities for data definition, manipulation, and retrieval. The data definition facilities of the RIS allow the DBA to define the data at conceptual level as well as internal level. They also allow a variety of external views to be defined upon common underlying data to provide different users with different views of the data base. The external views of the common data also facilitates some authorization checks and privacy control. One of the important features of the DDL facility is that each attribute in the relational database may be optionally associated with an integrity control routine which will be triggered to execute upon the corresponding update operation. The data manipulation facilities of the RIS allow data to be inserted, deleted, modified, manipulated, and retrieved by using a set of high level relational operators which exempt the users from knowing the internal storage structures of the stored data. Moreover, a high level, nonprocedural query language has also been implemented to provide a friendly interface for the novice
Figure 6: Architecture of the DBMS
users. Each query statement is translated into a transaction constructed by a sequence of relational algebra operators with some heuristic optimization techniques.

The PSS manages disk storage allocation, indexed file access. B-tree structure has been used for indexing secondary keys of each relation. The disk space contains a set of files each of which is constructed by a set of fixed-size pages. For the purpose of saving space and speeding up access, we employ the technique of mapping logical space (what the relational operator sees) into physical space (what the data is really stored). A page map table has been maintained to map logical page number into physical page number. This mapping technique can reduce the problems of hashing dispersed records and is particularly useful in overflow control. The PSS also supports concurrency control by using shared-read-lock and exclusive-write-lock with lock algorithms none strategy.

For the purpose of providing end users with a consistent database and making the user have an illusion that they are using a large unified database, a transaction monitor has been implemented on top of the database management system. The transaction monitor receives transactions from the end users and serializes the DML statements of multiple transactions to achieve high-degree parallelism of executing database operations.

Due to the address space limitation of PDP-11/23(64K bytes), the DBMS under UNIX has been structured into several related processes. The process structure of the DBMS is shown in Figure 7. The function of each process is self-explanatory and will not be recounted here. Routines for indexed access,
Figure 7: Process Structure of DBMS
integrity control, and error recovery will be called by these processes at proper time. More detail of the DBMS can be found in [CHAN83].

QUERY PROCESSING

As we pointed out before, one of the objects of the DDBS is to make the distribution of data and processing power be transparent to the user. In DDBS, we achieve this goal by providing the user with a high level query language which in turn is supported by the "VIEW" definition facility.

A VIEW is defined by the DBA and is a subschema of a set of related data items in the problem domain. To the user, each VIEW represents the relationships among those data items that the user is allowed to access. The set of data items defined in a VIEW may be distributed over several sites, however, the locations of stored sites and the data access paths are transparent to the user. In the following we will use an example to illustrate the technique of query processing in the DDBS.

Assume that three sites, A, B, C are stored with data as described below:

Site A: BOOK(BID,BNAME,BAUTHOR,PUBLISHER,PUB_DATE,CATEGORY, STATUS,L4,S4)

JOURNAL(JID,JNAME,VOL,NO,DATE,JPUBLISHER,JCATEGORY, L4,S4)

PROCEEDING(PID,PNAME,CONFERENCE,DATE,LOCATION, L4,S4)

Site B: PAPER(PNO,TITLE,PAUTHOR,JPID,DATE,PAGE)

Site C: SUPPLIER(S4,SNAME,SCITY,SPHONE)

LIBRARY(L4,LCITY,ADDRESS,DIRECTOR,LPHONE)
Basing on this knowledge of data distribution, the DBA can define a VIEW, named VIEW_LIB, as follows:

```
DEFINE
FROM BOOK.A, JOURNAL.A, PROCEEDING.A, PAPER.B,
      SUPPLIER.C, LIBRARY.C
INTO VIEW_LIB(*)
WHERE BOOK.S# = SUPPLIER.S#
     AND JOURNAL.S# = SUPPLIER.S#
     AND PROCEEDING.S# = SUPPLIER.S#
     AND BOOK.L# = LIBRARY.L#
     AND JOURNAL.L# = LIBRARY.L#
     AND PROCEEDING.L# = LIBRARY.L#
     AND PAPER.JPID = JOURNAL.JID
     AND PAPER.JPID = PROCEEDING.PID
```

In response to this VIEW definition a semantic graph (see Figure 8) will be generated and can be used as the foundation of query translation.

In this example, we have included all data items in VIEW_LIB. However, this is not always the case. The DBA may define a VIEW which contains only a subset of data items depending on the security degree of the user.

Now, suppose that a user at site C issues a query, to find the director of the library which owns a journal that contains the paper "On Modeling Relational Databases", by specifying the query as follows:

```
RETRIEVE FROM VIEW_LIB INTO R(L#, DIRECTOR)
WHERE TITLE='On Modeling Relational Databases'
     AND PID=JPID
```

Basing on the semantic graph in Figure 8, the access path for this query can be found as shown in Figure 9.

From the access path shown in Figure 9, a transaction for retrieving data to answer the user's query can be generated as shown below:
Figure 8: Semantic Graph
Figure 9: Access Path

DIRECTOR=

LIBRARY

SUPPLIER

BOOK

JOURNAL

PROCEEDING

PID

JPID

PAPER

TITLE='On Modeling Relational Databases'
BEGIN TRAN
B: RETRIEVE FROM PAPER INTO R1(PID)
    WHERE TITLE='On Modeling Realitonal Databases'
A: RETRIEVE FROM PROCEEDING INTO R2(PID,L#)
B: MOVE B.R1 TO C.R1
A: MOVE A.R2 TO C.R2
C: JOIN (R1,R2) INTO R3 WHERE JPID=PID
C: JOIN (R3;LIBRARY) INTO R(L#;DIRECTOR) WHERE
    R3.L#=LIBRARY.L#
END_TRAN

The ordering of DML statements has been optimized by first reducting
the relations with restriction and projection operations. The
reduced data will be moved to the query site, and join operations
will then be performed if necessary. The technique of using
semi-join[BERN81] has not been employed in this implementation.
However, we have developed a technique, called graph
projection[CHAN82], which hopefully can minimize the overhead of
semi-join operations. We plan to implement this technique in DDBS
query processing in the near future.

CONCLUSION

This paper describes the design and implementation of a
heterogeneous local area network and a relational data base
management system. On top of the computer network and the DBMS, a
distributed data base system has been built up. The current computer network consists of one PDP-11/70 with RSX-11/M and two
PDP-11/23's with UNIX. Two VAX's are planned to join the network in the near future. The highly portable communication software
will make the required effort of this extension become trivial.
To construct a computer network under the existing operating systems(i.e. UNIX and RSX-11/M), several modifications of the
operating systems have been made to enhance the capability of
interprocess communication.

A library application system is now run to test the reliability and performance of the DDBS. The statistical results of this test will be presented in a further report.

REFERENCES


Design and Implementation of a Heterogeneous Computer Network

This report describes the design and implementation of a heterogeneous computer network. The network configuration is shown below. In the network, two different operating systems, namely RSX-11/M and UNIX, have been installed in PDP-11/70 and PDP-11/23, respectively. To couple the communication software with the existing operating systems, several modifications of the operating systems have been made. In UNIX, we modify the process scheduling strategy and enhance the interprocess communication capability by incorporating the 'message exchange port' mechanism. In RSX-11/M, we achieve the multiplexing and communication control capability by incorporating the 'virtual terminal' mechanism. In the following, this documentation has been divided into two parts. Part I describes UNIX network communication software design detail. Part II describes the same of the RSX-11/M. Since that we are keeping on developing our communication software, the correctness of this documentation is subject to change, if necessary.
Figure 1. Network Configuration
Part I UNIX Network Communication Software

1. UNET

1.1 UNET Function

UNET is a family of software programs which extend UNIX operating system so that it can take the role of computer communication. It supports the following basic requirements in the network environment:

- Process-to-Process Communication:
  Programs on one host can exchange data with programs on the other host.

- Remote File Access:
  UNET provides both terminal users and programs with capability of accessing files that reside on remote nodes. Remote file access facilities allow users to perform the following operations:
  * Transferring files between nodes
  * Manipulating files residing at a remote node.

- Remote Terminal Facilities:
  UNET allows a local terminal to be connected logically to a remote node, which then executes the commands entered from that terminal.

1.2 UNET Architecture

UNET is based on a set of network protocols, each of which is designed to fulfill some specific functions within the network.
Figure 1 indicates the functional layers within UNET. Each layer defines a distinct set of functions as well as rules for implementing those functions.

```

USER LAYER
NETWORK APPLICATION LAYER
NETWORK SERVICE LAYER
DATA LINK LAYER
PHYSICAL LINK LAYER

```

Figure 1: UNET Architecture

The functions and purpose of each layer are described below.

The Physical Link layer encompasses the software device driver for each communication device plus the hardware itself. The hardware includes interface devices, modems, and the communication line.

DDCMP is the software emulator of the DECNET standard data link control protocol handler. It handles the physical level link control and error recovery within UNET. It has been designed to operate with existing hardware interfaces over full-duplex facilities, and is independent of communication device characteristics.

NSP handles network management functions within UNET, including the routing of messages between systems, and the routing of messages within any given system. This protocol handler
allows process-to-process communication within UNET. Modules in the user layer and file transfer layer can use the facilities provided by NSP.

The Network Application layer defines the functions used by the user layer. This layer includes two sublayers, i.e. File Transfer Layer (FTP) and Remote Job Entry Layer (RJE). The functions currently operating with this layer are remote file access, file transfer, and the remote login capability.

FTP supports remote file access and file transfer facilities.

RJE allows a local terminal to be connected logically to a remote host.

The User layer encompasses user-written programs and services that access the network. It is the highest layer in the network architecture.

1.2.1 DDCMP

DDCMP, a data link protocol, handles all the problems associated with getting messages from one end of a link to the other, in proper order, and with no errors. It has the following characteristics:

- using CRC-16 for error detection
- using retransmission for error recovery

1.2.2 NSP
NSP (Network Services Protocol) is the interface through which two programs in different sites can talk with each other. It is also called the "logical link" protocol, and has the following characteristics:

- Providing process-to-process communication

NSP is the heart of UNET, it takes the role of UNET's user interface and provides the power associated with its process-to-process communication capability.

- Routing messages between systems

Whenever a program sends a message, the NSP software at the same node puts the message in an envelope and sends it out onto the appropriate network link.

- Routing messages within system

Once a message arrives at the proper node in the network, the NSP software uses the address specified by the sending software to determine which program is to receive the message. NSP strips off the envelope and hands the message to the receiving program.

- Creating "virtual circuit"

NSP's logical link facility allows processes in various computers to communicate with each other in a flexible manner, while still limiting the overhead information carried in each message. Two programs wish to communicate with each other should first go through an initial 'handshaking' procedure. The result
of this 'handshake' is the creation of a virtual circuit, which is a logical connection that allows these programs to exchange data messages without regarding to the traffic on the physical link connecting the two systems. Such logical connections between two programs may be created or destroyed dynamically, by allowing either end initiating the creation and/or destruction.

managing network

NSP keeps track of nodes and links which are running, in order to properly route messages and manage virtual circuits.

1.2.3 FTP

FTP is directly analogous to the file control system in UNIX operating system. FTP operates at user level; indeed, to NSP, it appears to be just another user program. It interfaces to the local file system, user programs, and NSP.

FTP provides the means for terminal users and the programs on one machine to use devices and files resident on other machines. Essentially, it allows implementation of a network-wide file system. FTP has the following characteristics:

.providing remote file transfer operations.

FTP supports user programs to open, close, read, write, and unlink files over any node in the network.

.providing remote device operations.

Since UNIX operating system treats devices as files, FTP also supports the facility to access remote devices for the user.
Figure 1.2 shows the relationships among these protocol handlers.

Fig 1.2: UNET Protocols

Implementation of these protocol handlers is achieved by enhancing the interprocess communication capability of the UNIX system through the incorporation of a 'port' mechanism. Details of this enhancement can be found in [15].
2. User Interfaces

Presumably, there are three ways in which users can employ UNET features:

.Interprogram communications:

This allows implementation of simple and complex multiprocessor systems in a wide variety of situations.

.File operations:

A terminal user can create, read, write, delete and rename files in local and remote nodes. These capabilities provide tools for implementing distributed data base systems.

.Virtual terminal access:

Terminals that are physically connected to a remote node can be treated as if they were physically connected to the local host.

2.1 The interface for interprogram communications

Interprogram communication is the most important feature common to all UNET implementations. Interprogram communication allows programs to create "virtual circuits", which are full-duplex virtual data paths. Programs have the capability to create virtual circuit, transmit and receive data over the created virtual circuit and destroy virtual circuit.

The following system calls implement the process-to-process communication.
CONECT(dnode, dname, sname, pname): request to create a virtual circuit, where 'dnode' is the destination node of the virtual circuit, 'dname' is the destination process name in the destination node, 'sname' is the source process name in the source node, 'pname' is the portname in the source node.

SEND(cid, buf, count): send data over a virtual circuit, where 'cid' is the return value of CONECT operation.

RECEIVE(cid, buf): receive data over a virtual circuit.

DISCONECT(cid): disconnect a virtual circuit.

2.2 File operations

FTP defines a set of commands that control the execution of remote file access and outlines procedures to accomplish file operations.

The following calls are implemented to enable remote file transfer.

ROpen(dnode, username, filename, mode): open or create a remote file

RREAD(fid, buf, bytes): read remote file

RWRITE(fid, buf, bytes): write remote file

RCLOSE(fid): close remote file

2.4 Virtual terminal support

Virtual terminal support is a feature that allows a terminal, which is physically connected to one system, operate
logically as if it were connected to another. The concept of virtual terminal is transparent to the user's application program.

3. UNET Design

UNET design considers the network and its corresponding communication facilities as a number of distinct layers. Within each layer the functions have been logically separated into modular units. This structured design allows for easy of subsetting and modifications within our network implementation. At each layer the functional modules are independent and may be implemented as either individual tasks or subroutines within a task. The layered approach allows modification and/or replacement of a specific protocol without disturbing other layers in the protocol structure.

3.1 Layered protocols

Data to be sent from a user program in one node to a user program in another is enveloped within two or three layers of protocol control information to form a message. The highest layer is the end-to-end or user interprocess communication mechanism layer. The next layer, NSP, must multiplex messages into a single queue or data stream for transmission over a physical data link. This multiplexing usually consists of the addition of header routing information to facilitate demultiplexing at the receiving end. The data stream is then passed to a data link protocol which is concerned with error detection and correct sequencing of messages being sent between adjacent nodes. This
A layer communicates with a driver layer protocol that interfaces with the hardware of the system.

Figure 3-1 illustrates the structured design of the protocol structure. Figure 3-2 shows the data flow through the protocol structure.

Figure 3-1: Structured Design of Protocol Structure
Figure 3.2 Enveloping User Data in Protocol Layers
4. DDCMP

DDCMP's function is to create an error free data link between two stations which exchange data over a medium that has some possibility of corrupting the data being sent. It is concerned with error detection, correction, and message frame sequencing.

4.1 DDCMP functions

DDCMP provides the following functions within a network system:

1. Correct sequencing and integrity of the data, even over noisy error producing channels.
2. Efficient line utilization by simultaneous use of duplex lines in both directions.
3. Efficient transmission of any bit sequence (data transparency).
4. Error detection using a 16-bit CRC-16 polynomial.
5. Ability to achieve frame synchronization and to protect itself from erroneous interpretation of user data as control information.

4.2 DDCMP Messages

User information is exchanged over DDCMP links between the data source and data sink through numbered data frames. Protocol control information is exchanged via unnumbered control frames, or by "piggy backing" on the headers of data frames. Each frame carries a sequence number (modulo 8) assuring correct frame sequencing. The receiver always acknowledges the correct
receipt of frames by piggybacking numbered frames or sending an
unnumbered frame. For efficiency, an acknowledgment of frame
number implies acknowledgment of all frames sent up to and in-
cluding that frame. Routing information to or from processes is
applied by the user and contained within the text portion of
numbered messages.

2.1 Numbered Data Messages

Numbered messages carry data over DDCMP links. The format
of a numbered message is shown as follows.

<table>
<thead>
<tr>
<th>HEADER field</th>
<th>DATA field</th>
<th>CRC16</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SOH</td>
<td>COUNT</td>
<td>Receive no.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following table gives details on each field:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOH</td>
<td>The numbered data message identifier.</td>
</tr>
<tr>
<td>COUNT</td>
<td>The byte count field specifies the number of bytes in the DATA field.</td>
</tr>
<tr>
<td>Receive No.</td>
<td>The response number used to acknowledge correctly received message.</td>
</tr>
<tr>
<td>Transmit No.</td>
<td>The transmit number, which denotes the number of this data message.</td>
</tr>
<tr>
<td>CRC16 (first)</td>
<td>The block check on the numbered message header.</td>
</tr>
<tr>
<td>DATA field</td>
<td>The DATA field consisting of COUNT 8-bit quantities. This field is totally transparent to the protocol and has no restrictions on any bit patterns. The only requirement is that it should be a multiple of 8 bits.</td>
</tr>
<tr>
<td>CRC16(second)</td>
<td>The block check on the DATA field. Computed on DATA only.</td>
</tr>
</tbody>
</table>
12.2 Unnumbered Control Messages

Unnumbered messages carry control information, transmission status, and restart information between the processes implementing the protocol.

Control messages have the following form:

| ENQ | TYPE | SUBTYPE | RESPONSE | NUMBER | CRC16 |

- **ENQ**: The unnumbered control message identifier.
- **TYPE**: The control message type.
- **SUBTYPE**: The SUBTYPE field provides additional information for some message types.
- **RESPONSE**: The control message response field used to pass information field from the numbered message receiver to the numbered message sender.
- **NUMBER**: The control message number field used to pass information field from the sender to receiver.
- **CRC16**: The block check on the control message.

The control message TYPE field can be one of the following codes:

**ACK** **⇒** Acknowledge Message:

The ACK message is used to acknowledge correct receipt of numbered messages. It conveys the same information as RECEIVE field in numbered messages and would be used when acknowledgments are required and no piggybacking is available.

**NAK** **⇒** Negative Acknowledge Message:

The NAK message is used to pass error message information from the receiver to the sender. The NAK message also includes
the same information as the ACK message, thus serving two functions: acknowledging previously received messages and notifying the sender of some error condition. The receipt of a NAK usually results in the retransmission of messages that have been corrupted by the transmission medium.

.REP—Reply to Message Number

The REP message is used to request received message status from the receiver. It is usually sent when the sender has transmitted data messages and has not received a reply within a timeout period. It resynchronizes the transmit and receive message numbers of the sender and receiver respectively. The response to a REP is either an ACK or NAK depending on the received message number at the receiver.

.START—Start Message

The START message is used to establish initial contact and synchronization on a DDCMP link. It is used only on link startup. It operates with the start acknowledge message STACK described below.

.STACK—Start Acknowledge Message

The STACK message is returned in response to a START when the station has completed initialization. The proper response to a STACK is an ACK.

.DISCONNECT—Disconnect Message

The DISC message is used to destroy the DDCMP link.
4.3 Operation

Both stations can send a STRT message to startup the connection. The proper response to a STRT is a start acknowledge STACK message. The startup procedure is illustrated by the diagram below (Figure 4.1).

**Event**

**Action**

- HALT
  - request link setup
  - destroy link or DISC received
  - send DISC
  - request link setup
  - send STRT & set timer

- ISTRT
  - timeout
  - send STRT set timer

- ASTRT
  - STRT received or timeout
  - send STACK set timer

- RUN
  - STACK received
  - send ACK stop timer

- STACK received
  - send ACK stop timer

**Figure 4.1: DDCMP state diagram**
After DDCMP enters run-state, the control of messages exchanging follows the rules illustrated below (Figure 4\#2).

<table>
<thead>
<tr>
<th>EVENT</th>
<th>ACTION</th>
</tr>
</thead>
</table>
| MSG \( r \) received | send MSG if there is data to be sent  
or ACK if there is no data to be sent  
or NAK if this msg is error |
| ACK \( r \) received | send MSG if there are data to be sent  
or nothing |
| NAK \( r \) received | resend data message |
| REP \( r \) received | send ACK or NAK depending the checking  
of the sequence \( r \) |
| STACK received | send ACK |
| timerout | send REP |
| | send MSG if there is data to be sent |
Figure 4.2 illustrates an example of messages exchanging between two DDCMP stations.

Station A

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| STRT   | ********>
|        |        |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| ACK #0 | ********>
|        |        |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| (ACK#0)| ********>
| MSG#1  |        |
|        |        |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| (ACK#2)| ********>
| MSG#2  |        |
|        |        |

(Receive on #3)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| MSG#3  | ********>
|        |        |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| NAK#3  | ********>
|        |        |

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(ACK#3)</td>
<td></td>
</tr>
</tbody>
</table>
| MSG#3  | ********>
|        |        |

(Receive NAK)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG#3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(ACK#2)</td>
<td></td>
</tr>
<tr>
<td>MSG#4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Receive ACK)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG#3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MSG#4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Receive REP)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>REP#4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.2: DDCMP Operation
5. NSP

NSP defines the rules that govern the creation and operation of virtual circuits which are in turn constructed by a set of logical links. In addition to creating and operating individual logical links, NSP modules enable multiple logical links to share a single communications line. Within the UNET network, each communications line carries data packets belonging to one logical link intermingled with data packets belonging to other logical links. NSP modules format outgoing logical link data for transmission by the communication hardware. Conversely, these modules separate incoming data into logical link streams and deliver the data to the appropriate local programs.

5.1 NSP function

NSP provides the following functions within a network system:

1. Services to the dialogue level:
   Create a conversation path (virtual circuit).
   Transmit data over a sequence of logical links.
   Receive data over a sequence of logical links.
   Destroy logical links and virtual circuits.

2. Functions for operation of link:
   Multiplex logical links into physical links.
   Control traffic over logical links.

3. Functions for network supervision:
   Detect node/link failures and maintain routing paths.

4. Functions for network maintenance:
   Maintain error logs.
5.2 NSP Messages

5.2.1 Data Messages

| DATA/| Dnode | Snode | Dsocket | Ssocket | Seq# | ACK# | DATA |
| DATA1 |       |       |         |         |      |      |      |

DATA0 : The data message identifier indicates that more data messages will transmit.
DATA1 : The data message identifier indicating this is the last data message of the segment
Dnode : Destination node
Snode : Source node
Seq# : Sequence number
ACK# : Acknowledge number
DATA : DATA field

5.2.2 Control Messages

Acknowledgement Message

| ACK | Dnode | Snode | Dsocket | Ssocket | Seq# | ACK# |

ACK : Acknowledge Message

Logical Link Control Messages

| TYPE | Dnode | Snode | Dsocket | Ssocket | 0 | 0 |

CR : Connect Request message
CC : Connect confirm message
Dsocket : Destination socket
Ssocket : Source socket

| TYPE | Dnode | Snode | Dsocket | Ssocket | Reason | 0 |

DR : Disconnect Request message
DC : Disconnect Confirm message
Reason : Disconnection reason

Node Initialization Control Message

| NI | Dnode | Snode | Func | SIZE | LINKS | OS |

NI : Node initialization message
Func : Supporting functions
SIZE : Max. block size
LINKS : Max. logical links
OS : the OS version
Figure 5.1: Logical Link State Diagram
5.3 Logical Link Operation

Logical links define the end points of a conversation (dialogue) and in no way imply the technique used for message transmission between the two end nodes. Figure 5.1 shows the state diagram of the logical link control.

5.3.1 Creation of Logical links

NSP modules create the logical link on behalf of two cooperating programs. Two programs wish to communicate over a logical link must firstly follow the handshaking procedures regardless of either program's location in the network. The two programs must have a preliminary dialogue, with the NSP modules acting as intermediaries, before exchanging data. The preliminary dialogue is sometimes called a handshake where each program recognizes and agrees to be linked to the other program.

During the handshake sequence, each program specifies a link identifier to the local NSP modules. If the connection is successful, the program uses the link identifier to address all messages to be sent over the link. In turn, the NSP modules cooperate to assign their own link address that defines the link uniquely to each of them. At either end of the link, each NSP module associates the NSP-level addresses with the local program's link identifier. Figure 5.2 illustrates the interrelationship of the programs' identifiers and the NSP modules' addresses.
Figure 5.2: Interrelationship of Link Identifiers and Addresses

5.3.2 Messages Transfer over Logical Links

After creating a logical link, the NSP modules begin to orchestrate data transfers between the connected programs. When a program hands over a unit of data for transmission, the local NSP modules may send the data in one data packet, or it may divide the data into segments and send each segment in a separate packet. In the later case, the remote NSP modules reassemble the segments before delivering the data to the remote program.

5.3.3 Disconnection of Logical Links

Logical links may be terminated upon (1) request from the user, (2) failure of the user process, or (3) failure of a communication link.
<table>
<thead>
<tr>
<th>Node A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program 'AA' NSP</td>
</tr>
</tbody>
</table>

**Request Connection**

<table>
<thead>
<tr>
<th>CR</th>
<th>Request Connection</th>
</tr>
</thead>
</table>

**Transmit Data**

<table>
<thead>
<tr>
<th>MSG #1</th>
<th>Receive 1st Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSG #2</th>
<th>Receive 2nd Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Receive 1st Data**

<table>
<thead>
<tr>
<th>ACK #1</th>
<th>Request Disconnect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Request Disconnect**

<table>
<thead>
<tr>
<th>DR</th>
<th>Request Disconnect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DC</th>
<th></th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Node B</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSP Program 'BB'</td>
</tr>
</tbody>
</table>

Figure 5.3: Inter-Program Communications

5.3.3 **Frame Acknowledge and Flow Control**

The NSP modules that manage both ends of a logical link guarantee that:

- All transmitted data is received.
- All received data is given to the target program in the proper sequence.

To guarantee proper frame sequencing, an NSP module numbers the frames transmitted over the link. The receiving NSP module, using the transmit numbers for identification, must acknowledge the delivery of the frames. If a frame is not acknowledged within a certain period of time, the sending NSP module...
retransmits it.

Window mechanism has been employed to help the data flow control. Since the message sequence number can be from 0 through 7, the window size is defined to be 4.

6. FTP

FTP provides the facilities of remote file access and file transfer. Remote file access requires the cooperation of two network programs: a program in one node issues a remote file access request, and, in the target node, a FTP receives the file access request and transfer it to a spawned process to handle. Before the first program can issue the remote access request, the two process establish a logical link between them.

6.1 FTP Functions

FTP provides the following services in a UNET environment:

1. Open or Create a file on a remote node.
2. Transfer a file between two nodes.
3. Perform file maintenance functions (delete, rename).

6.2 Operation

In the context of remote file access, the accessing program can be a user-written program, a UNET utility, or a system call, depending on the application. The remote FTP spawns a child process to complete connection initiated by accessing program and convert the incoming request into calls to the file system. The spawned process then sends the resulting file data back to the
accessing program.

Like process-to-process communication, remote file access require a handshaking sequence at the beginning of the operation. Not only does UNET set up a logical link between the accessing program and the remote spawned process, it also exchanges initial FTP messages to prepare for the file operation to be performed over the link.

The open.remote.file.call issued by the accessing program passes the following information to remote FTP:

- Access control information

- A file specification and Characteristics of the file to be accessed

The access control information logs the program into the remote system, enabling it to access any of the remote system's resources. The accessing program must have an account to a guest account on the remote system to log in successfully. The access control information consists of:

- A user identification name

- A password associated with the user identification.

Logging in successfully does not always guarantee that the accessing program's requested file operations can be admitted. In UNIX operating system, each file has a corresponding protection code that determines the type of access allowed. The user identification specified by the accessing program determines the right to access that file.
The file specification identifies the remote file to be accessed. Because the remote file system actually carries out the requested file operation, the programmer must know how the file is identified by users in its local node.

The file characteristics defines the file to be accessed in the following modes: read, write, or both.

7. RJE

A virtual terminal driver is added to the system such that RJE allows a user to connect his physical terminal to a virtual terminal on any other host in the network.

7.1 Operation

A user who wants to login remote node can keyin the command

@rex nodename

After this command is entered with a valid remote node's name, the local process requests connection with the remote RJE. When the logical link is set up, remote RJE spawns a private server to execute the "/bin/login" task. After the right user name and password have received by the private server, the login procedure is completed. Then the user can keyin any command that the remote system can execute, and the remote private server receives the command over the logical link, executes and sends the result to the local terminal. The current implementation of remote login procedure is a two-step operation. However, we are going to change it into a one-step operation in the near future.
Node A

User

Request remote login
(Keyin %rex B)

setup a logical link

CR

Spawn a private server
to execute "/bin/login"

request user name

$NAME

check user privilege and request password

$PASSWD

check user password and return prompt

% command

Execute "command" and answer the result

Receive result

Request logout

(signal)

Private server request disconnect and exit

Return to local login

Node B

NSP

RJE

---

Figure 7-1: Remote Login Access
8. Implementation of UNET

Figure 8-1: Structure of the UNET Implementation
8.1 Implementation of DDCMP

Data Flow

Control Flow

from NSP
    ↓
    port
    ↓
    from port
    → frame
    sender
    ↓ make
    frame
    ↓< frame

Figure 8.2: Modules of DDCMP
§2 Implementation of NSP

Figure 8.3: Modules of NSP
8.3 Implementation of RJE

from RJE          from user          from NSP
  ↓             ↓                ↓
  |****|       |****|           |****|
  |port|       |port|           |port|
  |****|             |****|
  ↓             ↓                ↓
  read VT     write VT       pwrite
    A          A              (logout)
    .          .                .
    .          .                .
    .          .                .
    .          .                .
    .          .                .
    .          .                .
  login       (fork)         private       (fork)       RJE
  server        server       server
  (execute     (exit)         (exit)
   command)
  .          .                .
  .          .                .
  .          .                .
  .          .                .
  close      set              open
  VT         VT               VT

Figure 8.4: Modules of RJE (VT : virtual terminal)
8.4 Implementation of FTP

from FTP ↓ from user ↓ from NSP
| ****** | ****** | ****** |
| port   | port   | port   |
| ****** | ****** | ****** |

pwrite (file or command receiver) pwrite (connect concern) pwrite (file or user server)

pwrite (file or user server)

pwrite (file or user server)

pwrite (file or user server)

pwrite (file or user server)

pwrite (file or user server)

pwrite (file or user server)

file (fork) check (fork) FTP

server (exit) server (exit)

open/create/ UNIX
read/write/ FILE
close/unlink SYSTEM

Figure 8.5: Modules of FTP
Part II  RSX-11/M Network Communication Software

This part describes the implementation idiosyncracies of the communication software in RSX-11/M operating system. The protocol hierarchy of this implementation is the same as UNET and has been described in Part I. Hence, we will only describe the facilities that have been incorporated into the original RSX-11/M to enhance its communication capability. Essentially, the mechanism we employed is centralized on the concept of virtual terminal, through which multiplexing technique can be achieved. Detail of this implementation can be found in [14].

1. Virtual Terminal Concept

In RSX-11/M, each job (a log-in user) is usually initiated by a user at a physical terminal. Many tasks may run concurrently with input and output through the terminal device. For each physical terminal there is a control block of memory in the operating system containing information about the physical terminal and including I/O request buffers as the linkage between the physical terminal(user) and the job(running task).

In computer communication software, it is desirable to allow a task in the system to be initiated by another task, most commonly the communication control task, instead of a user at a physical terminal. Since a controlling task cannot use a physical terminal in the way a user can, some means must be provided in
the executive for the controlling task to send input to and accept output from the job being controlled. Unfortunately this facility is not provided by the original RSX/11/M operating system. Hence, we introduce a pseudo device to provide this capability via the Virtual Terminal (VT). The virtual terminal is a simulated terminal and is not defined by hardware. Like hardware-oriented terminals, each VT has a control block of memory associated with it. This block of memory is used by the VT in the same manner as a hardware-oriented terminal uses its block memory. Figure 1 shows the parallelism between a hardware-oriented terminal and a software-oriented virtual terminal.

Figure 1: Parallelism between PT and VT
The controlling task uses the VT in the same way as the user uses a physical terminal device. It initiates the VT, input characters to and wait for output from the VT, and closes the VT using the appropriate programmed facility. The task controlled by the controlling task performs I/O to the VT as though the VT were a physical terminal.

A controlled task may get into a loop and not accept any input from its associated VT device, therefore, it is not possible for the controlling task to simply rely on busy-waiting for activities in the controlled task. A controlling task may wish to drive more than one controlled task, and be able to respond to any of these tasks; therefore, the controlling task cannot stationarily wait for any particular VT. For these two reasons, the VT differs from other devices in that it is never in an I/O wait state. Synchronization between the controlling task and the controlled task is accomplished by event flags and asynchronous system traps (AST) provided by the RSX-11M operating system.

Event flags are a means by which tasks recognize specific events. In requesting a system operation such as an I/O transfer, a task may associate an event flag with the desired I/O operation. When the I/O complete event occurs, the executive will set the specified flag. Each event flag has a corresponding unique Event Flag Number (EFN). A task can set, clear, and test event flags to check whether specific event occurs or not. A task may also wait on more than one event flag to monitor many outstanding asynchronous activities.
Asynchronous System Traps (AST) detect events that occur asynchronously to the task's execution. That is, task has no direct control over the precise time that the event (the trap) may occur. For example, the completion of an I/O transfer may cause an AST to occur. The primary purpose of an AST is to inform the task that a certain event has occurred. As soon as the task has serviced the event, it can return to the interrupted code. ASTs can be used as an alternative to event flags or the two can be used together. Users can specify the same AST routine for several external activities, each with a different event flag. When the executive passes control to the AST routine, the event flag can determine the action required.

The controlling task in the host may create several VT channels for the remote users. The input and output AST routines are specified commonly for each VT device. When an I/O request is issued from the controlled task, the AST routine will be entered. In AST routine, it checks which device the I/O request comes from and sets a specific event flag associated to that channel. Thus, the main program can monitor all the event flags to handle I/O conditions of VT devices. The synchronization between the controlling task and the controlled tasks is achieved by this mechanism.
2. Virtual Terminal Device

The RSX-11M I/O system is structured as a hierarchy in Figure 2. At the top of the hierarchy are file control service (FCS) and record management services (RMS), which provide device-independent access to devices included in the system. The QIO directive is the lowest level of task I/O. The QIO directive allows direct control over devices that are connected to a system and that has an I/O driver. The I/O services provided by the executive consist of QIO directive processing, and a collection of subroutines used by drivers to obtain I/O requests, and facilitate interrupt handling. The actual control of the device is performed by the driver.

There are four data structures important to the driver; all I/O operations are affected by these data structures. They are:

(1) the Device Control Block (DCB).
(2) the Unit Control Block (UCB).
(3) the Status Control Block (SCB).
(4) the I/O packet.

Once a device has been created, the related data structures must be established. When a VT channel is opened, the data structures for a VT unit Device Control Block (DCB) and Unit Control Block (UCB) are created and linked into the device list assigned by the lowest available VT unit number. Three AST routines may be specified, the input AST, the output AST, and the attach and detach AST.
Figure 2: NIX-11/N I/O flow control
The controlling task can service each offspring (controlled) task's input or output request with a corresponding output or input request to the correct virtual terminal unit. For example, suppose that a controlled task has been activated as an offspring task of the remote log-in control task with a TI: of VT2:, then

1. Offspring issues an IO.RVB or IO.RLB to TI: for its input line. The virtual terminal driver queues the request internally and effects an AST in the remote log-in control task at the virtual address "IAST" with the unit number 2 and the byte count from offspring's I/O request on the stack.

2. In the controlling task's AST routine, an event flag associated with that channel is declared. The remote log-in control task detects this event, retrieves an input line for offspring from the physical I/O port, and specifies this line in a QIO directive to a LUN assigned to VT2: with an IO.WVB or IO.WLB.

3. The virtual terminal driver reads the line from the control task's buffer, writes the line to offspring's buffer and then signal I/O completion for both I/O requests. Similarly, if offspring needs to print a message, it does so with an IO.WVB or IO.WLB to TI:.

4. In the controlling task's output AST routine, a specific event flag is also set. For the declared event flag, the control task issues an IO.RVB or IO.RLB to retrieve the line via the virtual terminal driver. Then, the control task may output this line to the physical I/O port with a user ID in front of the message.
For each remote user's command line the control task may use
spawn directive queuing a command line to a specific task for
execution, and establishing the task's TI: as a previously opened
virtual terminal unit. The task being spawned is a command line
interpreter called Monitor Console Routine(MCR), it allows users
to operate and control the RSX-11M system. The I/O operation
process of the virtual terminal device has been illustrated in
Figure 3.

\[\text{Figure 3: Modified RSX-11M Terminal I/O}\]
3. Virtual Terminal Driver

The virtual terminal driver is primarily intended for facilitating a parent task to simulate terminal I/O for an offspring task activated with the spawn directive. This simulation takes place via a virtual terminal unit whose unique data structures (DCB and UCB) are dynamically created while the virtual terminal unit is being opened. Only one common SCB is used for all virtual terminal units.

The virtual terminal driver employs the UC.QUE bit (in UCB U.CTL word) to receive all I/O packets directly from the QIO directive. Offspring read and write requests are queued to the common SCB and dequeued one by one (FIFO), based on the attachment of the device and the presence of other requests. Each time an offspring read or write is dequeued, the parent task receives an AST at its input or output AST entry point. The parent task is then expected to issue a complementary write or read request to simulate a terminal I/O transfer.

Only Offspring tasks may attach the virtual terminal unit. Parent task's requests are always serviced in spite of the attachment of the virtual terminal unit.

The driver initiator entry point is entered from the QIO directive whenever a parent or offspring request is issued. Parent I/O requests are always serviced immediately, normally resulting in a block I/O transfer of data and the completion of both the parent request and the corresponding offspring request.
Offspring requests are initially queued and then dequeued one by one. An AST is declared in the parent task whenever an offspring read or write is dequeued.
REFERENCES


(4) "RMX80 User's Guide", Intel Corp.

(5) "REX80 Primer", System and Software Corp.


(8) D.W. Davies "Computer networks and their protocols"


Proceedings of International Computer Symposium 1982 (Vol. II)

ENHANCED INTERPROCESS COMMUNICATION MECHANISMS FOR UNIX

Chien-Chou Lin
Department of Electrical Engineering
National Taiwan University
Taipei, R. O. C.

Ching-Liang Lin and Jyh-Sheng Kuo
Institute of Information Science
Academia Sinica
Nankang, Taipei, R. O. C.

ABSTRACT

This paper presents the design and implementation of an interprocess communication and synchronization mechanism for UNIX operating system. In particular, the message-based synchronization technique has been employed to enhance the UNIX's capability of handling asynchronous event.

1. INTRODUCTION

In the UNIX operating system, which is widely known and used as a general purpose time sharing system on a single computer, the interprocess communication channel is created by a 'pipe' call. 'Pipe' is dedicated to the synchronized processing and the redirection of standard I/O. For example, the command 'cat datafile | pr' means that the output of program 'cat' is piped as the input of program 'pr', illustrated as Fig 1.1.

Related processes may also be synchronized by using 'wait' primitive in the sense that a parent process waits for the offspring process to terminate. No explicit waiting primitive is provided to synchronize the asynchronous events. The only way for a process to monitor asynchronous events is by busy-waiting.

However, in many applications, e.g. computer network and database management, the behaviors among processes are not restricted to be synchronous. Only 'pipe' and 'wait' are difficult and not sufficient to implement an asynchronous communication channel and moreover, generalize to a network environment.

In this paper, we present the design of an enhanced interprocess communication facility for implementing a local network server including file transfer, remote command execution, etc., and for further research on distributed data processing including distributed database system, and distributed programming language. Presumably, the following goals have been achieved:

(1) Provide a message-based interprocess communication mechanism for UNIX.
(2) Provide a synchronizing primitive to listen one or more asynchronous events, and allow more than one signal being caught simultaneously.

(1) and (2) are accomplished by introducing 'port' and 'listen' primitives.

This paper is organized as follows: related works are reviewed in section 2: 'port' and 'listen' primitives are described in section 3: section 4 explains message and software interrupts; data structures are illustrated in section 5: finally, in section 6 the conclusion is given.

2. OVERVIEW OF RELATED WORKS

In this section, several other systems, which provide facilities similar to some of those provided by our enhanced interprocess communication facilities, are reviewed and compared.

Carnegie-Mellon's Interprocess Communication Facility for UNIX

An interprocess communication facility has been implemented at Carnegie-Mellon University for VAX/UNIX version 7.2.

The facilities for interprocess communication are supported through one or more ports. A port is created by a process through the AllocPort System call. The owner process may destroy the port by performing a DeallocPort system call. Send and Receive calls are used for adding and removing the messages to/from the port. For flexibility and efficiency consideration, the messages have various types which can either be system-defined or user-defined. Process synchronization, which is directly explicit signaling, is supported by DefineSignal, GetSignal and ClearSignal calls. The interprocess communication message system provides a reasonable model for communication under the assumption of the error and guarantees notification of error conditions as long as a communication dependency exist between two processes.

UNIX-Based COCANET

A local network COCANET based on the UNIX has been developed at the University of California, Berkeley, to support research on distributed databases systems and provide shared access to resources.

* This work was partially supported by National Science Council, Republic of China, under the contract NSC71-0404-E001-02.
UNIX has been extended to allow existing programs to access remote resources with no source program changes. Programs may access remote files, have a remote working directory, execute programs, and communicate with remote computers using the standard UNIX interprocess communication mechanisms (i.e. pipes).

Purdue's UNIX Based Local Network

A UNIX-based local computer network with load balancing has been developed at Purdue University2,4.

The network provides the capabilities of virtual terminal access, remote process execution, file transfer, load balancing, and user programmable network interface. The unique feature of this network is the load balancing strategy. Load balancing is a technique to increase resource utilization so that network performance can be enhanced. Only a few of specially written system programs at the high level protocol, together with a library of host-to-host network functions are needed to establish the local computer network. No major changes of the UNIX operating system were made to establish the network functions.

A Distributed UNIX System Based On A Virtual Circuit Switch

A distributed UNIX system based on a virtual circuit switch has been built by G. N. R. Luderer1.

The new operating system consists of two components: the S-UNIX subsystem provides a complete UNIX process environment enhanced by remote files access; the P-UNIX subsystem is specialized to offer remote files service. The two operating system components allow us to construct configurations with different degrees of file system access and terminal-to-terminal communication. A new special UNIX interprocess communication mechanism is the 'fifo', which provides a communication between unrelated processes by associating a new special file type with a file name. Terminal-to-terminal communication and interprocess pipe are not implemented, but remote 'fifo' can to some degree replace the latter.

3: INTERPROCESS COMMUNICATION

In this section, we present an enhanced interprocess communication mechanism for UNIX. The enhanced interprocess communication (EIPC) employs the notion of 'port', an extended data structure and buffer area for message exchange. A port is a protected kernel object into which messages can be placed by processes and from which messages can be removed.

A process may communicate with the other process directly by creating one or more ports, or indirectly through a transporting process.

Two examples of process-to-process communication are shown below.

Example 3.1: communication between user process and spooler process

In Fig 3.1, spooler process creates a port `/spool` to be shared with user processes. Whenever a user process queues the spooler, a frame is put into the port to request a file to be printed out.

![Fig 3.1](image)

Example 3.2: communication between local process, network server, and remote process

In Fig 3.2, 'port 1' is shared by user process and the network server, with which message may be sent from local user processes to remote processes through the network server.

![Fig 3.2](image)

Example 3.2 shows that many senders and one receiver may share a port, but there are, in general, many senders and many receivers, i.e. a many-to-many relationship among processes.

Conceptually, each port has two access modes: sending access and receiving access. A process with sending access mode can put messages into a port, and with receiving access mode can remove messages of his own out from a port. Only the port's owner can specify the access capability.

In reality, a port is shared by two or more processes. These processes may send or receive messages to/from a port simultaneously. This results in the concurrency control problem. To solve this problem, a mutually exclusive locking protocol using semaphores has been implemented to serialized data accesses, in the sense that only one sending process can access a port at a time, but receiving processes are allowed to access the port simultaneously to increase the degree of concurrency.

Manipulation of a port is through a set of system services, of which we explain the meaning in the following subsections.
Proceedings of International Computer Symposium 1982 (Vol. II)

Create and Connect

A port can be created by system call

\[ \text{pfd} = \text{palloc} \text{(name, mode)} \]

where

name is global and given by the calling process; mode is used to specify the capability of the port; pfd is a port descriptor, i.e. a local port id given by the system.

Similarly, a port can be connected by the system call

\[ \text{pconnect} \text{(name, mode)} \]

Send and Receive

Messages can be sent to ports by the system call

\[ \text{pwrite} \text{(pfd, pfd, address, count)} \]

It means that sending messages to port 'pfd' from location 'address' of 'count' bytes to the receiver 'pfd'.

Messages are buffered on the port area with a header added by the system. Similarly, processes may request to receive message from ports by the system call

\[ \text{pread} \text{(pfd, pfd, address)} \]

It means that messages of sender 'pfd' are received from port 'pfd' to the location 'address'.

Disconnect

One of the following conditions will cause a port being disconnected:

1. A process requests to disconnect using the system call

\[ \text{pclose} \text{(pfd)} \]

2. Process termination.

An entry is freed from the port table when all processes associated with it have been disconnected.

Message

Two processes may communicate with each other directly by the exchange of messages which are structured data with variable length.

Syntactically, a message is composed of two parts: header and data.

The header part contains:

1. Source pid --- sender's identifier
2. Destination pid --- receiver's identifier
3. Type --- reserved for further use
4. Count --- data field length

(1) and (2) control the flow of a message, and guarantee the exclusive access of a message.

The data part can be one of the following two types:

1. Data with no header
2. Data with header.

Each of these two types is interpreted by the receiver.

The scheduling of messages employs the FIFO discipline. The advantages of this discipline are:

1. Simple in nature,
2. Easy to implement,
3. Save storage,

and the disadvantage is:

1. Infinite delay may occur if a destination process never arrives.

Fig 3.3 illustrates the data structure of a message in the port area.

<table>
<thead>
<tr>
<th>spid</th>
<th>spid: source process id.</th>
</tr>
</thead>
<tbody>
<tr>
<td>dpid</td>
<td>dpid: destination process id.</td>
</tr>
<tr>
<td>type</td>
<td>type: data type</td>
</tr>
<tr>
<td>count</td>
<td>count: datafield length</td>
</tr>
<tr>
<td>data</td>
<td>data: data part</td>
</tr>
</tbody>
</table>

Fig 3.3

4. SOFTWARE INTERRUPT

Software interrupt is a mechanism to notify that a normal or abnormal event has occurred.

In UNIX, 'signal' and 'kill' are two primitives to set the interrupt service entry and to send a signal (interrupt) to a process. For example, synchronization between two processes A and B. Process A waits for event 1 or event 2 and process B sends signal 2 to A, can be coded as shown in Example 4.1.

Example 4.1: synchronization between two processes

```c
process A

signal(1, gl1);
signal(2, gl2);
while(1){
    /* busy waiting */
    ...
    ...;
}
g11{
    /* service function */
}
g21{
    /* service function */
}

process B

kill(pid(A), 2);
```

In practice, there are some drawbacks in example 4.1:

(1) Since pid is dynamically allocated, two unrelated processes are difficult and impossible to know the pid of each other.
(2) Busy-waiting reduces the overall throughput of the system.

So, a more general mechanism is necessary to handle the asynchronous events. Two system calls, 'listen' and 'kill', have been implemented to accomplish this function.
The 'listen' call

```c
listen(&events);
```

allows a process to release CPU and wait for signals to come; where 'signals' is a long integer type.

The 'kill' call

```c
kill(getpid(), signal);
```

sends the signal to the process whose name is 'processname'; where 'getpid' is a function call to
convert process name to pid.

Using these primitives, the Example 4.1 can be recoded as:

```c
process A
struct event{
    int i1;
    int i2;
}

train:
    i1=0;
    i2=0;
    signal(1,gl);
}

signal(2,gl1);
    kill(getpid(A), 2);
    listen(&events);

process B

Example 4.2: messages communication between processes

```c
process A
    struct event{
        int i1;
        int i2;
    }
    train:
        i1=0;
        i2=0;
        setrm("A"); /* registration to the system with a process name */
        do{
            Spid=mntopid("A");
            apid=mntopid("A");
            while (Spid != -1); /* busy waiting to get the designated process id */
            pd=palloc(name,mode);
            fd=open(file,0);
            read(fd,address,count);
            write(fd,pd,address, count);
            kill(Spid,2);
            listen(&events);
            pread(Apid,pd,address);
        }
```

Since processes have the information of the port name and the process name of each other, they can communicate through ports and messages, and can use the software interrupt mechanism to notify the event with each other.

5. DATA STRUCTURES

The I/O system in UNIX is implemented by the mapping of several data types. Fig 5.1 illustrates data types and their mappings.

Data type 'file' contains the global information of the file being opened, including usage, share-count, and a pointer to the 'inode' which keeps the active inode in core (file, device and pipe are nodes in the file system and are mapped uniformly through 'inode', to which an i_number is given).

A file descriptor 'fd' and a local I/O channel number, is an index to the 'ofile' which points to 'file'. An I/O operation, read or write, causes a mapping from 'fd' to i_number.

The new data type 'port' is designed to link with 'file'. Structurally it is similar to 'inode'. One may combine 'port' into 'inode', but in our implementation we have separated them for the following considerations: (1) Functionally and parametrically, port is different from 'file', 'dev', and 'pipe'. (2) It is inconvenient and inefficient for implementing the 'port' to be an 'inode' because of the massive and continuous storage allocation. (3) File system should not be influenced by the side effect of 'port', e.g. reorganization of forests by adding a new type of configuration, bugs in programming the port.

![Fig 5.1](image1)

Fig 5.1 illustrates and comments each field in the port.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>port</td>
<td>port states</td>
</tr>
<tr>
<td>pt_flag</td>
<td>flag: port states</td>
</tr>
<tr>
<td>pt_count</td>
<td>port attributes</td>
</tr>
<tr>
<td>pt_uid</td>
<td>port attributes</td>
</tr>
<tr>
<td>pt_gid</td>
<td>port attributes</td>
</tr>
<tr>
<td>pt_pid</td>
<td>port attributes</td>
</tr>
<tr>
<td>pt_addr</td>
<td>port attributes</td>
</tr>
<tr>
<td>pt_offset</td>
<td>port attributes</td>
</tr>
<tr>
<td>pt_name</td>
<td>port attributes</td>
</tr>
</tbody>
</table>

![Fig 5.2](image2)
6. CONCLUSION

The design and implementation of an interprocess communication channel system based on UNIX operating system has been described. An efficient message-oriented interprocess communication mechanism and processes synchronization directly by explicit signaling have been added to the UNIX system to support the development of distributed systems and to make it easier to connect the computer network. This work has been finished at Institute of Information Science, Academia Sinica, Republic of China, where we are working on a project for developing a distributed database system. More steps of modifying the UNIX system will be taken to achieve the goal of the project.

7. ACKNOWLEDGMENT

The authors would like to express their thanks to the help and assistance of Ms. Chou Feng Wang.

References


A Distributed Database Management System over UNIX Environments

SUMMARY

This report describes the design and implementation idiosyncrasies of a distributed relational data base management system over UNIX environments. A relational model DBMS with a high-level non-procedural query language facility has been implemented and run to test. A transaction monitor is also built on top of the DBMS, which will make the distribution of data and processing power be transparent to the user.

There has been a lot of debate over the inefficiency of the relational model data bases. However, since our major concerns in selecting data model were in choosing a data model with high degree data independency and strong mathematical foundation for query processing, we were convinced that the relational model is the only model which can meet our goal.

Implementation of the DBMS has been strongly influenced by INGRES[STON76] and System R[ASTR76]. The DBMS includes two sub-systems, namely Relational Interface System(RIS) and Physical Storage System(PSS)(see Figure 1). The RIS provides high level, data independent facilities for data definition, manipulation, and retrieval. The data definition facilities of the RIS allow the DBA to define the data at conceptual level as well as internal level. They also allow a variety of external views to be defined upon common underlying data to provide different users with different views of the data base. The external views of the common data also facilitates some authorization checks and
privacy control. One of the important features of the DDL facility is that each attribute in the relational database may be optionally associated with an integrity control routine which will be triggered to execute upon the corresponding update operation. The data manipulation facilities of the RIS allow data to be inserted, deleted, modified, manipulated, and retrieved by using a set of high level relational operators which exempt the users from knowing the internal storage structures of the stored data. Moreover, a high level, nonprocedural query language has also been implemented to provide a friendly interface for the novice users. Each query statement is translated into a transaction constructed by a sequence of relational algebra operators with some heuristic optimization techniques.

The PSS manages disk storage allocation, indexed file access. B-tree structure has been used for indexing secondary keys of each relation. The disk space contains a set of files each of which is constructed by a set of fixed-size pages. For the purpose of saving space and speeding up access, we employ the technique of mapping logical space (what the relational operator sees) into physical space (what the data is really stored). A page map table has been maintained to map logical page number into physical page number. This mapping technique can reduce the problems of hashing dispersed records and is particularly useful in overflow control. The PSS also supports concurrency control by using shared-read-lock and exclusive-write-lock with lock=all-or-none strategy.

For the purpose of providing end users with a consistent data base and making the user have an illusion that they are
using a large unified database, a transaction monitor has been implemented on top of the database management system. The transaction monitor receives transactions from the end users and serializes the DML statements of multiple transactions to achieve high-degree parallelism of executing database operations.

Due to the address space limitation of PDP11/23 (64K bytes), the DBMS under UNIX has been structured into several related processes. The process structure of the DBMS is shown in Figure 2. The function of each process is self-explanatory and will not be recounted here. Routines for indexed access, integrity control, and error recovery will be called by these processes at proper time.

In the following, Section 1 illustrates the DDL and DML facilities provided by the RIS, Section 2 describes the storage structures and access technique of the PSS, Section 3 presents the query processing in the distributed database environment, and Section 4 discusses the problems of concurrency control.

1. Data Definition Language and Data Manipulation Language

DDL is used by the Data Base Administrator to define database schemas and subschemas. Figure 3 illustrates the syntaxes of DDL statements.
DATA DEFINITION LANGUAGE (DDL)

/*
domain definition */
DOMAIN IS name;
PIC(TRUE) IS { F(9), S(9), I(9), V(9) .... }
[ LEFT(RIGHT) JUSTIFY ];
TYPE IS { COMPUTATIONAL, DISPLAY, BIT, INDEX }
[ FROM int1 TO int2 STEP BY int3 ]
[ (brown, red, ....) ]
[ IN units ];
END;

/*
relation definition */
RELATION IS name;
USER IS <name, right> ......... ;
WITHIN directory fileno FILES;
(assign filename in accordance with
the directory and the fileno)
[ PAGE SIZE blockno ;] (default pagesize is 4 blocks)
FIELD role name DOMAIN name
[ IS { ACTUAL , VIRTUAL } RESULT OF derived program ]
[ ON GET CALL check_program ]
[ ON STORE CALL check_program ]
[ ON MODIFY CALL check_program ]
[ ON DELETE CALL check_program ];
FIELD...... ;
RULE PART name1,name2,.... ;
[ ACCESS USING { VSAM, DAM, .... } ; ] (default is DAM)
[ INDEX USING name1,name2,.... ; ]
[ INDEX USING name1,name2,.... ; ]
[ ON GET CALL check_program ; ]
[ ON STORE CALL check_program ; ]
[ ON MODIFY CALL check_program ; ]
[ ON DELETE CALL check_program ; ]
[ ON OPEN CALL concurrent program ; ]
[ ON CLOSE CALL concurrent program ; ]
[ ON PRIVACY CALL check_program ; ]
END;

Figure 3. Syntaxes of Database Definition Language

The description of DDL can be found in [KE81], and will not be recap for here.

Using DDL, the DBA can define database schemas which will be converted into machine code by a DDL compiler and kept in four dictionary files. Each of these four files keeps some relevant information about the defined relations as described below:
(1) DIRECTORY file:

Keep all domain names and relation names information by using balance tree data structure.

(2) DOMAIN file:

Keep information of each domain.

(3) RELATION file:

Keep information of each relation.

(4) PROTECTION file:

Keep protection information of each relation.

These four dictionary files will be used by the DML interpreter and all storage operators.

DML is used by the end user to access the stored data. Figure 4 illustrates the syntaxes of DML statements. Detail of the semantics of each DML statement can be found in [KE81].
DATA MANIPULATION LANGUAGE

Storage operations:

(1) DELETE
    FROM        relname
    WHERE clause.

(2) MODIFY
    FROM        relname
    SET(WITH)   (new value list with assignment syntax)
    WHERE clause

(3) STORE
    FROM       (data list)
    $PIPE load via pipe file
    $FILE filename ...load via data file
    INTO relname

Information retrieval operations:

(1) RETRIEVE
    FROM        rel1
    INTO rel2   [ < attr. list >]
    [ WHERE clause ]

(2) UNION
    FROM rel1,rel2,...,reln
    INTO relnm  [ < attr. list >]
    [ WHERE clause ]

(3) INTERSECT
    FROM rel1,rel2,...,reln
    INTO relnm  [ < attr. list >]
    [ WHERE clause ]

(4) DIFFERENCE
    FROM rel1,rel2
    INTO relnm  [ < attr. list >]
    [ WHERE clause ]

(5) DIVISION
    < attr. list ; attr. list >
    FROM rel1,rel2
    INTO relnm  [ < attr. list >]
    [ WHERE clause ]

(6) JOIN
    [ < attr. list ; attr. list > ]
    FROM rel1,rel2
    INTO relnm  [ < attr. list >]
    [ WHERE clause ]

Figure 4. Syntaxes of DML statements
Usually the user's simple query can be answered by a sequence of DML retrieval statements. For the complicated query, a C-language program can be written to call the DML statements and then process the retrieved data.

2. Physical Storage Structures and Operations

The physical storage of the database is a collection of UNIX files. A relation to the user is usually stored as one or more disk files. The storage hierarchy of the database is illustrated in Figure 5.

![Database storage hierarchy diagram]

Figure 5: Database storage hierarchy
Logically a relation contains a set of numbered fixed-size pages (the default page size is 2k bytes) which may be actually dispersed in the disk storage. A page is the basic unit transporting between the disk and the user buffer. Each page has a logical page number which is referenced by the relational operators. Pages with contiguous page numbers do not indicate that they are stored in contiguous physical space. A page map table is maintained to map logical page numbers into physical page addresses. This mapping technique is particularly useful in storing dispersed data and overflow control. Data tuples are stored in each page sequentially. Each page maintains a tuple address directory, each slot of the directory contains the tuple address, tuple prefix size, and tuple data size. The data structure of a page is shown in Figure 6.

<table>
<thead>
<tr>
<th>offset</th>
<th>prefix length</th>
<th>tuple size</th>
</tr>
</thead>
</table>

**Figure 6. Page structure**
Each tuple has a tuple ID (TID) which is constructed by concatenating a page number and a slot number. Three access methods are provided for storing or retrieving data from dat pages:

* **Sequential**: accessing data tuples with page-by-page sequential searching.

* **Direct**: using 'hashing' to find the page number and then search the tuples in that page sequentially.

* **Indexed**: using B-tree index to access tuples. The data structure of B-tree is shown in Figure 7.

**INDEX file:**

```
header key value address key value address ...
```

```
header key value address key value address ...
```

```
header key value block# record# key value block# record# ...
```

**TID file:**

```
tid1 tid2 tid3 ....
```

*Figure 7. B-tree structure*
The problem of concurrent accesses control in the DBMS has also been tackled by using 'lock' mechanism at relation level. Each relation has an associated lock. Share-read-lock and exclusive-write-lock policies are employed to control the access of every relational file in the sense that multiple users can read the same relational file at the same time; however, when a relational file in being updated by a user, all other next coming access requests, either read or write, should be suspended. When a relational command (e.g. RETRIEVE, JOIN) is being executed, all relational files it refers to should be locked first. This lock all or none strategy is mainly used to prevent deadlock situations.

3. Query Processing

One of the objects of the DDBS is to make the distribution of data and processing power be transparent to the user. In DDBS, we achieve this goal by providing the user with a high level query language which in turn is supported by the 'VIEW' definition facility.

3.1 Query translation technique

A VIEW is defined by the DBA and is a subschema of a set of related data items in the problem domain. To the user, each VIEW represents the relationships among those data items that the user is allowed to access. The set of data items defined in a VIEW may be distributed over several sites, however, the locations of stored sites and the data access paths are transparent to the
user. In the following we will use an example to illustrate the
technique of query processing in the DDBS.

Assume that three sites, A, B, C are stored with data as
described below:

Site A: \text{BOOK} (\text{BID}, \text{BNAME}, \text{BAUTHOR}, \text{PUBLISHER}, \text{PUB_DATE}, \text{CATEGORY}, \text{STATUS}, \text{L\#}, \text{S\#})

\text{JOURNAL} (\text{JID}, \text{JNAME}, \text{VOL}, \text{NO}, \text{DATE}, \text{JPUBLISHER}, \text{JCATEGORY}, \text{L\#}, \text{S\#})

\text{PROCEEDING} (\text{PID}, \text{PNAME}, \text{CONFERENCE}, \text{DATE}, \text{LOCATION}, \text{L\#}, \text{S\#})

Site B: \text{PAPER} (\text{PNO}, \text{TITLE}, \text{PAUTHOR}, \text{JPID}, \text{DATE}, \text{PAGE})

Site C: \text{SUPPLIER} (\text{S\#}, \text{SNAME}, \text{SCITY}, \text{SPHONE})
\text{LIBRARY} (\text{L\#}, \text{LCITY}, \text{ADDRESS}, \text{DIRECTOR}, \text{LPHONE})

Basing on this knowledge of data distribution, the DBA can define
a VIEW, named \text{VIEW_LIB}, as follows:

\text{DEFINE} \quad \text{FROM} \quad \text{BOOK.A, JOURNAL.A, PROCEEDING.A, PAPER.B, SUPPLIER.C, LIBRARY.C}
\text{INTO} \quad \text{VIEW_LIB(*)}
\text{WHERE} \quad \text{BOOK.S\#=SUPPLIER.S\#}
\text{AND} \quad \text{JOURNAL.S\#=SUPPLIER.S\#}
\text{AND} \quad \text{PROCEEDING.S\#=SUPPLIER.S\#}
\text{AND} \quad \text{BOOK.L\#=LIBRARY.L\#}
\text{AND} \quad \text{JOURNAL.L\#=LIBRARY.L\#}
\text{AND} \quad \text{PROCEEDING.L\#=LIBRARY.L\#}
\text{AND} \quad \text{PAPER.JPID=JOURNAL.JID}
\text{AND} \quad \text{PAPER.JPID=PROCEEDING.PID}

In response to this VIEW definition a semantic graph (see Figure
8) will be generated and can be used as the foundation for query
translation.

In this example, we have included all data items in
\text{VIEW_LIB}. However, this is not always the case. The DBA may de-
fine a VIEW which contains only a subset of data items depending
on the security degree of the user.
Now, suppose that a user at site C issues a query, to find the director of the library which owns a journal that contains the paper "On Modeling Relational Databases", by specifying the query as follows:

\[
\text{RETRIEVE FROM VIEW LIB INTO R(L\#,DIRECTOR) WHERE TITLE='On Modeling Relational Databases' AND PID=JPID}
\]

Basing on the semantic graph in Figure 8, the access path for this query can be found as shown in Figure 9.

\[\text{Figure 8: Semantic Graph}\]
Figure 9: Access Path

From the access path shown in Figure 9, a transaction for retrieving data to answer the user's query can be generated as shown below:
BEGIN_TRAN
B: RETRIEVE FROM PAPER INTO R1(JPID)
   WHERE TITLE='On Modeling Realitonal Databases'
A: RETRIEVE FROM PROCEEDING INTO R2(PID,L#)
B: MOVE B.R1 TO C.R1
A: MOVE A.R2 TO C.R2
C: JOIN (R1,R2) INTO R3 WHERE JPID=PID
C: JOIN (R3,LIBRARY) INTO R(L#,DIRECTOR) WHERE
   R3.L#=LIBRARY.L#
END_TRAN

The ordering of DML statements has been optimized by first reducing the relations with restriction and projection operations. The reduced data will be moved to the query site, and join operations will then be performed if necessary. The technique of using semi-join[BERN81] has not been employed in this implementation. However, we have developed a technique, called graph projection[CHAN82], which hopefully can minimize the overhead of semi-join operations. We plan to implement this technique in DDBS query processing in the near future.

3.2 Query Processor

Figure 10 shows the procedures of processing a user's query. A user's view definition is compiled to generate a View Map(VM) and a Relation Distance Matrix(RDM). When the user issues a query, the query will be scanned by a parser which examines the syntactical correctness of the query and generates a parsing tree. The parsing tree is then sent to the defuzzifier which determines the semantic correctness of the query and finds the access path for the query by using the information stored in VM and RDM. This access path will then be used by the code generator to generate an equivalent transaction command which is constructed by a sequence of relational commands. The resulted
transaction command is finally sent to the transaction monitor for being executed.

```
view definition
    ▶ Compiler ▶ VM
        ▲
        RDM

user's query ▶ Parser ▶ Defuzzifier ▶ Code generator ▶ Transaction monitor
```

Figure 10. Query Processor

4. Concurrency Control

In DDBS, a transaction is essentially constructed by a sequence of relational commands and is the basic unit of computation from the user's viewpoint.

At the present time of this writing, the concurrency control at transaction level has not been implemented yet. However, in this section we will propose a technique for handling concurrency control problem, with granularity at relation level, in our DDBS. This technique is a simplified version of SDD-1's[BERN80] approach of handling concurrency control problem. Figure 11 shows a model for processing transactions in the distributed database environment.
Figure 11. Transaction Processing

At each site, there is a Data Base Management System (DBMS), a Transaction Scheduler (TS), a Transaction Monitor (TM), a Communication Subsystem (COMM). Each user is assigned with a Query Processor (QP) to translate the user's query into a Transaction
Command(TC) which in turn is constructed by a set of Relational Commands(RC). All transaction commands are sent to TM for being analyzed and then distributed to proper execution sites via COMM. The TM first puts a timestamp on each transaction command. This timestamp is generated by concatenating the system clock with the identification number of TM. TM also classifies every relational command into READ, WRITE, and EXECUTE operations. This classification can be shown below:

.join R1 and R2 into R3

READ R1
READ R2
EXECUTE
  * WRITE R3

.retrieve from R1 into R2

READ R1
EXECUTE
  * WRITE R2

.move A.R1 to B.R2

  at A: READ R1
  * at B: WRITE R2

.modify R1

READ R1
EXECUTE
WRITE R1
Those statements preceded with a '*' are concerned with intermediate relational files, and can be ignored when considering concurrency control problem. Other relational commands can be classified similarly.

TM collects all READ(WRITE) commands in a transaction into a READALL(WRITEALL) command. It then submits these READALL/WRITEALL commands to TS's by following some synchronization protocols. Figure 12 shows the synchronization protocol between TM and TS.

TS processes the received READALL command by reading all the desired data into a workspace (one for each transaction per site). After finishing the READALL command, an acknowledgement is sent to the TM which submitted the READALL command. The EXECUTE commands are then processed on the workspace independently. When all EXECUTE commands of a transaction have been completed, TM submits the WRITEALL command to TS for updating with new data.

Every relational file has an associated timestamp. This timestamp is the timestamp of the last WRITE command that updated it. All relational files can only be updated by the WRITE command with a larger timestamp. All READ commands can only read the relational files with a smaller timestamp.

When TS receives commands from either the remote TM (via COMM) or the local TM, it will serialize the READALL commands and WRITEALL commands and guarantees the effective total serial ordering of executing transactions. In the following we will use the example transaction of Section 3 to illustrate the process of serialization and synchronization.

93
Figure 12. Synchronization protocol between TM and TS
For illustration, we add a DELETE command in the example transaction as below:

BEGIN_TRAN
B: RETRIEVE FROM PAPER INTO R1(JPID)
   WHERE TITLE='On Modeling Relational Databases'
A: RETRIEVE FROM PROCEEDING INTO R2(PID,L#)
B: MOVE B.R1 TO C.R1
A: MOVE A.R2 TO C.R2
C: JOIN (R1,R2) INTO R3 WHERE JPID=PID
C: JOIN (R3,LIBRARY) INTO R(L#,DIRECTOR) WHERE
   R3.L#=LIBRARY.L#
C: DELETE FROM LIBRARY WHERE LIBRARY.L#=R.L#
END_TRAN

This transaction includes seven relational commands. The READ*ALL commands and WRITE*ALL commands that will be submitted to the execution sites are described below:

To TS of Site A: READ*ALL(PROCEEDING)
   WRITE*ALL(none)
To TS of Site B: READ*ALL(PAPER)
   WRITE*ALL(none)
To TS of Site C: READ*ALL(LIBRARY)
   WRITE*ALL(LIBRARY)

At READ*STATE, the three READ*ALL commands will be sent to A, B, and C, respectively. These READ*ALL commands will be serialized with other transactions by the respective TS receiving
them. After all READ\ALL commands and EXECUTE commands have been processed, the TM will enter WRITE\STATE and send the WRITE\ALL commands to each TS. These WRITE\ALL commands will also be serialized by the respective TS receiving them.

CONCLUSION

This report describes the design and implementation of a distributed relational data base system.

A high-level nonprocedural query language facility has been implemented to facilitate the novice users. Using this query facility, the concept of access path is transparent to the user.

A concurrency control model has also been proposed to be implemented for our DDBS system. We understand that the proposed model may not be flexible enough for real-time applications. However, we believe that this model is very reliable and customizable. More flexible and efficient techniques are under study.

A library application system is now run to test the reliability and performance of the DDBS. The statistical results of this test will be presented in a further report.
REFERENCES


[ROTH80] Rothnie, J. B., etal., "Introduction to a System for Distributed Databases(SDD1)", ACM TODS, VOL.5, NO.1, March 1980.

This paper presents a petri-net-based formal modeling tool for the specification and verification of communication protocols. A protocol specification language has been designed and used for describing communication procedures and events. The development of a communication protocol simulation system has also been described. Finally, the alternating bits protocol has been used as an example to demonstrate the modeling power of the simulator.

1. Introduction

In last decade, a number of large scale computer networks have been implemented. Many of the problems in designing computer communication protocols, which involve asynchronous parallel processing, are due to the difficulty of realizing and analyzing the sequence of event occurrences. For overcoming these problems, some formal modeling tools are required. A formal modeling tool must be fit for protocol specification and verification. Many modeling tools have been proposed by specialists in the literature. These tools include Finite State Automata (FSA), Petri nets, and high level programming language. FSA is a good modeling tool, but it is not fit for modeling concurrent events and may become very large when the protocol is complex. Petri net is a good tool for modeling concurrent events, but it has the same problem as FSA when the protocol is complex, because that the Petri net can not describe numerical events as well. The high level programming language can be fit for describing the numerical events, but it can not be fit for describing "event occurrences".

In this paper, we propose the Petri Net Derived Model (PNDM) to overcome these problems. The extensions in PNDM based on the general Petri net may be categorized into four areas, including place extension for describing external events, variable extension for describing the formats of messages transferred, transition extension for describing transition actions and transition hierarchy, and function extension for describing the special conditions of protocols.

We have also developed a PNDM-based Communication Protocol Modeling System (CPMS) to simulate the behaviors of communication protocols. The CPMS simulator can detect deadlock situations and livelock situations in communication protocols, if any.

In the rest of this paper, section 2 surveys the existing methodology for the specification and verification of communication protocols. Section 3 is concerned about the Communication Protocol Modeling System (CPMS), Petri Net Derived Model (PNDM), and Protocol Specification Language (PSL). Section 4 presents an example of constructing PNDM of the alternating bit protocol and using the CPMS to simulate the same protocol.

2. Surveys of Protocol Modeling Techniques

A protocol is the set of rules which govern the exchange of messages between cooperating processes.

A protocol specification should state all the requirements that the protocol must satisfy. For many identified reasons and advantages (8), the protocol architecture is usually layered into several levels.

In general, each layer N of the protocol hierarchy may have three required specifications, namely, the N-service specification, the N-interface specification, and the N-protocol specification.

Presumably, protocol specification methodology may be classified as either transition model or programming languages model. The former includes flow chart, finite state automata, state transition matrices, formal grammars, petri nets and UCLA graphs. The latter includes "Structure English" and high-level programming languages.

The transition model generally consists of a set of states and transitions. States describe the situation of the modeled system, and transitions describe some activities to do in response to event happenings. When the protocol is complex, the number of transitions and states of the modeled system may become very large.

Programming language model describes protocols as one type of algorithms. This approach's major advantage over the pure
state transition model is that it has the capability of handling variables and parameters. But, the programming language model has the disadvantage of not being good for describing event happenings.

Mixed approaches have been proposed in the literature [1], which combine the advantages of state transition model and programming language model.

All these techniques have a common problem, that is, how to describe the real protocol completely and consistently so that they may be understood, analyzed and implemented. Most specialists only participated in the procedural specification of the protocol. In [9] and [6], the logical specification of the protocol, i.e. the format and meaning of messages specification have been included by employing the concept of abstract data types. However, both of them are inadequate of analyzing systems with concurrently executing components.

Protocol verification is a demonstration that the interactions of protocol modules satisfy the service specification. But, most of verification work has been done on design rather than implementation. We can only verify if the protocol is logically correct. In this sense, we can divide the functions of protocol verification into two categories, namely, general properties and specific properties.

General properties include:
- Liveness (freedom from deadlocks)
- Completeness
- Stability
- Freedom from improper loops
- Recovery from reception errors
- Termination

The specific properties of protocols are some application-dependent services to be provided.

Approaches to protocol verification have followed the two classifications of specification, namely, reachability analysis for transition techniques and program proofs for programming languages techniques.

Reachability analysis involves the exploration of all the possible interactive states of two or more protocol machines. It is well-suited to check some "general properties" described earlier, because these properties are a direct consequence of the structure of reachability graphs. However, the reachability graph may result in "state explosion" when the protocols become complex, this is the major problem of the reachability analysis approach.

The program proving approach involves the usual formulation of assertions to reflect the desired correctness properties. A major strength of this approach is its ability to deal with the full range of protocol properties to be verified rather than only general properties. But, several properties of protocols pose special difficulties for program proofs. These include concurrency of multiple protocol modules, message exchanges and use of shared variables among physically separated modules.

Other techniques used as methods of protocol verification are simulation and testing. Both techniques may find most design errors, but they cannot guarantee that all errors can be found.

At present, all protocol verification techniques could only prove correctness partially but not totally.

3. The Communication Protocol Modeling System

In this section, we propose a new model, which is called Petri Net Derived Model (PNDM), based on General Petri Net, and extended with some properties to increase the power of modeling. Basing on PNDM, a protocol simulator, called the communication protocol modeling system, has been developed for verifying the correctness of communication protocols. Fig 3.1 shows the architecture of the Communication Protocol Modeling System.

![Diagram of Communication Protocol Modeling System]

Fig-3.1 Communication Protocol Modeling System

With CPMS, the designer of communication protocols must first use PNDM to describe the protocol which concerns what is needed to be done and how to do it. The PNDM is then transformed into the syntactic description by using the protocol specification language, and is taken as the input of the CPMS through the operation of CPMS, i.e. through the simulation, to find out if any errors may take place. Repeatedly modify and simulate the PNDM until no more errors can be detected. Consequently, protocols may be implemented semi-automatically.

3.1. Petri Net Derived Model (PNDM)

The extensions in PNDM based on general petri nets can be categorized into four fields, including place extension, variable extension, transition extension, and function extension.

3.1.1. Place extension
In PNDM, a new type of places is introduced. The new type place is called external place in the sense that it has no input arc from any transition, but may generate tokens automatically when some special events occur.

3.1.2. Variable extension

Since petri nets have no way to describe numerical events, we extend them with some variables.

This extension includes three types of variables, namely, structure variables, counter variables, and timer variables. Each of these variables has its special usage and may increase the modeling power of Petri nets.

3.1.2.1. Structure variables

This type of variables is mainly needed for describing messages which are transferred between two communication stations. To use structure variables, the user must declare their formats as follows.

- variable name : simple type name (multiplicity) (1)
- variable name : block type name (2)

Expressions (1) and (2) are formal forms of variable declaration. Type name is the name of data type which may be categorized into simple type and block type. Simple type includes three formats: bit, byte, and word, and their keywords are B1, B8, and B16, respectively. Block type is the combination of simple types, it is declared at the type declaration part. The multiplicity means the length of the variable with type declared in expression (1). If the multiplicity is null, the string will have dynamic length which is determined by the assignment statement explained at the transition extension part (see 3.1.3).

The structure variables are mainly used for message transfer, and may be assigned and compared in the assignment statement and if-statement of transition extensions.

3.1.2.2. Counter variables

This type of variables may be used for controlling transition firings, i.e. it may influence the sequence of transition firings. The influencing method will be introduced at the transition extension part.

Counter variables are declared at the variable declaration part, they must declare counter name with modulus value. Each counter may be assigned by a value, and its value may be increased with one by increment-statement. The counter variables may also be compared in if-statement.

3.1.2.3. Timer variables

The value of timer variables can be initialized by the start-statement in which value is the modulus declared at timer variable declaration.

The value of timer variables is automatically decremented by the execution of the simulator. If the value of timer variables decreases to zero, the timeout mechanism of timers will be activated. The timeout signal may influence the sequence of transition firings, because the condition of forcing functions may be determined by timeouts.

3.1.3. Transition extension

Transitions in PNDM may be divided into terminal transitions and non-terminal transitions. Terminal transitions may be described by transition action statements, and their firing actions about token flow may be determined by the values of the associated counter variables. Non-terminal transitions may be replaced by a petri net module supported by PNDM.

3.1.3.1. Transition action description

The transitions of PNDM may use the action-statements to describe the actions of transition firings, and these action-statements may be categorized into six types.

3.1.3.1.1. Counter-statement

Two types of counter-statements are shown as follows:

- Counter-assign-statement
  - For example,
    - A := 2
      - sets the counter variable A have value 2.

- Counter-increment-statement
  - For example,
    - ++A
      - means the value of counter variable A will be incremented by one.

3.1.3.1.2. Timer-statement

Two types of timer-statement are shown as follows:

- Timer-start-statement
  - For example,
    - timer TA := 0
      - starts timer TA.

- Timer-destroy-statement
  - For example
    - <timer TA
      - stops the mechanism of timer TA.

3.1.3.1.3. Transfer-statement

Two types of transfer-statement are shown as follows:

- Send-statement
  - For example,
    - send MESS
      - sends the message MESS with format
declared at variable declaration and type declaration to transfer-port 1.
.. Receive-statement
   For example, receive 1 MESS
   receives the message MESS from transfer-port 1.

3.1.3.1.4. Termination-statement

Two types of termination-statement are shown as follows:
   .. Stop-statement
     For example, stop
     can stop the operation of simulator.
   .. Exit-statement
     For example, exit
     exits from the transition firing, and the transition is regarded as
     that it has been fired.

3.1.3.1.5. Assignment-statement

A structure variable may be assigned by this statement. For different types of
variables, we must use different keywords for the simulator.
   For example,
   HEADER= $01001001 (14)
   TITLE= 104 (15)
   DATA= "abcdef" (16)

   The equal symbol '=' represents assignment, the percentage symbol '%' represents
   the header of bit string, the double quote "" represents the header of
   byte string, and the word variable is directly assigned by an integer.

3.1.3.1.6. if-statement

The format of if-statement is:
   if (condition) statement-1 else
   statement-2.

   If the condition is true, then execute
   statement-1, else execute statement-2.

   The condition part may allow the counter-
   variables and non-counter-variables to be
   compared.

3.1.3.2. Transition control

This is the ability of transitions
that they can select the path of token
moving when a transition fires.

There are two types of such transitions
as shown in Fig 3.2 and Fig 3.3.

3.1.4. Function extension

PNDDM allows users to set the initial
value of variables, and to terminate the
PNDDM by describing the termination condi-
tions. It also allows the force function to
change states of PDDM.

3.1.4.1. Initial function

The users may initialize all places
and counter variables by using initial
functions at the module initialization
part. For example, pl:2 means that place
pl holds two tokens, and A <->2 means that
counter A has initial value two.
3.1.4.2. Termination function

The users can specify the normal termination condition of PNMD at the module termination part. If the status of PNMD matches the termination condition described by the user, the simulator will terminate and restart itself to simulate again.

The termination condition is composed of places names with the number of tokens they hold. For example, p1:2; p2:1; i.e., place p1 holds two tokens, place p2 holds one token, and no tokens are in other places.

3.1.4.3. Forcing function

This function is similar to the termination function, except that it only activates some actions to be executed. The user can describe many forcing functions each of which includes two parts, one is the forcing condition, and the other is the forcing action.

For example, p1:2; p2:4; timeout t01; force p1:1; p2:1; p3:1;

In this forcing function, texts before "force" are the conditions, and after it are the forcing actions. While place p1 holds two tokens, place p2 holds four tokens, and timer t01 timeouts, the forcing condition matches, and the PNMD will be enforced to execute the forcing action, i.e., each of place p1, p2, and p3 holds one token.

Using forcing functions can increase the modeling power of PNMD, particularly we can add the timeout mechanism into the forcing condition, hence the modeled system may be more close to the real world.

2.2. Protocol Specification Language (PSL)

For the purpose of computer processing, the protocol designer must transform the PNMD into the linear syntactic form by using the protocol specification language (PSL).

The protocol specification language is a high-level procedural language, its basic unit is called module, which includes eight parts, as described below.

Place declarations and transition declarations are used to declare the places and transitions by their names. These two parts must exist in every module.

The data types used for variable declaration are declared at the type declaration part. At the variable declaration part, there are three types of variable declaration, namely, structure declaration declares the data variable for message transfer, counter declaration declares counter variables, and timer declaration declares timer variables. The system is initialized at the module initialization part which is one type of function extensions, and is explained in 3.1.4.1. In this part, the designer may initialize the places with the number of tokens held and the value of counters which have been declared.

Next part is called the module termination part, the designer can specify the termination conditions which can be more than one, and each termination condition may be represented by places with the number of tokens to be held.

The module forcing part is similar to module termination part, except that the forcing function doesn't terminate while the condition matches, and will go into a new situation defined by the force part of the forcing function. Its condition may be determined by the timeout of timers.

The last part is the most important in PSL, mainly to define the structure of the PNMD, and it is called transition description. Each transition structure must be described in this part which includes the input places, output places, and transition actions explained in 3.1.3.1.

4. The Alternating Bits Protocol

In this section, we will use the alternating bit protocol as an example to describe the procedures of using the CPMS simulator.

The alternating bits protocol is provided for a simple but reliable message transfer service over an unreliable transmission medium. It uses an one-bit sequence number for each message sent. The sequence number is complemented on each new message sent. When the receiver receives messages correctly, it transmits the acknowledgement is not received by the sender within a time interval, the sequence number will be retransmitted. The protocol guarantees correctly sequenced delivery of messages even if the medium loses messages or acknowledgements.

4.1 Modeling the alternating bits Protocol by PNMD

In Fig. 4.1, we add new transitions ML and AL to represent the message and acknowledgement transmission losses.

SEND

\hspace{1cm} t1 \hspace{1cm} t2 \hspace{1cm} t3 \hspace{1cm} t4 \hspace{1cm} t5 \hspace{1cm} t6 \hspace{1cm} ML

RECEIVER

\hspace{1cm} p1 \hspace{1cm} p2 \hspace{1cm} p3 \hspace{1cm} p4 \hspace{1cm} p5 \hspace{1cm} p6

\hspace{1cm} VAR: M(Mess. No) Fig. 4.1 VAR: A(ACK. No)

Here, we use counter M and counter A to represent the local counters of sender and receiver, respectively. In addition, we use the counter R to represent the number of retransmission times. The action of transition t2, t4, and t5 are replaced by
transition action statements as shown below.

\[
\begin{align*}
t4 & \text{ receive 1 mess exit} \\
& \text{ if (mess-no == %1) ax<-1} \\
& \text{ if (mess-no == %0) ax<-0} \\
& \text{ if (ax == a) ++a} \\
t2 & \text{ receive 2' ack exit} \\
& \text{ ++m} \\
& \text{ r <- 0} \\
& \text{ <- timer t01} \\
t5 & \text{ send 2 ack}
\end{align*}
\]

At transition t4, when the sequence number of the message received is correct, the local counter A of the receiver will be complemented by statement ++a. At transition t2, when the sequence number of acknowledgement received is correct, the local counter M at the sender will be complemented by statement ++m and the retransmission times counter will be reset by statement r <- 0. At transition t5, the receiver sends the acknowledgement to the sender.

The action of transition t1 may be extended to become a submodule t1.sub by employing the concept of transition hierarchy as shown below.

\[
\begin{align*}
t1(t1.sub) & \text{ H} \\
& \text{ M=0} \\
& \text{ P2} \\
& \text{ t3} \\
& \text{ M=1} \\
& \text{ P3} \\
& \text{ t4} \\
& \text{ t5}
\end{align*}
\]

\[
\begin{align*}
t1.t2 & \text{ when(t1.p1*m==0: t1.p2 t1.p1*m==1: t1.p3) } \\
t1.t3 & \text{ mess-no=0} \\
t1.t4 & \text{ send 1 mess-no} \\
t1.t5 & \text{ mess-no=1} \\
t1.t5 & \text{ send 1 mess-no} \\
t1.t5 & \text{ +++r} \\
& \text{ if (r<5) stop} \\
& \text{ timer t01 <- 0}
\end{align*}
\]

The action of transition t1 send the message with sequence number determined by the value of local counter M. When the number of retransmission times is over five, the simulation will be terminated, and the timer is started by the statement' timer t01 <- 0' at transition t1.t5.

4.2 Converting Representation from PNDM to PSL

After the protocol has been specified as a PNDM, we then transform it into the form of PSL. We first declare the places, transitions, and variables used, and then specify the termination conditions and forcing functions. Finally we describe the structures and actions of each transition. Appendix 1 shows the PSL specification of the alternating bit protocol. Appendix 2 shows part of the simulation results. Detail of the simulation results can be found in [2].

5. Conclusion

The Petri Net Derived Model (PNDM) has been proposed for communication protocol specification in this paper. The PNDM can specify some properties of communication protocols. These properties include the specification of concurrent behaviors between the communication stations, the numerical events description (e.g. retransmission times), the special condition description (e.g. timeout and protocol termination condition), and the actions of transition firings.

The PNDM can also help the protocol designer to solve the logical problems of communication protocols. These problems include the self synchronization which can be solved by the forcing function of PNDM, and the recovery from reception errors which can be solved by the timeout mechanism of PNDM.

For protocol verification, the Communication Protocol Modeling System (CPMS) has been implemented. The CPMS can find out the livelock condition and the deadlock condition in communication protocols, and provide the sequence of transition firing for the protocol designer to verify the completeness property of communication protocols.

The PNDM has been used for specifying the alternating bits protocol, and the CPMS has been used for simulating the alternating bits protocol in this paper. The experiment has proved that the integration of PNDM and CPMS is a good tool for the specification and verification of communication protocols.

While we feel that we have had considerable success in handling communication protocols with PNDM, several areas need further studies. These areas include:

- Extend the expressive power of protocol specification language.
- The type declaration can be extended to simplify the specification of high level protocols, e.g. the file transfer protocol.
- Extend the verifying power of protocol simulation.
- More properties of communication protocols, e.g. completeness and cyclic behavior, can be examined by increasing the complexity of simulation.
- Communication protocols may be implemented automatically.

The PNDM can be automatically transformed into hardware circuits and software routines which can work as the desired communication protocol machines.

REFERENCES

1. J. Day & C. A. Sunshine, "A Bibliograph

Appendix 1: Alternating Bits Protocol
This appendix only shows part of the ABP description. A complete description can be found in [21].

modb test * module name
dclp * place declaration
pl1; p2; p3; p4; p5; q1; q2; q3; q4;
dclp
dclt * transition declaration
tl1; tl2; tl3; tl4; tl5; tl6; ml = 1;
edclt
dclv * variable declaration
struct
mess-nc: bl(1)
ack-nc: bl(1)
ctr a(2); m(2); z(5);
mx(2); ax(2);
tmcr t01(9);
edclv
modi * module initialization
p31; p61;
var m - 0
a - 0
r - 0
emodi
modt * module termination condition
term testl pl1; p4:
eterm testl
emodt
modf * module forcing
forcing testl

timeout t01
force pl1
p2: 0
p3: 0
eforcing test1
force pl2
p2: 2
p3: 2
p4: 2
eforcing test2
emodf
dest * transition description
tran t6 * transition t6
rarc * receive arc
p6
erarc

tran tl * transition tl
* transition hierarchy
* describing transition tl
modb tl.sub * transition hierarchy
dclp * describing transition tl
tl1; tl2; tl3; tl4; tl5; tl6;
edclp
dclt
tl1; tl2; tl3; tl4; tl5; tl6;
edclt;
dest

tran tl.t4 * transition tl.t4
farc
etl4 * end of module tl
ertran tl
erest
emodt

APPENDIX 2: CPMS simulation results of the alternating bits protocol
place p1 hold 1 token
place p2 hold 0 token

transition t5 is selected to simulate.
transition t6 is selected to simulate.
transition ml is selected to simulate.
transition ml is selected to simulate.
transition tl1 is selected to simulate.
transition tl2 is selected to simulate.
transition tl3 is selected to simulate.
transition tl4 is selected to simulate.
transition tl5 is selected to simulate.

transition tl1 is enabled
transition tl1 firing
absorb 1 tokens from place p1
send 1 token to place tl1.p1
absorb 1 tokens from place tl1.p1
send 1 token to place tl1.p1
send 1 token to place tl1.p1
absorb 1 tokens from place ml
send 1 token to place ml.p3
send 1 token to place ml.p3
action statement : when(tl1.p1*m==0; tl1.p2
when statement action
transition firing reversing
send 1 token to place tl1.p1
absorb 1 tokens from place tl1.p2
absorb 1 tokens from place ml.p3
absorb 1 tokens from place ml.p3
send 1 token to place ml.p3
send 1 token to place ml.p3
transition t4 is enabled
transition ml is enabled
transition ml firing
absorb 1 tokens from place m
transition t4 conflict with previous transition firing
system deadlock
terminate with p2=1, p4=1

104
Optimization of Query Processing in Distributed Database Systems

P. T. Chang
Institute of Information Engineering
Tamkang University, R. O. C.

J. S. Ke
Institute of Information Science
Academia Sinica, R. O. C.

Abstract:
The optimization of query processing is concerned to be one of the most difficult problems in the distributed database systems. The functional modules of distributed query processing are outlined. The criterion for the distinction of tree query from cyclic query graph is analyzed. A graphic methodology, called graph projection map, is well-defined and proposed to test the membership of queries and to find all the equivalent query expressions. Among these equivalent queries, we can obtain the optimal expression for the heuristic distributed query processing strategies.

Abstract:

Abstract:
The optimization of query processing is concerned to be one of the most difficult problems in the distributed database systems. The functional modules of distributed query processing are outlined. The criterion for the distinction of tree query from cyclic query graph is analyzed. A graphic methodology, called graph projection map, is well-defined and proposed to test the membership of queries and to find all the equivalent query expressions. Among these equivalent queries, we can obtain the optimal expression for the heuristic distributed query processing strategies.

Abstract:

Abstract:
The optimization of query processing is concerned to be one of the most difficult problems in the distributed database systems. The functional modules of distributed query processing are outlined. The criterion for the distinction of tree query from cyclic query graph is analyzed. A graphic methodology, called graph projection map, is well-defined and proposed to test the membership of queries and to find all the equivalent query expressions. Among these equivalent queries, we can obtain the optimal expression for the heuristic distributed query processing strategies.

Abstract:

Abstract:
The optimization of query processing is concerned to be one of the most difficult problems in the distributed database systems. The functional modules of distributed query processing are outlined. The criterion for the distinction of tree query from cyclic query graph is analyzed. A graphic methodology, called graph projection map, is well-defined and proposed to test the membership of queries and to find all the equivalent query expressions. Among these equivalent queries, we can obtain the optimal expression for the heuristic distributed query processing strategies.
一般而言，其 DDBS.QP 方法可分為直接結合法（Direct Joins Approach）與半結合法（Semi-Joins Approach）兩種，這已在（BERN 81）中討論。在此，吾人先勾劃出 DDBS.QP 之功能架構如下：

使用者查詢經由使用者介面轉譯成正規化的型式（canonical form）其查詢內函（qualifications）之型式為：\((A \land B \land \cdots) \lor (C \land D \land \cdots) \lor \cdots)\)
其中 \(A, B, C, D, \ldots \) 表查詢（query clauses）。
在本文中，吾人同（WONG 77）（CHHO 80）

（BERN 81）（YLCC 82）等文章，所討論的研究都是相交式（conjunctive form）之子查詢，且作結合之子查詢皆為等結合（equi-join）。

如例 2.1 所示。

例 2.1.1：
一查詢 \(Q(TL,q), TL \) 爲其目標列（Target List）

\[ q = (R1.A1 = R2.A2) \land (R2.A2 = R3.A3) \land (R3.A3 > 100) \]

\( q \) 為查詢 \( Q \) 相應式等結合之內函。

[ 定義 2.1 ] 遷移封閉（Transitive Closure）

對一查詢 \( Q(TL,q) \) 其相應式之內函 \( q \) 中，

若存在任二個子查詢 \((X = Y)\) 和 \((Y = Z)\)，則

由遷移律將產生第三個子查詢 \((X = Z)\)；

原來 \( q \) 中的子查詢加上所有由遷移律產生的子查詢，稱為此查詢 \( q \) 之遷移封閉，記為 \( q' \)。而 \( q' \) 對應之查詢為 \( Q' = (TL,q') \)。

例 2.1.1 查詢 \( Q(TL,q) \) 做遷移封閉後，

\[ Q' = (R1.A1 = R2.A2) \land (R2.A2 = R3.A3) \land (R3.A3 > 100) \land (R1.A1 = R2.A2) \land (R2.A2 = R3.A3) \land (R3.A3 > 100) \]

置底線的部分為由遷移律所產生的子查詢。從這裡可看出，作遷移封閉可產生所有相關的子查詢；因此，所有相關的單獨數子查詢，如例 2.1 中之 \((R1.A1 > 100), (R2.A > 100)\)

\( (R3.A3 > 100) \) 可先計算以減小將來做結合時關係的大小。（WONG 77）。
此問問之求問圖如圖3.1所示，為方便辨認，其在支邊上標出結合域（Join domains）名。

\[ \text{圖3.1：例3.1之求問圖G}_Q \]

【定義3.2】結合圖（Join Graph）：對一求問

\[ Q(TL,q) \]

\[ J_Q(V,J,E) \]

\[ A \]

\[ B \]

\[ C \]

\[ D \]

\[ E \]

\[ F \]

\[ G \]

\[ H \]

\[ I \]

\[ J \]

\[ K \]

\[ L \]

\[ M \]

\[ N \]

\[ O \]

\[ P \]

\[ Q \]

\[ R \]

\[ S \]

\[ T \]

\[ U \]

\[ V \]

\[ W \]

\[ X \]

\[ Y \]

\[ Z \]

\[ \text{同齊閉表現於結合圖上的意義即是對每一結合域子圖作完全運算；既蘇結合域子圖是參考到相同的結合域，故在不致令人誤解下，節點可不必再列出其結合屬性；圖3.3表示例3.1作筆記閉閉後之結合圖。} \]

【定義3.3】樹狀求問（Tree Query）：一求問

\[ Q \]

【或是其等效求問】之求問圖是為樹狀。

【定義3.4】環狀求問（Cycle Query）：不屬於

\[ Q \]

【樹狀求問之求問皆為環狀求問。】

設\( TQ \)為樹狀求問所成之集合，\( CQ \)為環狀求問所成的集合，由定義可知\( TQ \cap CQ = \emptyset \)且\( TQ \cup CQ \)是為所有求問所成的集合。

樹狀求問與環狀求問在DBMS與法上是有所差異，本文主要在對樹狀求問做一探討。

\[ \text{圖3.2：例3.2之求問圖} \]

\[ J_Q = J_Q^A \cup J_Q^B \cup \ldots \cup J_Q^F \]

\[ J_Q \]

\[ J_Q^A \]

\[ J_Q^B \]

\[ J_Q^C \]

\[ J_Q^D \]

\[ J_Q^E \]

\[ J_Q^F \]

\[ J_Q^A \]

\[ J_Q^B \]

\[ J_Q^C \]

\[ J_Q^D \]

\[ J_Q^E \]

\[ J_Q^F \]

\[ \text{等。} \]
第四節 樹狀詢問之測定

吾人將詢問類別之測定轉化成圖形理論上的問題。在未正式描述問題之前，先介紹下面幾個定義及定理：

[定義4.1]等效詢問(Equivalent Queries):
若二詢問 \( A_1(T_L,q_1), A_2(T_L,q_2) \)，對任一資料庫狀態，詢問所得之結果相同，則稱 \( A_1 \)、\( A_2 \)為等效詢問，記 \( A_1 \equiv A_2 \)。

[定理4.1]一詢問 \( Q(T_L,q) \) 其遞移封閉 \( Q^*(T_L,q^*) \) 是唯一的且 \( Q^* \equiv Q \)。

證明：很明顯地，由遞移律所產生的 \( Q^*(T_L,q^*) \) 與 \( Q(T_L,q) \) 是為等效詢問， \( Q^* \equiv Q \)。

由結合圖之定義可知，一詢問 \( Q(T_L,q) \) 與其結合圖 \( Q(T_L,q) \) 是一對一對應； \( Q^*(T_L,q) \) 的意即為對 \( Q(T_L,q) \) 之結合圖 \( Q^*(T_L,q) \) 為完全連結；因任一節點集合，其完全連結圖是唯一決定，所以 \( Q(T_L,q) \) 之遞移封閉 \( Q^*(T_L,q^*) \) 是唯一的。

口得証

[定理4.2]設二詢問 \( A_1(T_L,q_1), A_2(T_L,q_2) \)，若 \( q_1,q_2 \) 有相同的遞移封閉，\( q_1^* = q_2^* \)。

證明：\( \therefore q_1^* \equiv q_2^* \)。

又：\( q_1^* \equiv q_2^* \)。

故 \( q_1 \equiv q_2 \)。口得証

[定義4.1]投影圖映(Graph Projection Map): 順二圖形 \( G_1(V_1,E_1), G_2(V_2,E_2) \)，投影圖映 \( p \) 是一圖形耶繫之運算子；設 \( G_p(V,E) \) 爲投影圖映後之圖形，則 \( G_p = G \times G_2, V = V_1 \cup V_2, E = E_1 \cup E_2 \)。

例4.1: \( G_p = G_1 \times G_2, (V_1,E_1), (V_2,E_2) \)。

(4.1) 如圖4.1所示

圖4.1 例4.1之 \( G_p = G_1 \times G_2 \)
設 $V$ 表一節點集合，$SP(V_i)$，若非特別聲明，係指對 $V$ 所形成的完全連結圖 (complete graph) 之一葉閉叡 (Spanning Tree)。令 $G_1(V_i, E_1)$ 表對同一結合構之評選組於形成的訪問圖，由定理 4.2 可知 $SP(V_i)$ 所代表的評選與 $G_1$ 所代表的評選是等效的，且 $SP(V_1)$ 是為最少結合數的等效評選。設一評選 $Q(T,Q)$，$m$ 爲 $Q$ 中不同的 $G_1$ 個数，其訪問圖 $G_1(V_i, E_1)$，$V_i$, $E_1$, 也就是重復作 $G_1 G_1 G_1 G_1$, $i = 1,2,...,m$。於是，測定一評選 $Q$ 之訪問類別是否為樹狀（或環狀）的問題，吾人可覩之為：是否存在一 $SP(V_i)$ 可使得對每一結合構之訪問圖 $G_1(V_i, E_1)$ 存在 $SP(V_i) \subseteq SP(V_i)$，這也就是重覆作 $G_1 G_1 G_1 G_1$ $i = 1,2,...,m$，一開始 $G_1 G_1 G_1 G_1$，若最後等價訪問圖 $G_1$ 是否為樹狀。若最後之 $G_1$ 為樹狀，則此訪問圖 $Q$ 為樹狀訪問，不然則 $Q$ 為環狀訪問。吾人已知，對一連結的簡單圖 (simple connected graph)，若且惟若其支路數不等於結點數減一（即 $|E| = |V| - 1$），則此圖形是為樹狀（DEO 74）；由此，吾人可得下面之定理，用以測定一訪問之類別。

【定理 4.3】設 $m$ 為一訪問 $Q$ 不同的結合構子圖之個數，$V_1$ 表第 $i$ 個結合構之子圖訪問組所參考到的關係桿所成之集合，且惟若存在一組 $SP(V_1)$，$SP(V_2)$，$SP(V_m)$ 其相異支路數不等於 $|V_i| - 1$ 則此訪問圖 $Q$ 為樹狀訪問。

証明：因為重覆作 $G_1 G_1 G_1 G_1$，$i = 1,2,...,m$，一開始 $G_1 G_1 G_1 G_1$，最後之 $G_1 G_1 G_1 G_1$，故若存在一組 $SP(V_1)$，$SP(V_2)$，$SP(V_m)$，其相異支路數不等於 $|V_i| - 1$ 則此訪問圖 $Q$ 為樹狀訪問。反之，若 $Q$ 為樹狀訪問，$G_1 G_1 G_1 G_1$ 必為樹狀；於是即每一組 $SP(V_1)$，$SP(V_m)$ 其相異支路數不等於 $|V_i| - 1$ 之情形分小於和大於二種情形來討論：

(i) 小於 $|V_i| - 1$；

小於的情形不會發生，因為小於 $|V_i| - 1$ 之意為 $V_i$ 中某些節點在 $V_i$ 中未曾参考到，即 $V_i = \bigcup_{i=1}^{m} V_i$ 相矛盾。

(ii) 大於 $|V_i| - 1$；

$G_1 G_1 G_1 G_1$ 若自一組 $SP(V_1)$，$SP(V_m)$ 相異支路數皆大於 $|V_i| - 1$ 由樹狀圖之判別式 $|E| = |V| - 1$ 可知 $G_1 G_1 G_1 G_1$ 不為樹狀，故 $Q$ 為環狀訪問而非樹狀訪問。

因此，若每一組 $SP(V_1)$，$SP(V_m)$ 相異支路數皆不等於 $|V_i| - 1$ 則 $Q$ 為環狀訪問，而此命題是為 $n$ 唯若 $n$ 命題之逆命題（同義命題）。\[口得証口\]

由定理 4.3 吾人可直接導出測定樹狀訪問的方法，即計算評選 $Q$ 其 $m$ 個相異 $G_1$ 之 $SP(V_1)$，$SP(V_2)$，$SP(V_m)$ 相異支路數，若等於 $|V_i| - 1$ 則此訪問為環狀訪問。因 $G_1 G_1 G_1 G_1$ 即為例 3.1 測定之情形，為方便表示計算之情形，吾人設 $|V_i| - 1$，從 $V_i$ 做起，設 $G_1 G_1 G_1 G_1$ 之 $SP(V_1)$，$SP(V_2)$，$SP(V_m)$ 相異支路數不等於 $|V_i| - 1$；故例 3.1 之訪問為樹狀訪問。圖 4.3 是為環狀的例子。

第五節 等效訪問之最佳化

在第三節已提出等效訪問之方法，若可能不只一種。在本節，吾人以投影圖映可求出所有的等效訪問圖，從而求出最少層次中最長短的等效樹狀訪問圖。

吾人知道，對一結構子圖 $G_1(V_i, E_1)$，$SP(V_1)$ 是為最少結構數的等效結構子圖。對一樹狀訪問 $Q(T,Q)$，$m$ 為其相異結構子圖的

109
例3.1 之詢問Q

其週移封閉結合因子圖

\[ J_1^1, J_2^1, J_3^1, J_4^1 \]

\[ J_5^1 \]

\[ J_6^1 \]

\[ J_7^1 \]

累計第1次相異支邊數和

\[ \delta_1 = 1 \]

\[ \delta_2 = \delta_1 + 1 = 2 \]

\[ \delta_3 = \delta_2 + 1 = 3 \]

\[ \delta_4 = \delta_3 + 2 = 5 \]

\[ \delta_5 = \delta_4 + 1 = 6 > |V_Q| - 1 \]

\[ * \]

\[ G_p = SP(V_1皮) \to G_p, i = 1, 2, \ldots, 5, \]

\[ G_p \] 最後可得樹狀之例子（例2.1）

\[ G_p = SP(V_1皮) \to G_p, i = 1, 2, \ldots, 6, \]

\[ G_p \] 最後必不為樹狀之例子

\[ */ * \since \delta_1 \text{ is monotonic increasing w.r.t. } i, \text{ so,} \]

\[ V_6 \] would not be necessary to be tested. *
個數，為方便計算，欲 \(|V_1| \leq |V_2| \leq \ldots \leq |V_m|\)；一開始 \(G_p = G_{P(V_1)} \to G_p, 1 = 1, \ldots, m\) 於是最後之 \(G_p\) 為樹狀之所有可能情形，即為所有等效樹狀對象。

因樹形圖不等於滿結點，所以我們在 \(G_p = SP(V_1) \to G_p\) 時，\(SP(V_1)\) 支換的選擇是以已經出現被投影圖 \(G_p\) 之優先。下面定理 5.1 的定理 5.6 係討論一般化 (generalized) 第 1 次投影時，\(SP(V_2)\) 的 \(SP(V_1)\) 之所有可能情形。分析兩集合 \(V_1, V_2\) 之間的關係可能是 \(V_1 = V_2\) 或 \(V_1 \neq V_2\)；在 \(V_1 \neq V_2\) 中又可分為 \(V_1 \supseteq V_2\) 與 \(V_1 \subseteq V_2\) 兩種；在 \(V_1 = V_2\) 情形中又可再分為 \(V_1 \cap V_2 = V_{12}, |V_1| = |V_2|, V_1 = V_2\) 等三種情形。定理 5.1 和 5.2 係考慮 \(V_1 = V_2\) 與 \(V_1 \supseteq V_2\) 時，\(SP(V_2) \supseteq SP(V_1)\) 為樹狀之情形有一種。定理 5.4 考察 \(V_1 \subseteq V_2\) 且 \(V_1 \leq V_2\), \(SP(V_2) \supseteq SP(V_1)\) 為樹狀之情形有 \(m-2\) 種，

- \(\lVert V_2\rVert\) 定理 5.4 考察 \(V_1 \subseteq V_2\) 且 \(V_1 \leq V_2\), \(SP(V_2) \supseteq SP(V_1)\) 為樹狀之情形有 \(m = (n+1) \ldots (m-1)\) 種，

- \(\lVert V_1\rVert\) 定理 5.5 註明了當 \(V_1 \subseteq V_2\), \(\lVert V_1\rVert \geq 2\) 時，存在 \(SP(V_2) \supseteq SP(V_1)\) 為樹狀之情況必要條件為：當 \(V_1 \supseteq V_2\) 中每一點在 \(SP(V_1)\) 上不應由 \(V_1 \supseteq V_2\) 中任一點而相連結。當 \(V_1 \supseteq V_2\) 中存在某一點 \(V_1 \supseteq V_2\) 中另外一點在 \(SP(V_1)\) 上的路徑 (path) 須通過 \(V_1 \supseteq V_2\) 中之點時，\(SP(V_2) \supseteq SP(V_1)\) 必不為樹狀；這是 \(SP(V_2) \supseteq SP(V_1)\) 會產生唯一情形。定理 5.6 係針對符合定理 5.5 之必要條件，證明 \(SP(V_2) \supseteq SP(V_1)\) 之樹狀之情形有 \(h \cdot (h+1) \ldots (m-1)\) 種，\(h = |V_{12}|, m = |V_2|\)。

[定理 5.1] \(V_1, V_2\) 為兩點之集合 \(V_1, V_2\) 若且唯若 \(SP(V_2) = SP(V_1)\) 則 \(G_p = SP(V_1)\) \(\supseteq SP(V_1), G_p\) 為樹狀，若 \(SP(V_1)\) 為唯一一定，即 \(G_p = SP(V_1)\)。

證明：(i) "若 " 為存在性 (existence) \(SP(V_2) = SP(V_1)\)。\(\forall G_p, G_p = SP(V_1) U SP(V_2) = SP(V_1)\)。
情形為 \( V_2 \)之圖形所決定；由卡雷爾定理知 11
，不同的 \( \text{SP}(V_2) \) 有 \( m^2 \) 種，所以 \( G_p = \text{SP}(V_2) \) \( \rightarrow \text{SP}(V) \) 爲無環圖之情形有 \( m^2 \) 種。

(ii) 若 \( V_{12} = 1 \) 則 \( \text{SP}(V_2) \cap \text{SP}(V_1) \) 之圖形沒有連線存在，所以 \( G_p = \text{SP}(V_2) \rightarrow \text{SP}(V_1) \) 在 \( V_1 \) 上不產生閉環，故 \( G_p \) 有 \( n \) 種無環圖之情形

11
為 \( V_2 \)之圖形所決定；同上，\( G_p \) 爲無環圖之情形有 \( m^2 \) 種。

口得証口

[定理 5.4] \( V_1 \), \( V_2 \) 爲沉點所成之集合，\( V_1 \cap V_2 \)

\( |V_2| = m, |V_1| = n, m > n \), 對一切 \( \text{SP}(V_1) \)

而言，\( G_p = \text{SP}(V_2) \rightarrow \text{SP}(V_1) \)，\( G_p \) 爲沉點之情形有

\( n (n+1), \ldots, (m-1) \) 種。

證明：吾人先考慮 \( V_1 \cap V_2 \) 相交之部份。由定理

5.2 可知相交之部份使 \( G_p \) 爲樹狀之情形是唯一。

再考慮不相交之部份。若不相交之部份

\( \approx \) 有一節點 \( V^* \) 使 \( G_p \) 爲樹狀，\( \text{SP}(V_2) \) 之

情形為 \( V_1 \) 與 \( V_1 \) 任一點相連結，即

\( 0 \approx n \) 種情形；由此可推及不相交部份不一一節點的情形亦可視為第一次是在 \( n \) 點中選一點，

第二次是在 \( (n+1) \) 點中選一點，..直到不相交部份的最後一點是在 \( (m+n-m+1) \)

點中選一點。故 \( G_p \) 爲樹狀之情形有 \( n (n+1), \ldots, (m-1) \) 種。

口得証口

[定理 5.3] \( V_1 \cap V_2 \) 之節點所成之集合 \( V_{12} \),

\( V_{12} \cap V_1 = V_{12}, V_{12} \cap V_2 = \varnothing \) 若且唯若 \( V_{12} \)

中每一節點在已知之 \( \text{SP}(V_1) \) 與其他 \( V_{12} \)

中最近的節點之距離大於 1 （"小於"

不會發生，因 \( |V_{12}| > 2 \) 兩相異點之距離

不小於 1 ），則存在任一 \( \text{SP}(V_2) \) 使得 \( G_p =

\text{SP}(V_2) \rightarrow \text{SP}(V_1) \) 爲非樹狀。

設 \( v, v' \in V_{12}, v \in \text{SP}(V_1) \) 與其他 \( V_{12} \)

中最近的節點 \( v' \) 之距離為 \( d_{\text{min}} > 1 \)，此即在

\( \text{SP}(V_1) \) 之 \( v \) 的路徑 \( P_1 \) 之 \( v \) 之路徑 \( P_1 \), 必經 \( V_{12} \)

中的某些節點，\( |P_1| = d_{\text{min}} > 1 \)；又在 \( G_p \) 與

\( G_p = \text{SP}(V_2) \cup \text{SP}(V_1) \) 亦可經由 \( \text{SP}(V_2) \) 到 \( v' \) 此路徑為 \( P_2 \) 之 \( v \) 和 \( v' \) 間有 \( 1 \) 個節點存在，

在 \( G_p \) 存在一個經過 \( v, v' \) 之路徑，如圖

5.1b 所示。

若 \( P_1 \neq P_2 \) 與 \( d_{\text{min}} = 1 \), 與 \( d_{\text{min}} > 1 \) 相反，

口得証口

[定理 5.6] \( V_1 \cap V_2 \) 之節點所成之集合 \( V_{12} \),

\( V_{12} \cap V_1 = V_{12}, V_{12} \cap V_2 = \varnothing \) 若且唯若 \( V_{12} \)

中每一節點在已知之 \( \text{SP}(V_1) \) 與其他 \( V_{12} \)

中最近的節點之距離等於 1，\( G_p =

\text{SP}(V_2) \rightarrow \text{SP}(V_1) \) 所以吾人可作一
再展開 $V_2 - V_{12}$：此情形符合定理 5.4，

故 $SP(V_2)^P - SP(V_1)$ 為樹狀之情形形有 $h(\cdot h + 1)$
1) $(h+h-1)+h(\cdot h+1)...(h-1)$ 種。

綜合以上的定理，吾人可整理出下表 (表 5.1) 以示一般化之第 1 次投影圖映所有可能的情形。

集合 $V_2$、$V_1$，點表 $V_2$ 之節點，而點表 $V_1$ 之節點；(a) $G_p$ 為樹狀，(b) $G_p$ 不為樹狀。

EQTGEN：

0. 合併所有參考到完全相同關係樑之結合弧
子圖為一；當 $m$ 為相異結合弧子圖的個數
而其所參考到關係樑分別為 $V_1, V_2, \ldots, V_m$，
令 $Q = V_1 \cup V_2 \cup \ldots \cup V_m$。
1. $G = \emptyset, i = 1$。
2. 求出所有的 $SP(V_1)$，作 $G_p = SP(V_1)^P - G$，$G_p$ 為
SP(V_1) 與 G 之邊重疊最多之圖，$G_p$ 為弧狀
在 $G_p$ 支邊上對應 $V_1$ 之結合弧名。
3. 對各個產生的 $G_p$ 由每一個 $V_j, 1 \leq j \leq m$，測試
是否符合定理 5.5 之最後投影圖映的條件？

不相同的 $G_p$ 則令 $G = G_p$，繼續第 4 步。

相同的 $G_p$ 則於分支停止。
4. $i = i + 1$，重複作 2, 3 步；直到 $1 \leq m$ 或 $V_Q$ 每一
節點都已投影到。
5. 若 $1 \leq m$ 則結束。

若 $1 \leq m$ 但 $G_p$ 每一節點都已投影到現有產生
的 $G_p$ 支邊上每一個 $V_j, 1 \leq j \leq m$，對應之結
合弧名，然後結束。

現在我們以例 3.1 之問題 Q 為例子，用
EQTGEN 來求出 Q 之所有等效弧狀之圖。如圖
5.2 所示，令 Q 之本文尚未投影到的節點（關係
樑），黑點表已投影到的節點。因於測試 Q 中
$V_1 \cup V_2 \cup \ldots \cup V_m$，可參考到完全相同的關係樑，故這兩個
結合弧子圖可合併為一考慮，以 $V_2$ 表之。一開始
$G = \emptyset$，如 (0) 圖所示。$i = 1$，作 $G_p = SP(V_1)^P - G$，
因於 $V_1$ 與 Q 參考到同一關係樑，$SP(V_1)$ 為子
所以 $G_p$ 有一個；支邊圖上對應 $V_1$ 之結合弧
名 C，如 (1) 圖所示。$i = 2$，作 $G_p = SP(V_2)^P - G$。
再因 $V_1$ 與 Q 參考到同一關係樑，$SP(V_2)$ 為子
所以 $G_p$ 有一個；支邊圖上對應 $V_2$ 之結合弧
名 B，如 (2) 圖；支邊圖上對應 $V_2$ 之結合弧名 B，如 (2) 圖；支邊圖上對應 $V_2$ 之結合弧名 B，如 (2)
表 5.1：$G_p = SP(v_2) \cup SP(v_1)$之情形

<table>
<thead>
<tr>
<th>定理條件</th>
<th>示意圖</th>
<th>相異樹狀$G_p$之個數</th>
<th>註</th>
</tr>
</thead>
<tbody>
<tr>
<td>$v_1 = v_2$</td>
<td>$v_1, v_2$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$v_1 \neq v_2$</td>
<td>$v_1, v_2$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$v_1 \cap v_2 - v_{12}$</td>
<td>$v_1, v_2$</td>
<td>$m-2$</td>
<td>$m =</td>
</tr>
<tr>
<td>$</td>
<td>v_{12}</td>
<td>\leq 1$</td>
<td>$v_1, v_2$</td>
</tr>
<tr>
<td>$v_1 \cap v_2 = v_{12},</td>
<td>v_{12}</td>
<td>\geq 2, v_{12}$</td>
<td>$v_1, v_2$</td>
</tr>
<tr>
<td>$v_1 \cap v_2 - v_{12},</td>
<td>v_{12}</td>
<td>\geq 2, v_{12}$</td>
<td>$v_1, v_2$</td>
</tr>
</tbody>
</table>

圖5.2：$G_p = SP(v_5) \cup \{2\}, G_p$必為環狀圖。
對應例 3.1 之問問 Q
其選移封閉結合城子圖

圖 5.3：例 5.1 詢問 Q 之所有等效樹狀詢問圖的求法，
圖 (12) - (17) 是 Q 之所有等效樹狀。

i=4, sp(v_q)^E(4) 及 sp(v_q)^E(5) 為樹狀之情形由
定理 5.3 可知各有 3 種，即 (6) - (11) 圖。 

1 = 5，因為 (6) - (11) 圖上已經投影到 v_q 的每一
節點，再由定理 5.2 可知 sp(v_q) 投影到每一
(6) - (11) 圖，為樹狀之情形均只有一種，分別
為 (12) - (17) 圖。於是，(12) - (17) 圖即為 Q 之
所有等效樹狀詢問圖。

設 R1 為發問地，則 (12) - (17) 圖中，(15)
與 (17) 圖是為最少層次中且最長類型之詢問圖
（參見右圖）。

圖 5.4：例 3.1 詢問之所有等效詢問中，
最少層次且最長類型之詢問圖。
於是乎對上下結合程序而言，Q之最佳等效
詢問便應要比較圖5.4中斷開圈的部分；如何
預估這一部分作半結合後的結果，是進一步最
佳化尚待解決的問題。

第六節 結 論

本文一開始便明確地定下DDBS.QP之功能
架構，及DDBS.QP核心問題之所在；也簡明地
討論直接結合法與半結合法用來處理樹狀詢問
與環狀詢問。

逐移封閉法一詢問轉化成最多餘(redu-
dant)的唯一型式。從此最多餘的型式，吾人
以投影圖映的觀念，提出一測定樹狀詢問之定
理，利用該定理將使得詢問類別之測定變得更
單純、快速；更合理地，本文以投影圖映可求
出所有的等效詢問，從而依吾人之策略以
得最佳之等效詢問；較之目前所有樹狀詢問的
測定法一任意的求出u—u等效樹狀詢問，以
之替代詢問圖非樹狀而實為樹狀詢問之詢問一
顯然要合理許多。

雖然在本文中u最少層次且最長鏈型u之
最佳等效詢問的策略仍很主觀，不過這顯是個
開始，相信今後對討論最佳等效詢問，必定將
以本文一些定理為基礎。

參 考 文 獻

[柯志昇 81]| n 分散式資料庫系統之詢問處理
[ADIB 80]| M.Adiba et. al., "POLYTRIME: An
Experience in Distributed Data
Base System Design and Implemen-
tation," Distributed Data Bases,
C. Delobel and W. Litwin eds.,

[BECH 81]| P.A. Bernstein and D.M.Chiu,
"Using Semi-Joins to Solve Rela-
tional Queries," J.ACM, Vol.28,

[BEGO 79]| P.A.Bernstein and M.Goodman,"Full
Reducers for Relational Queries
Using Multi-Attribute Semijoins,"
Proc. Computer Network Symposium,

[BEGO 81]| P.A.Bernstein and M.Goodman,
"Power of Naturnal Semijoins,"
SIAM J. COMPUT, Vol.10, No.4,Nov.

[BERN 81]| P.A.Bernstein et al., "Query Pro-
cessing in a System for Distrib-
uted Databases(SDD-1)," ACM-
602-625.

[CEME 80]| W.Cellary and D.Meyer, "A Multi-
Query Approach to Distributed
Processing in a Relational Data-
base Management System," same

[CHHO 80]| D.M.Chiu and Y.C.Ho, "A Method-
ology for Interpreting Tree
Queries into Optimal Semi-Join
Expressions," Proc. ACM-SIGMOD

[DENO 74]| N.Deo, "Graph Theory with Appli-
cations to Engineering and Com-
puter Science," Prentice-Hall.

[EVEN 79]| S.Even, "Graph Algorithms,"

[HEYA 79]| A.R.Heyner and S.B.Yao, "Query
Processing in Distributed Data-
base Systems," IEEE Trans. Soft-
ware Eng., Vol.SE-5, No.3, May
1979, pp.177-187.

Identifying a Class of Database
Queries That Can Be Processed
Efficiently," IEEE 4th Interna-
tional Computer Software and

[BOYO 76]| E.Woog and Yossefi, "Decomposi-
tion -- A Strategy for Query

(WONG 77) E.Wong et.al., "Retrieving Dispered Data from SDD-1: A System for Distributed Databases," the Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.
