CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

Canada

DESIGN OF MULTILAYER INFORMATION SYSTEMS

by

Pierre Cousineau

A thesis
presented to the School of Graduate Studies and Research
of the University of Ottawa
in partial fulfillment of the
requirements for the degree of
Master of Applied Science
in
Electrical Engineering

OTTAWA, Ontario, 1983

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

The University of Ottawa requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

ABSTRACT

This thesis presents a multilayered distributed query architecture applicable to multiple user information systems where user information requirements are well defined. This allows, via a small duplication of data, the segmentation of the system into autonomous functional areas. The result is a distributed system that can be easily modified and upgraded. It has in addition the advantage that it allows for easy inclusion of data access security and various operational safeguards needed in multiple user systems.

In order to show the applicability of such a distributed architecture, the design of one facet of a hospital information system, namely a patient medical record system, is presented. The operation of its critical element (data base manager) was simulated.

ACKNOWLEDGEMENTS

I am most grateful to my supervisor, professor Moshe Krieger, for his meaningfull advice and guidance during the course of my research. The many helpfull discussions confirmed the soundness of the ideas reflected in this thesis.

I would also like to thank my family, friends, and colleagues who made this research more enjoyable. Without them I would not have found the necessary drive.

Furthermore, I wish to acknowledge the financial support of the National Science and Engineering Research Council.

CONTENTS

ABSTRACT iv

ACKNOWLEDGEMENTS v

Chapter page

I. INTRODUCTION 1

II. DATA BASE SYSTEM OVERVIEW 4

DEVELOPMENT OF DATA BASE SYSTEMS 4
Evolution of data base concepts 4
Basic terminology 6
INTEGRATED DATA BASE 8
Characteristics of integrated data bases . . 9
Data base models 13
Link structure models 14
Relational model 17
CENTRALIZED DATA BASE SYSTEMS 18
Characteristics of centralized data base . . 18
Physical data base design 20
DISTRIBUTED DATA BASE SYSTEMS 24
Characteristics of distributed data base systems 26
Fundamental problems with DDBMS 30
DATA BASE MACHINE 33
CONCLUSION 37

III. HOSPITAL INFORMATION SYSTEM OVERVIEW 38

CHARACTERISTICS OF HOSPITAL INFORMATION SYSTEM 38
COMPUTERIZED MEDICAL RECORD SYSTEM 41
EXAMPLES OF EXISTING HIS 44
COSTAR 44
Medical record organization 45
User interface 46
PROMIS 49
Medical record organization 49
Guidance system 50
System implementation 53
CONCLUSION 54

IV. A DISTRIBUTED QUERY ARCHITECTURE FOR A HOSPITAL INFOMATION SYSTEM 58

SCOPE OF APPLICATION 59
User types 59
System information flow 60
DATA BASE ORGANIZATION 62
SYSTEM ARCHITECTURE AND OPERATION 65
SYSTEM ELEMENTS 74
On-line archival data base subsystem 74
Accessing requirements 74
Record organization 75
Implementation aspects 76
Current data base subsystem 80
Access requirements 80
Record organization 80
Implementation aspects 81
Work stations 83
Data base manager 84
Bus controller 85
SYSTEM INTEGRATION 86
CONCLUSION 86

V. IMPLEMENTATION ASPECTS OF THE DATA BASE MANAGER . . 88

DATA BASE MANAGER FUNCTIONS 88
Graphical notation 89
Definition of the data base manager operation 94
IMPLEMENTATION OF THE DATA BASE MANAGER . . . 112
Data base manager organization 113
Implementation aspects of the data base manager's units 118
CONCLUSION 122

VI. SIMULATION 123

SIMULATION DESCRIPTION 123
ACTIVITIES OF TASK DRIVERS 1 AND 2 124
Activity graphs of task driver "1" 126
Activity graphs of task driver "2" 133
SIMULATION EXAMPLES 139
Single request 139
Multiple requests 145

VII. CONCLUSION 151

Appendix page

A. SIMULATION LISTING 153

REFERENCES 186

LIST OF FIGURES

Figure		page
1.	Wording describing the logical data base	8
2.	Data base management approach to data organization	10
3.	Three levels of abstraction	13
4.	Example of a Hierachical data base	15
5.	Example of a network data base	16
6.	A relational data base	18
7.	Traditional data base system	20
8.	An example of a distributed data base system	25
9.	Dimensions that characterize the degree of decentralization	27
10.	Organization of a cellular device	36
11.	COSTAR record format	47
12.	COSTAR status report	48
13.	Data base display	52
14.	Architecture of PROMIS	54
15.	Data base organization	65
16.	System architecture	66
17.	Data base manager subsystem organization	67
18.	Archival medical record organization	77
19.	Current medical record organization	82
20.	Data flow	90

21. Control directed graph 91
23. Sequencing operators 93
23. Conditional 94
24. Admitting a new patient from admission 96
25. Admitting a former patient from admission 97
26. Admitting a new patient from emergency 100
27. Admitting a former patient from emergency 101
28. Service request 103
29. Update request 105
30. Additional information request 107
31. Releasing a patient 109
32. Transferring a patient 111
33. Data base manager organization 115
34. Bus organization 117
35. Task driver organization 119
36. Wait operator 125
37. Admitting a patient from admission 126
38. Admitting a patient from emergency 127
39. Service request 128
40. Update request 129
41. Additional information request 130
42. Transferring a patient 131
43. Releasing a patient 132
44. Admitting a patient from admission 134
45. Admitting a patient from emergency 135
46. Additional information request 136
47. Transferring a patient 137

48. Releasing a patient 138

Chapter I

INTRODUCTION

The extensive (wide) use of multi-user computerized information systems led to the re-evaluation of a number of major points in the design of data base systems. To name a few, data security, system reliability and cost performance. Most of the researchs in this area consider these problems only for high performance gereral-purpose data base systems. With the present day wide varieties of computerized information systems for the non computer professional (lawers, doctors) and general public, it is imperative to consider the above problems with respect to special-purpose 'low cost' data base systems.

This thesis introduces a layered data base architecture applicable to multi-user information systems where user information requirements are restricted to a small portion of the overall data base. This type of application arises in a number of systems such as: hospital information system, banking system, school information system. The suggested data base system is a special-purpose distributed data base system where each user has its own dedicated user-oriented sub-data base. To overcome the data consistency, the data security and the communication problems of completely dis-

tributed system, a central data base with very restricted accessing protocol is maintained. This duplication also provides for data reliability.

The layered approach requires some duplication of data and a high distribution of processing. Today, this is highly cost effective because of the declining cost of mass storage and the availability of inexpensive LSI hardware.

To show the applicability and feasibility of this distributed architecture, the design outline of a hospital information system is presented. Specifically, we consider the design of one facet of a hospital information system, namely, a patient medical record system used for medical care, financial and administrative needs.

In order to provide the necessary background, chapter two presents the basic data base concepts and the current trends in the major areas of the data base system research. In chapter three, the major characteristics of a hospital information system are discussed. In order to demonstrate what has been achieved in this area, the description of two highly regarded hospital information system; PROMIS and COSTAR, are given. Chapter four presents the design outline of a patient medical record system using the layered data base approach. This is done by specifying the data base organization and defining a system architecture which supports the above data base organization. The critical element of this

system is the data base manager. The implementation of the data base manager is described in chapter five. In the last chapter, a simulation demonstrating the feasibility of the system is presented.

Chapter II

DATA BASE SYSTEM OVERVIEW

The purpose of this chapter is to introduce basic data base concepts and to provide a rough categorization of data base systems. The chapter begins with a brief discussion of the development of data base systems. It then continues with a description of the major aspects of an integrated data base. Finally, the implementation of an integrated data base is presented in terms of centralized, distributed data bases and data base machines.

2.1 DEVELOPMENT OF DATA BASE SYSTEMS

2.1.1 Evolution of data base concepts

In the late fifties and early sixties, government organizations and large businesses began to store data on computer-accessible files. Time consuming tasks that were previ-

ously accomplished manually (e.g. payroll, inventory), could now be performed automatically by computer systems at very high speeds. These systems used magnetic tapes as storage devices and the software provided little support for accessing the information stored on these devices (i.e. it executed only the input/output operations of the storage devices). Also, the average access times of magnetic tapes were very poor. Subsequently, other types of storage devices such as disks emerged. These devices permitted more direct access to the stored information; it is faster to record or retrieve non consecutive information from a disk drive than from a tape drive. New software also evolved to provide support for manipulating and organizing files on these direct-access storage devices. For example, simple file organizations such as relative, indexed sequential, and others were provided . These software systems were referred to as file systems. These file systems isolated the users from the storage device details, but they favored a close dependency between the data and the programs that used the data. As a result of this dependency, the following problems arose:

(1) Since most files were used and structured for one application, the data had to be stored redundantly in many files, thus creating the problem of keeping the data consistent.

(2) It was impossible to change the file organization and access techniques without affecting the application program.

(3) In many cases, to service a new request, a new application program had to be written to access several private files.

To overcome the above limitations, the concept of an integrated data base was introduced during the late sixties and early seventies. In this approach, all data are kept in a central pool where it can be shared by all users. Furthermore, the concept of data independence was introduced. Data independence means that the physical organization of the data base could be changed without affecting the application programs. With the declining cost of memories and processing hardware, the current trend in the development of data base systems is moving toward multi-user distributed systems [DEEN 82].

2.1.2 Basic terminology

At this point, we shall introduce the basic data base terminology. It should be noted that different terms for describing files and data bases have been used by various sources (e.g. companies, research institution). The following definitions appear to be the most accepted in the liter-

ature [MART 77, TEOR 82] and were conceived by the CODASYL Data Base Task Group.

Data Items - Smallest unit of named data, in many instance it is referred to as a field.

Record - Named collection of data items.

File - Named collection of all occurrences of a given type of record.

Data Base - Collection of the occurrences of multiple record types (files) containing all necessary relationship information between records and/or between data items.

It should be noted that the above definitions describe the elements of the logical data base, and not the elements of the physical data base. These logical elements are illustrated in figure 1.

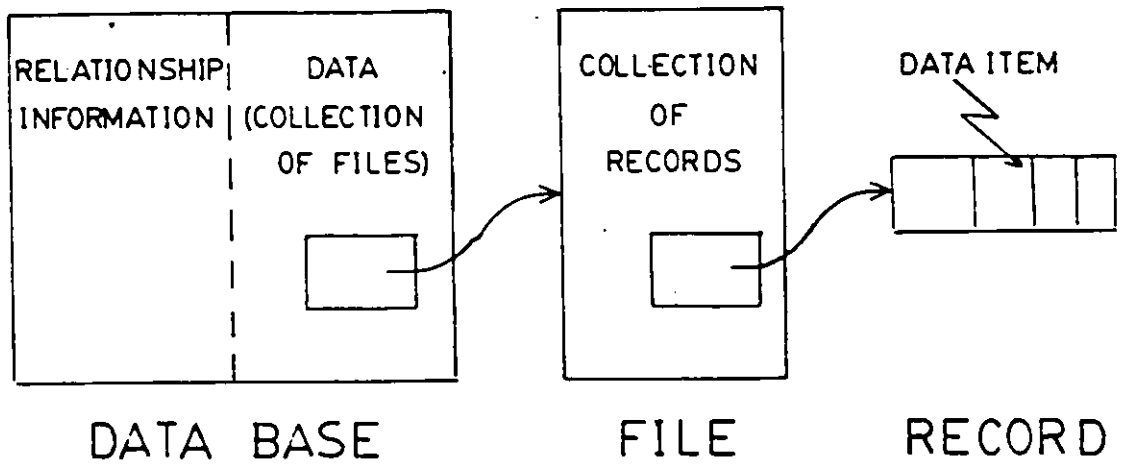

Figure 1: Wording describing the logical data base

2.2 INTEGRATED DATA BASE

As mentioned previously, to overcome the limitations of file systems, the data must be integrated and its physical structure must be transparent to the users. An integrated data base may be defined as follow: it is an integrated collection of data serving multiple applications where the data are stored so that they are independent of the programs that will use them.

2.2.1 Characteristics of integrated data bases

Basically, integration of the data is achieved by defining the structure of the data storing it, and allowing user access via a general-purpose software called Data Base Management System (DBMS). In general, the user interface with the DBMS is provided by way of two languages [YAO 78]: "Data Definition Language", which is used to define the logical structure of the data, and "Data Manipulation Language", which allows insertion of new data, retrieval and deletion of old data. Furthermore, there are many other functions that must be carried out by a DBMS, such as [ULLM 80]:

(1) Providing integrity mechanisms for maintaining the accuracy of the data.

(2) Providing security mechanisms to prevent unauthorized access.

(3) Providing concurrency mechanisms to allow users to access the data base at the same time.

(4) Providing facilities to reconstruct the data base after a hardware or software error.

Figure 2 shows the data base management approach to data organization.

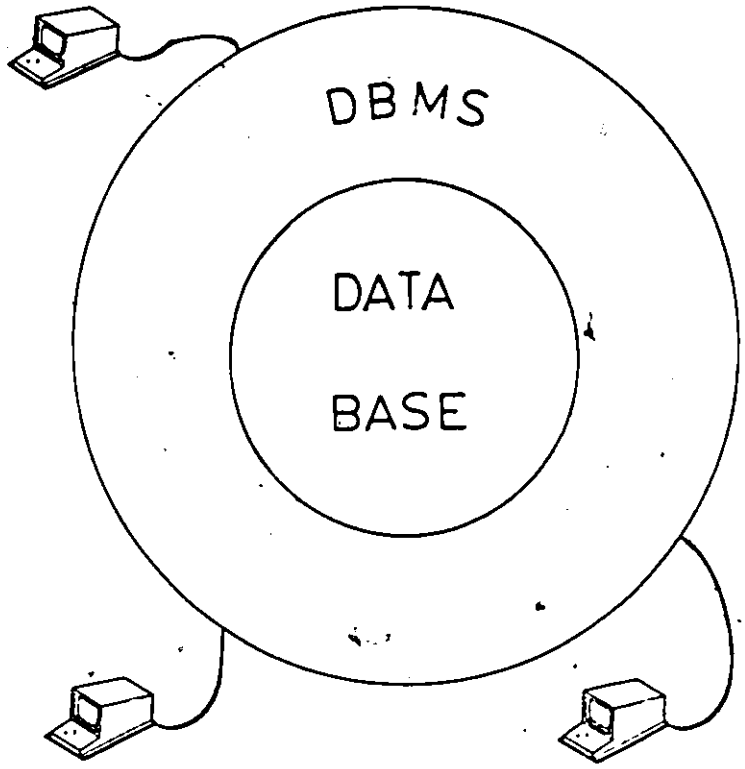

Figure 2: Data base management approach to data organization

The major objectives of the DBMS are :

(1) To provide physical data independence; that is the users need not be aware of the physical data base organization.

(2) To provide logical data independence by allowing the users to define a subset of the logical data base. As a result, the logical structure of the data base can be expanded without modification to the application programs. Also logical data independence restricts the application program to view only the portion of the data base which is relevant to it.

(3) To provide a high level data manipulation language in order to make program development easier and faster.

In order to achieve the above objectives, one has to recognize three levels of abstraction in the data base organization. [MART 77, TEOR 82, ULLM 80]:

Conceptual level - The conceptual level represents the logical organization of the entire data base. More specifically, it defines the types of the data items and specifies the relationships between them. Therefore, it provides a framework for inserting the values of the data items.

External level - The external level is concerned with the way in which the data is viewed by the users. At this level many views can be defined where each is an abstraction of a portion of the conceptual data base. In general, the difference in the level of abstraction between views and the conceptual data base is not great.

Physical level - The physical level represents the physical structure of the logical data base (i.e it is the implementation of the conceptual data base). This level can be subdivided into several levels ranging from files down to the actual physical addresses on storage devices. This is the level which has the greatest impact on the data base system performance.

The interrelationships between the three levels are shown graphically in figure 3. The conceptual and external level corresponds to the logical data base and the physical level corresponds to the physical data base. In the next subsection, the different approaches to the design of a logical data base will be described. The major issues concerning the design of the physical data base are postponed to the next sections since they are dependent on the environment on which the data base is implemented.

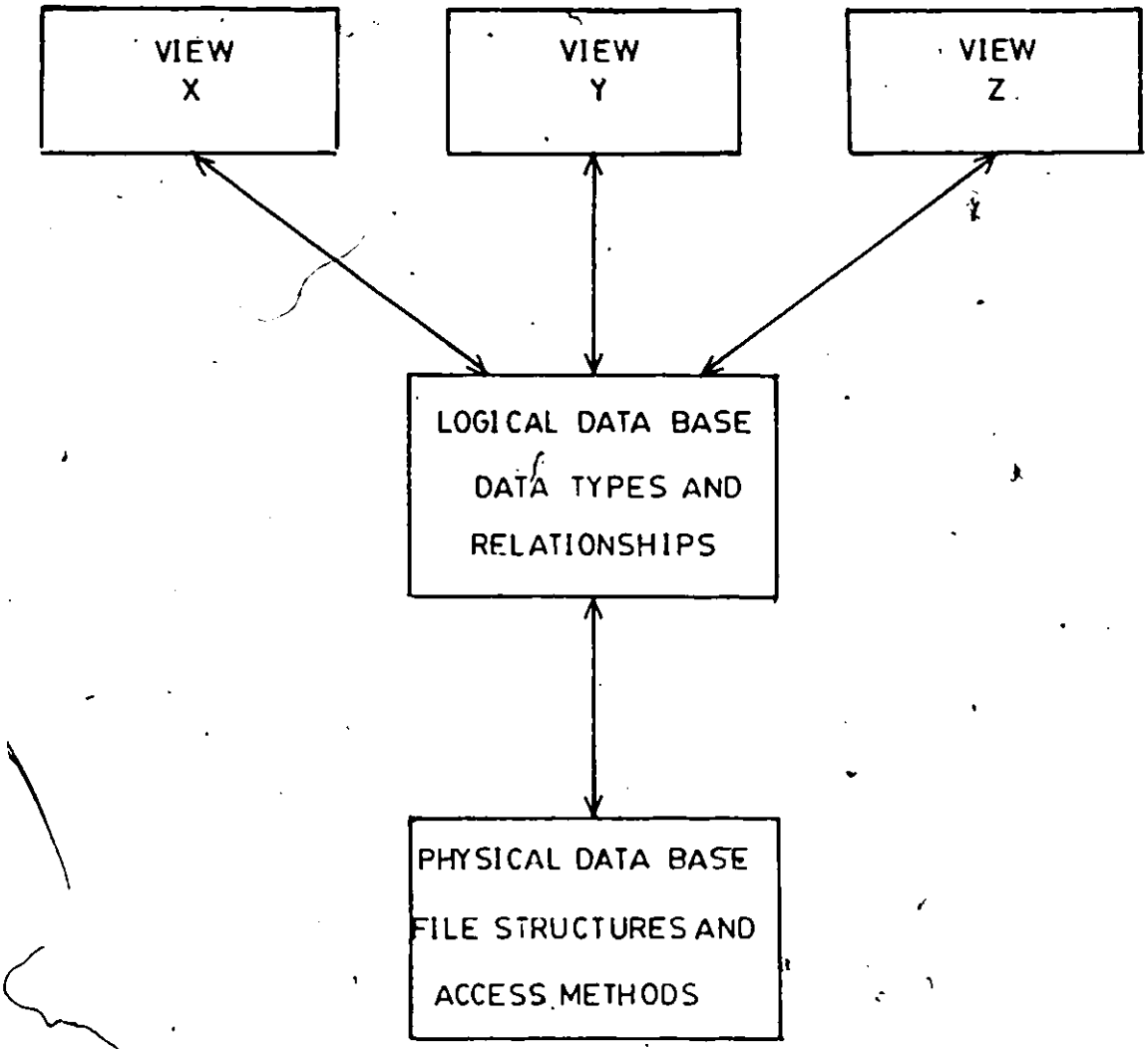

Figure 3: Three levels of abstraction

2.2.2 Data base models

In the following, we will investigate the various methods for defining the data items and structures needed to model the information requirements of a given enterprise. These methods are often referred to as data models and serve as a

basis for implementing the Data Manipulation Language and Data Definition Languages of any DBMS. Among the various data models proposed in the literature, two types of data model are widely available today [CHAM 79]: link structure models and relational model.

2.2.2.1 Link structure models

Link structure models are structures in which the relationships between record types can be graphically represented by arcs. The two basics link structure models are:

Hierarchical - In this model, the relationships between the records types are represented in a tree structure . A hierarchical data base structure can be defined as a collection of record types in which there is one special record type called the root. The others are referred to as dependent record types. Each dependent record type has a parent (record type) and each parent has at least one child record type. Note that for one occurrence of a record type, there may be any number of occurrences for each of its children. Thus, only one-to-many relationships are allowed. The description of a simple data base using this approach is shown in figure 4. This is a very simple structure and it is difficult to map a complex real-world situations with this model.

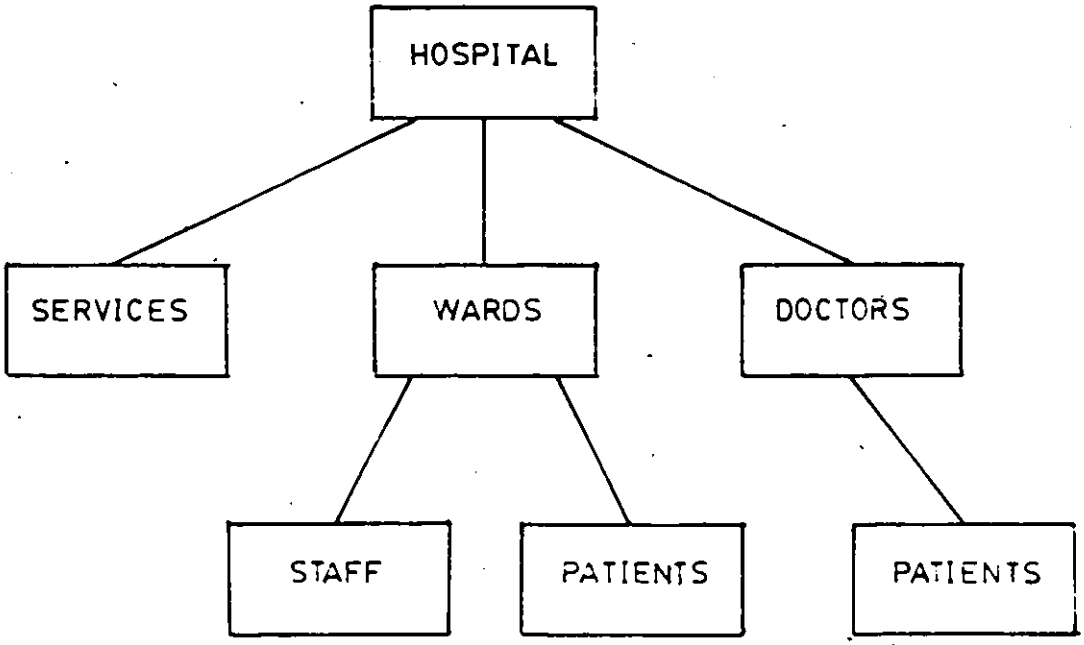

Figure 4: Example of a Hierachical data base

Network - The network model is an extension of the hierarchical model which allows more directly the modelling of many-to-many relationships since it allows the representation of many-to-one and one-to-many relationships. Figure 5 illustrates the description of the hierarchical data base of figure 4 using the network model. Here the "patient" record type is not duplicated and a "patient" record can be accessed either via doctors or wards. The

main advantage of this model is its efficiency in performing basic DBMS operations, since different access paths can be defined at data base creation (a record can be accessed in different ways). The major drawback of this approach is the complexity of the data model and of the data manipulation language.

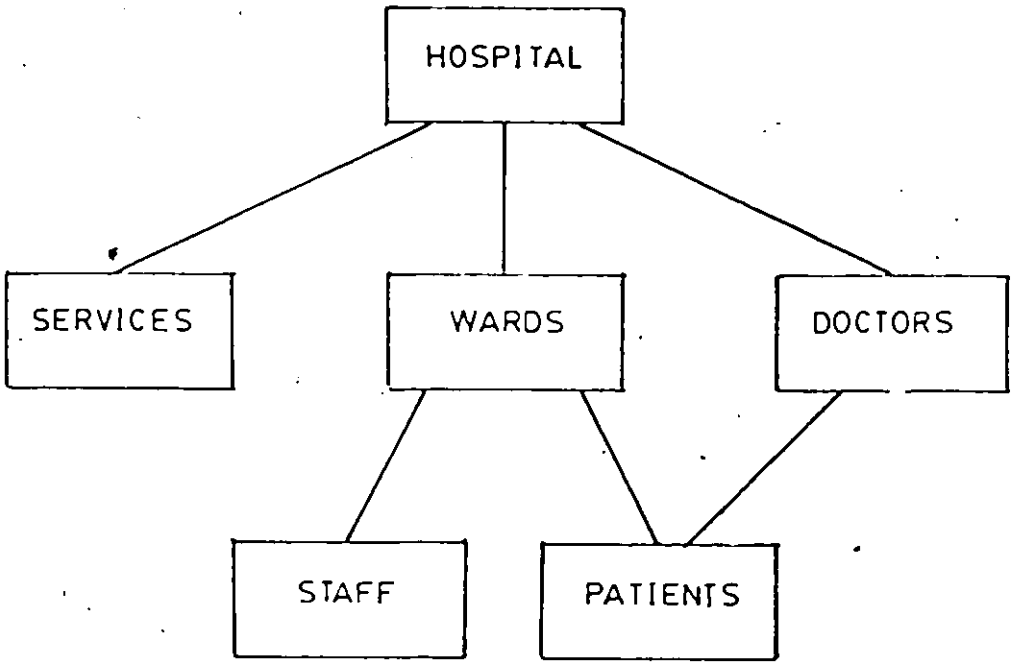

Figure 5: Example of a network data base

2.2.2.2 Relational model

The basic concepts to the relational model were introduced by E.F. Codd [CHAM 76]. In this model, the data is organized in a number of two-dimensional tables. A table contains X table entries each having Y fields. The relationships between records which may belong to the same table or to seperate tables, are represented by a common value for a given field. This model shields the user from the complexity of the storage structure and access paths, and thus requires a powerfull set of operators for data manipulations. The main drawback of this model is that having no predefined access paths, exhaustive search must be performed to satisfy a query, thus access efficiency is reduced. Note that in current implementation of the link structure models, association among records is made via pointers, where as in the relational model, association among records is made by the same value of a field [CHAM 79]. Figure 6 shows a data base expressed in the relational model.

DOCTOR RELATION	
DOCTOR	PATIENT
D1	P1
D1	P2
D2	P3

WARD RELATION		
WARD	PATIENT	STAFF
W1	P1	PIERRE
W2	P2	PAUL
W3	P3	GUY

Figure 6: A relational data base

2.3 CENTRALIZED DATA BASE SYSTEMS

Centralized data base systems are data base systems which are implemented at one site. In general, these data base systems operate on a general-purpose computer. Most existing large data base systems are implemented in this fashion. In this section, we will discuss the design issues encountered in the implementation of an integrated data base on a general-purpose computer.

2.3.1 Characteristics of centralized data base

At present, most large data base systems use moving-head disks as the storage elements. These disks can be regarded

as being a linear array of fixed size blocks with a block being the smallest addressable unit. A block usually contains one or more logical records plus the information associated with the storage format of the record. As shown in figure 7, a traditional centralized data base system consists of the application software, operating system, DBMS software and the various files (i.e. conceptual schema, subschemas, data, system files, etc.) [KERR 79]. Users interface to the data base management system via the operating system. The operating system controls all accesses to the storage devices and in many cases, file systems are used to allocate the storage space and to provide the access methods [WIED 81].

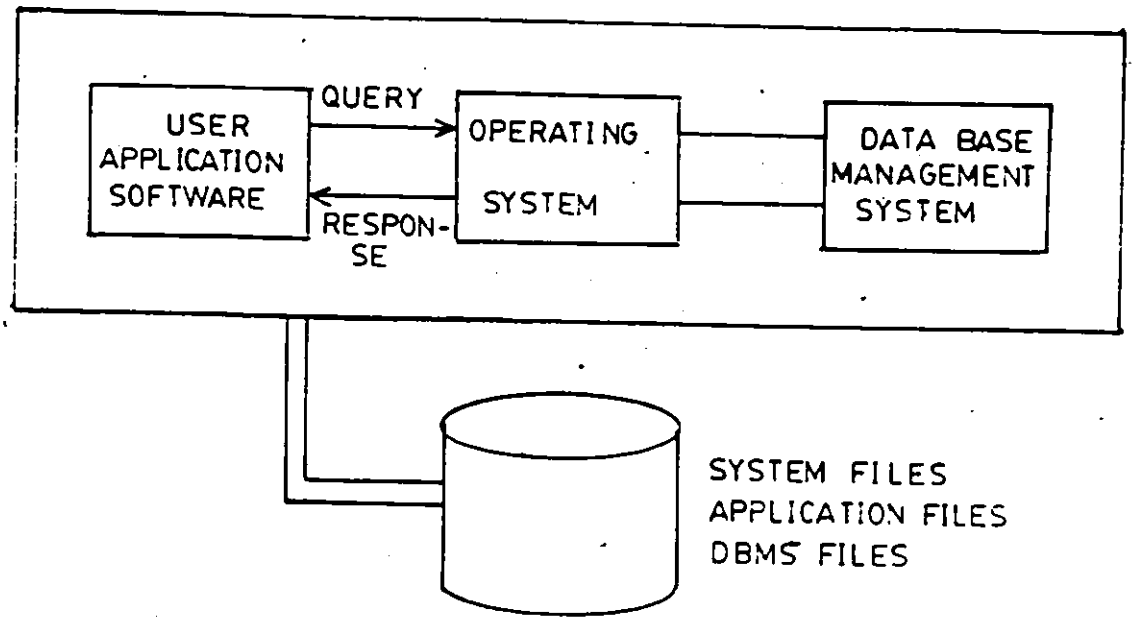

Figure 7: Traditional data base system

2.3.2 Physical data base design

The major factors that must be taken into account in the physical data base design are :

(1) Machine Characteristics:

- Constraints and characteristics of the physical media; storage space, seek times, etc.

- Machine environment; for example, DBMS processing attempts to retain critical pages such as indexes. However, this goal is difficult to obtain if the underlying system supports a virtual memory system.

(2) Work Requirements:

- Random or sequential processing
- Frequency of reference
- Response-time requirements
- Rate of records insertions and deletions
- Multiple-key or single-key retrieval

In most present day centralized data base systems, the physical structure tends to be a collection of files (single-record-type structure). The first problem encountered in mapping the logical data base structure onto the data storage devices, is to define the proper files. This is not a trivial task for the following reasons: First, one must determine the pointer structures among files for representing the various relationships present in the logical structure. Secondly, one may have to subdivide the logical files to improve access time. There are two basic ways of subdividing files [SCHK 78]: (1) partionning the records into one or more subfiles if the attributies of a file have different access activites, and (2) grouping the records according to their access activity.

Having defined the various files, the next step consists of selecting the proper structure for each file.

Given a collection of records, there are many trade-offs encountered in the selection of a file structure. For instance, an organization which provides high flexibilty for searching, will require more storage space and may lead slower access time. File structures can be classified into two groups: single-key and multi-key organizations.

Single-key organizations are structures which allow to access a record from a file according to a primary key and can be divided into sequential or random organizations. In sequential organization, records are stored according to a specific sequence and can be accessed either by scanning the file or by search mechanisms (e.g. binary search). In random organization, the primary key is used to specify the hardware address on the storage media. Two of the most common random organizations are indexed sequential and hashing organizations [MART 77].

(1) Indexed Sequential Organization

This file organization has the property that records can be accessed either sequentially or randomly through an index table. The records are stored sequentially and, as a rule, the index table

does not contain a reference to every records but to blocks of records. This organization allows to access efficiently the data both in a sequential and in a random format. However, the major drawback is that it requires complex maintenance operations.

(2) Hashing

The record address is calculated by inputting the key value into a hashing function which converts it into a physical address. This method provides efficient retrieval and there are no major problems resulting from insertions and deletions. However, it is inefficient in the case of sequential accessing and has poor storage utilization.

Multi-key organizations are structures which allow to process queries which involve more than one key. Most common multi-key organizations achieve the above by linking records containing the same value for a given field via pointers. If the pointers are embedded in the record, the organization is referred to as a multilist file. If the pointers are kept in an index table, it is referred to as an inverted file. The inverted file is more efficient for handling complex queries which involve long decision functions since no chains need to be followed. The major drawback of this organization is the maintenance of the index table.

There are many problems associated with a centralized data base systems. The major ones are performance, reliability and expandability. Performance is due to the fact that data can not be processed directly on the storage devices. As seen above, to access efficiently the data, complex data structures must be created. However, these structures introduce the following drawbacks [MYER 82]: (1) software complexity is increased, (2) additional storing space is required, and (3) significant software overhead is introduced when adding or updating data. At present, data base machine is a substantial resource for dealing with the above problem. Distributed data base systems are another emerging technology which provides many solutions for the above problems. Both distributed data base systems and data base machines are described respectively in the following sections.

2.4 DISTRIBUTED DATA BASE SYSTEMS

With the present-day declining cost of memories and processing hardware, it becomes cost effective to implement an integrated data base over a distributed system. At present, most existing distributed systems corresponds to loosely coupled systems (computer network) as shown in figure 8.

In this section, first, we will describe the characteristics of distributed data base systems. This will be done by

describing "what is a distributed data base system", enumarating its advantages over a centralized data base system, and describing the different approaches for implementing these types of data base systems. Finally, the fundamental problems peculiar to a distributed data base system are discussed briefly.

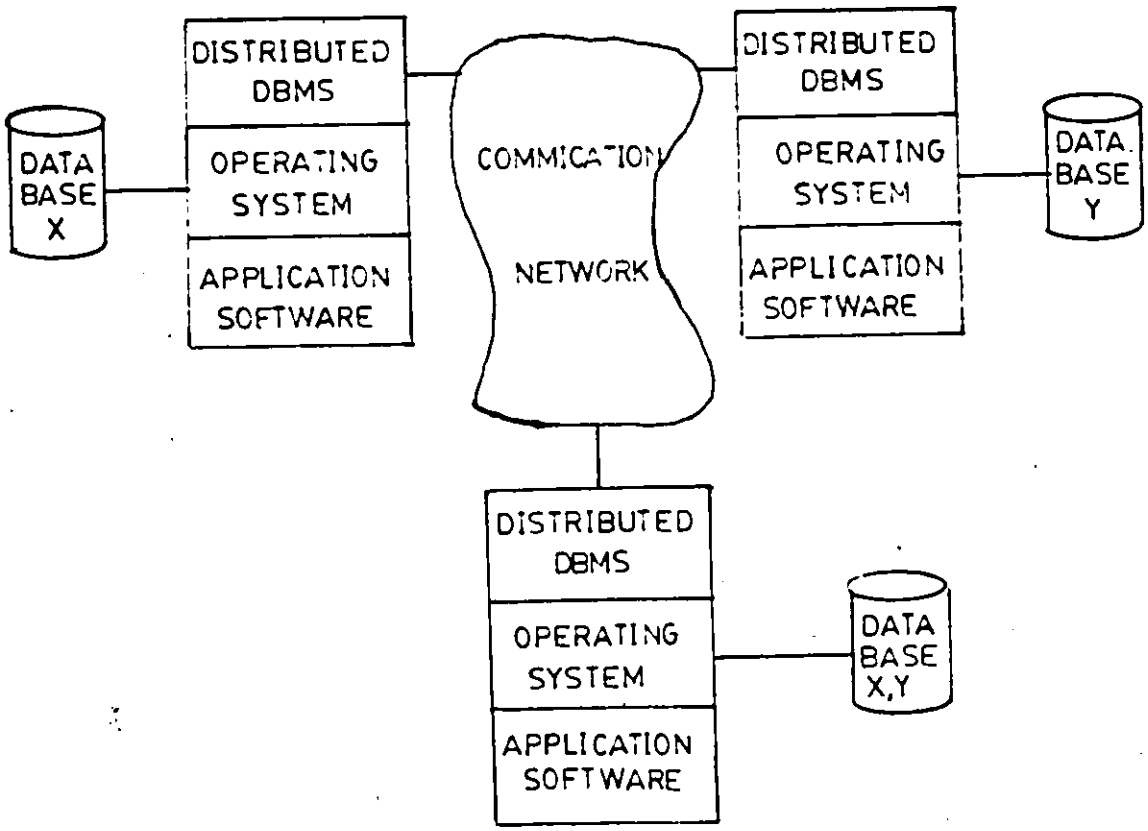

Figure 8: An example of a distributed data base system

2.4.1 Characteristics of distributed data base systems

A distributed data base system can be considered as a collection of cooperative data bases communicating via a general communication medium (low bandwidth). The user's view of the data is an integrated whole, where both data fragmentation and redundancy are invisible; that is, users are not aware of the location of their data.

The above definition of a distributed data base is a highly problematic definition since it is very difficult to give it a precise definition of a distributed data base system. According to Enslow [ENSL 78], there are three dimensions that characterize the degree of decentralization of a system: control, storage and processing hardware, and each of these dimensions has a spectrum of values. This is shown in more detail in figure 9. Thus, in many instances, it becomes a matter of interpretation whether a system is a distributed data base system or not. For example, how do we classify the following systems: (1) a data base computer combined with one or more general purpose computers, and (2) a data base distributed over a computer network, but where all global processing are controlled by a central computer. For a system to be identified as a distributed data base system, Enslow requires that the three dimmensions listed above be highly decentralized, thus it does not consider the above two systems as distributed data base systems. Other

authors [DEEN 82, MARY 78, CHAM 77] advocate a nominal degree of decentralization for each dimension (less restrictive criteria).

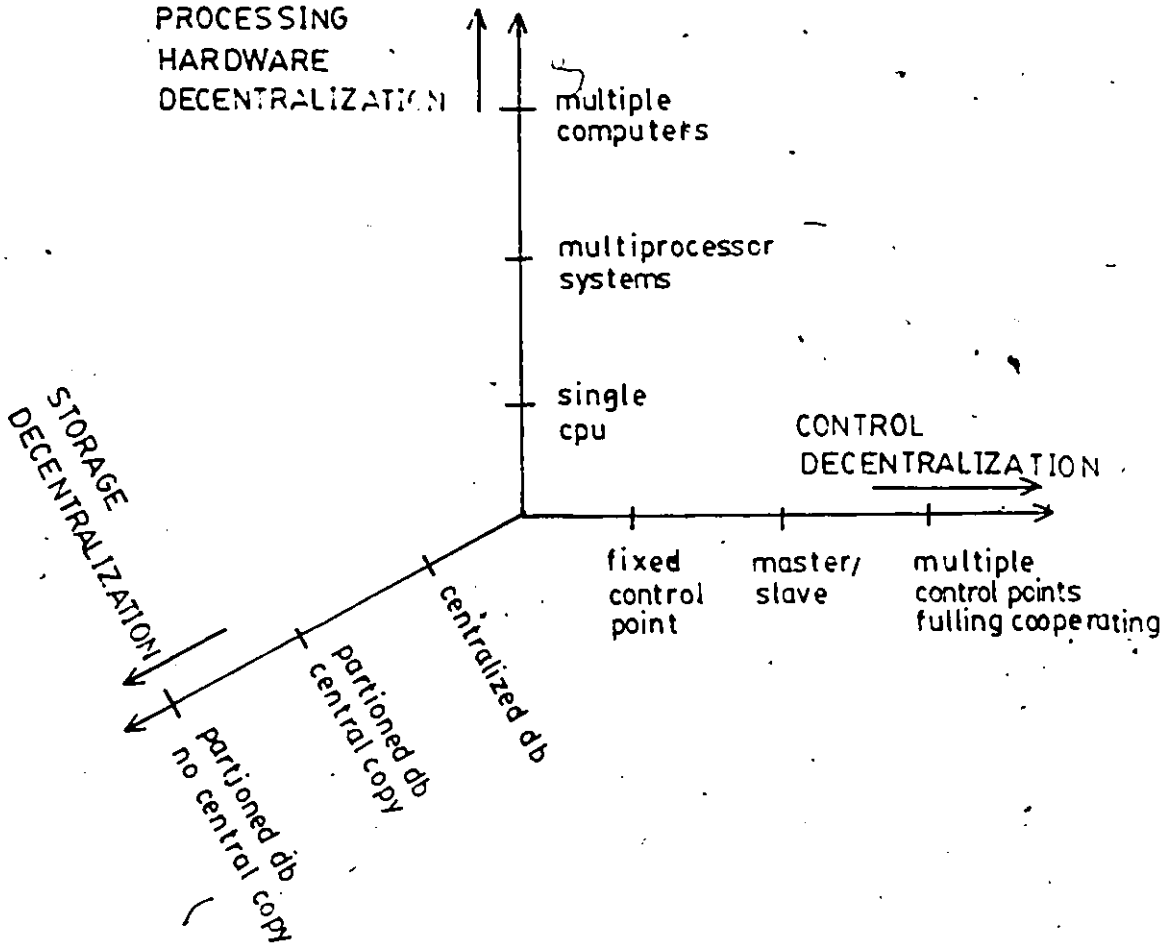

Figure 9: Dimensions that characterize the degree of decentralization

The major advantages of a distributed data base system as opposed to a centralized data base system are [ENSL 78, PEEB 78, CHAM 79, QUIN 81] :

High Availability - The user can access data from different locations in the network.

High System Performance - In general, the data is stored at locations where it is most frequently used. This provides faster access, higher throughput (parallel processing) and lower communication costs if the system specification requires a geographically dispersed system.

Reliability - The probability of complete total system failure is reduced.

Flexibility - It is easier to expend in function and in capacity; i.e. it is easier to upgrade the various components .

Autonomy - The subsystems of each functional area can be more specifically developed to meet their needs and independently of the others.

Hovewer a centralized data base system as the main advantage that it is simpler to implement an integrated data base on a general-purpose computer as opposed to a distributed processon architecture.

The main element of a distributed data base system is the Distributed Data Base Management system (DDBMS). A DDBMS is a generalized software system which in addition to providing all functions of a DBMS, allows users to access data stored on the network in the same way as if it was stored locally. DDBMS can be implemented as special-purpose and general-purpose DDBMS (YAO 78):

Special-purpose DDBMS are designed to perform only a specific task. For example, an airline reservation system could perform only one task : maintaining the inventory of the available seats. The PROMIS system (i.e a hospital information system) which will be described in chapter three is also another example of a special-purpose system.

Federated systems , a type of general-purpose DDBMs, are systems which integrate already existing data base systems to produce an unified data base system. In general, the existing DBMS are different in terms of data representation and data manipulation. These systems are the hardest to implement on account of the incompatibility of the computers, DBMS, ect. One requires a lot of software code in order to integrate the various DBMS. An example of federated system is SERIUS DELTA [LITW 82] which is the current project phase of the French National Institute for Research on Information and Automation (INRIA). SERIUS-DELTA is a general purpose DDBMS which interfaces with existing and largely unmodified DBMS.

Integrated systems are another type of general-purpose DDBMS which consists of a single software system running either on identical or nonidentical computers, with all nodes of the network supporting the same data model. This approach is possible only if the entire system is built from scratch. Example of such a system is POREL [NEUH 82]. This is a pro-

totype DDBMS which supports a relational system which can be implemented on different types and sizes of computers.

2.4.2 Fundamental problems with DDBMS

The fundamental problems associated with the design of a distributed data base system are:

Where to store the data - Data may be distributed in two basic ways: partitioned, where only one copy of each record exists, and replicated, where two or more copies of the same record exist. Replication provides locality of reference potential for parallel processing and better reliability. However, these benefits are realized at the cost of keeping the data consistent during updates and the extra memory space needed [BERN 81]. Determining the optimal data allocation scheme is a complex issue; many parameters are involved: access paterns, amount of data, frequency of update, communication cost and speed, ect. Many methods have been proposed to determine the optimal file allocation, but more research is needed before these models can become practical [BERN 81].

How to locate the data - In general, a directory which indicates the location of every data elements in the network is used. Two opposite extreme approaches for implementing a directory are described:

(1) A full directory may be kept at every node. This scheme results in better query response and reliablity but requires more storage space and multiple directory updates.

(2) On the other hand, the directory may be centralized. Updating is relatively easier but there are many disadvantages associated with this approach such as poor reliability, high communication costs , slow response, etc.

Between these two poles, a full range of possible schemes exist. In general, the manner in which the directory is distributed depends on the same considerations that governed the way the data is distributed (e.g. frequency of updates, delays for response).

Control of concurrent updates - As stated before, when using redundant data, the major problem is to ensure data consistency between the various copies. This problem is complicated because of asynchronous accesses and concurrent updates. Note that there is no such thing as absolute consistency, since there is always a transient state due to the physical delays in the system. Many update algorithms have been proposed to maintain data consistency [CHAM 79, PEEB 78, BERN 81]. These methods involve trade-offs: for instance, one cannot minimize both the communication costs and the delay in completing updates.

Deadlock - If a task must establish temporary exclusivity control over a portion of the data base, then a deadlock may occur. For instance, when a record is updated, no one can be allowed access to the record during the update. In a distributed data base as opposed to a centralized data base, the deadlock problem is amplified on the account that the control is distributed accross the network and the communication delays are long. Whether the data base system is centralized or distributed, there are three basic approaches to resolve the problem of deadlock [SREE 80, MARY 78, CHAM 79]: prevention, avoidance and detection. Many algorithms using these approaches have been suggested, but it is very difficult to determine analytically the operational efficiency of these methods [MARY 78].

Fault tolerance and recovery - The major problem involved here is maintaining data consistency in the presence of failures (e.g. network failures or sites crashes) during update transactions. Numerous papers have been written on how to update replicated data when not all copies are available for update [PARK 83]. Also methods have been suggested to provide proper recovery of failed site [SKEE 83].

2.5 DATA BASE MACHINE

In this last section, we will describe new types of hardware which are designed for the purposes of managing data. There are a number of factors affecting the performance of a DBMS running on a general-purpose computer. The major factors are [MCKE 82]:

(1) A general-purpose computer can not process data while it is in secondary memory. Thus, a large amount of data must be moved between the secondary memory and main memory. The numerous methods for limiting this traffic (e.g. indexing, hashing) require large amount of processing time to execute the insertion and deletion operations, and occupy a large amount of storage space .

(2) In general, a general-purpose operating system supports a virtual memory system which utilizes dynamically the memory in order to support efficiently multiple users. This works against DBMS efficiency where a DBMS tries to keep critical pages (indices) in the main memory.

Data Base Machines were developed to improve the performance and to reduce software complexity of centralized data base systems. These are specially designed systems (both in hardware and in software) that implement the traditional DBMS functions.

Data base machines which implement most of the processes of a DBMS are referred to as Data Base Computers [YAO 78]. An example of such a system is Infoplex which is implemented on a complex of microprocessors [MARY 78]. In this system high-level data base requests are decomposed into segments which are executed in parallel by the microprocessors. Other examples of Data Base Computers are [BERR 82]: the Relational Associative Computer System (RELACS) and the DIRECT system.

There are also other types of data base machines which perform only a subset of the general DBMS functions. These machines can be categorized as: special-purpose processors and intelligent memories.

The basic philosophy behind special-purpose processors is to transfer to hardware a defined DBMS function which was originally implemented in software. Examples of such system are [BERR 82]: search processors, directory processors, inverted file processors.

Intelligent memories are devices which provide quick real time searching by including some processing in the memory. Intelligent memories have two basic capabilities : data can be referenced by content rather than by hardware storage address, and searching can be done in parallel. These systems fall into two major groups:

Associative Memory - An associative memory is usually referred to as solid-state device which includes some logic that allows simple operations to be performed in parallel. Due to their high costs and limited capacity, recent research has focused on various ways to utilize these devices as staging devices for large data base instead of a complete memory system.

Cellular-Logic Devices - The main attribute of these devices is that more processing is moved closer to the secondary memory. The architecture of such devices is shown in figure 10. Examples of Data Base Machines based on this concept are [SU 79]: the Context Addressed Segment Sequential Memory (CASSM) and The Relational Associative Processor (RAP). CASSM consists of an array of identical cell controlled by a central controller. Each cell consists of a circular memory element (disk track of a fixed head disk) and a processing element. The search is direct , i.e. data is processed on-the-fly. The disadvantages associated with these devices are: (1) Fixed-head disks are an order of magnitude higher in cost per bit stored than moving-head disks. (2) The amount of processing that can be performed on a single revolution is limited since the flow of data is continuous. RAP is similar to CASSM except that each processing element has a track memory . Thus, data must be moved into an intermediate storage (track memory) before processing can be initiated.

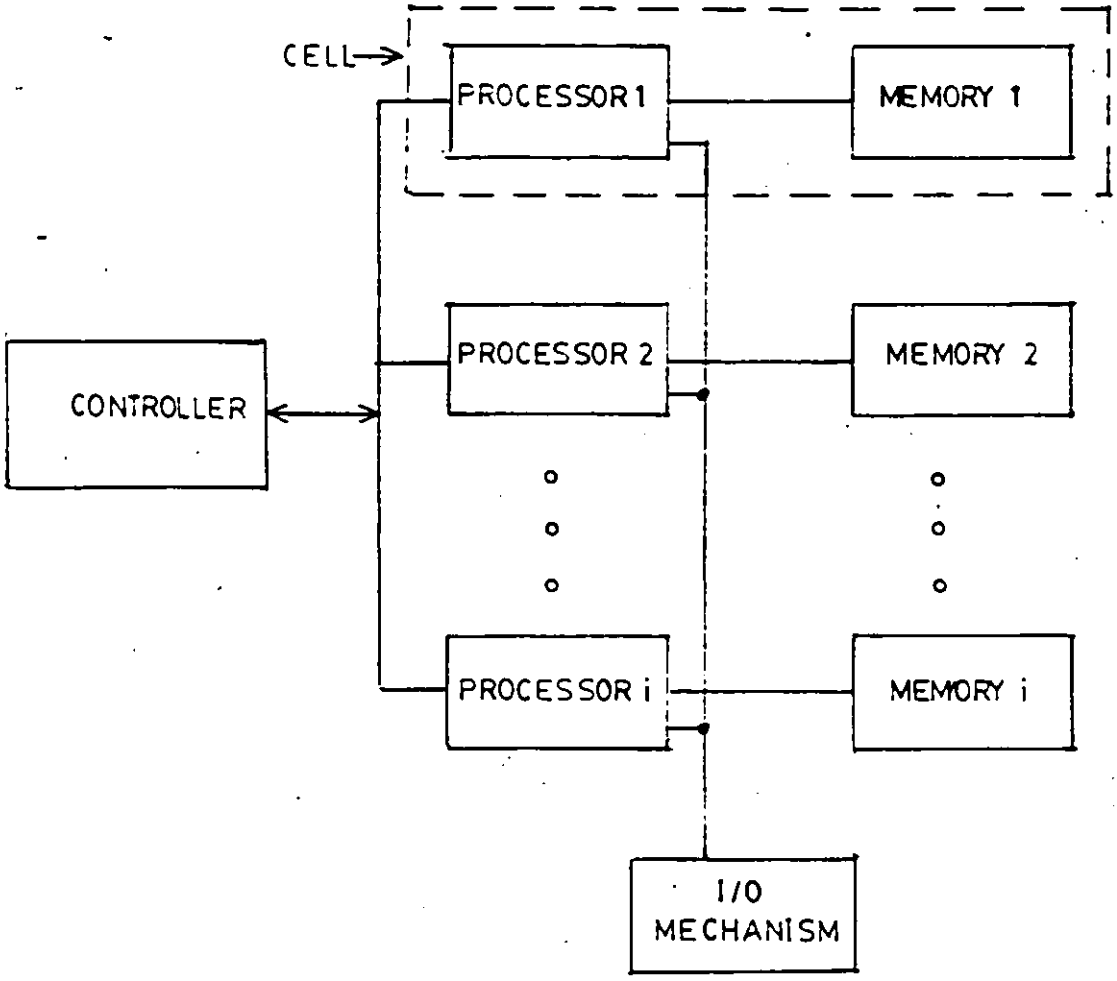

Figure 10: Organization of a cellular device

2.6 CONCLUSION

In this chapter, the basic concepts of data base systems were introduced. The current research activities and technology trends of data base systems were presented next in terms of distributed data base systems and data base machines. This data base overview was done to highlight the major problems associated with current DBMS and to provide the proper background for developing the proposed data base system. More research is needed to develop viable general-purpose distributed data base management system. There are a number of data base machines in operation today. These systems are cost effective in applications which require processing of large quantities of data at high speeds.

One aspect related to data base system not discussed in detail in this chapter is the advances in mass storage technology. With the continuing advances in magnetic recording, magnetic disks will continue to be the dominant technology for a few years [CHI 82]. Other mass storage technologies such as bubble and charge-couple devices offer some advantages over magnetic disks but they are costly to be used for as storage devices for large data bases [TERR 79]. Optical disk technology is emerging, but at present its use is restricted to write-once and read-only applications [CHI 82].

Chapter III

HOSPITAL INFORMATION SYSTEM OVERVIEW

The purpose of this chapter is to describe the characteristics of a Hospital Information system. In order to give an insight of the requirements of such a system, a description of two highly regarded systems is given. These systems are COSTAR and PROMIS.

3.1 CHARACTERISTICS OF HOSPITAL INFORMATION SYSTEM

A hospital information system (HIS), in addition to the data processing requirements of any large institutions (e.g. payroll, inventory control, planning), must satisfy all information activities related to patient care [BALL 75]. More specifically the clinical aspect of a HIS consists of providing a computerized medical record and of supporting the clinical data processing functions of the various ancillary services (e.g. pharmacy, clinical laboratory, radiology) [WHIT 81]. Also included here is patient billing. Although patient billing is a financial activity, it is directly related to patient care; an HIS must capture auto-

matically charges whenever a service is given to a patient [MISH 81].

The computerized medical record system can be considered as the core of a HIS [TOLC 81]. A well designed computerized medical record system must be capable of performing the following functions [BARN 82]:

(1) Allowing immediate access to any specific data in a patient medical record. For instance, when admitting a former patient in regular or emergency situation, the system should immediately provide a pertinent summary of his/her medical history and profile.

(2) Displaying the patient's data in different formats in order to accomodate the needs of the various users.

(3) Including safeguards for detecting errors and preventing unauthorized accesses.

The computerized medical record will be discussed in more detail in the next section.

Each of the ancillary services require various clinical data processing functions. The major functions consist of interfacing various computerized instruments to directly obtain test results, performing quality control tests, commu-

nicating results throughout the hospital and others [GIEB 75, MORT 82]. In general, these activities require access to two distinct data bases [MORT 82]: the medical record data base and special-purpose medical data bases. A medication data base is special-purpose medical data base which contains drug information such as drug-drug interactions, drug side-effects, ect. Another special-purpose medical data base is a clinical laboratory test data base which contains for each test the normal range of results, the amount and type of specimen required, test fees and others.

It can be seen that a HIS requires the integration of a large number of quite different data bases and processing activities [WHIT 81]. This integration allows for sharing of data (e.g. medical record) between the various departments. This integration also provides for controlled communication between the various departments (i.e. it automates the information flow of the hospital).

To give an insight to the daily functions of a HIS, consider the example of prescribing a medication for a patient. First the doctor in the ward may need to consult the patient's medical record to prescribe a medication. Following this prescription must be transmitted automatically as a request to the pharmacy department. Before submitting the medication, the pharmacy, in processing the prescription, must perform numerous verifications which require access to

the patient's medical record and its own special information system (medication data base). Examples of such verifications are : checking for drug allergies and drug-drug interactions, making certain that the prescription is related to one of the active problems. In addition when the medication is delivered, the system must update both the pharmacy inventory and the patient's medical record and provide the proper charges.

3.2 COMPUTERIZED MEDICAL RECORD SYSTEM

A properly defined and updated medical record is an invaluable tool for the clinician in practicing his profession. It is with the help of the information stored in the patient's medical record that the clinician establishes his diagnosis, monitors or guides the patient during therapy, decides on the tests to be performed and on the required surgical intervention. Furthermore, the patient record will serve in studying disease or treatment modality. The majority of published clinical studies are tabulated from such data.

To be effective it is important that the computerized medical record system provides a well organized record and presents it in a format so that the clinician is capable of screening the information relevant to the problems being

treated. There are many ways to organize and present the information stored in a patient medical record. At present, there are two basic record organizations which are widely used [BENJ 80]. The problem-oriented medical record is a format where data gathered from different sources (e.g. nurses, doctors, laboratories) at different times is recorded according to a series of problems [WEED 71]. This format is now taught in most of the medical schools [ESTR 79]. The alternative is the time-oriented medical record where data is recorded chronologically according to their sources (e.g. radiology, clinical laboratory) [FRIE 72]. The major difference between both organizations, is that the time-oriented medical record is source structured and chronologically sequenced as opposed to the problem-oriented medical record which is problem structured and chronologically sequenced. In the following, the information required in a patient medical record is defined without specifying any particular record organizations.

The "patient medical record", when a person seeks medical care, consists of two parts. The first part is the permanent medical record (PMR) and outlines his/her profile and history. The second part is the transient medical record (TMR) and contains all information gathered during the patient's hospital stay. The TMR is usually quite excessive and may contain irrelevant information after the patient has been released. Therefore, it is only kept during the

patient's hospital stay and is used to update the history section of the patient's PMR after the patient has been released.

The PMR of a patient contains the following information [WEED 71, ZIMM 78, BENJ 80]:

- Patient Profile : identification data, hereditary and chronic diseases, drug allergies, etc.
- History : all relevant past information gathered in previous encounters. This information can be organized in either time-oriented or problem-oriented format.

The TMR can also be organized in the same fashion as the history section of the PMR and roughly contains the following information [WEED 71, ZIMM 78, BENJ 80]:

- Admitting information : present patient problems, financial arrangement , attending physicians, etc.
- Physician notes: examination record, possible diagnosis, treatment and therapy, etc.
- Service reports: clinical laboratory , radiology, physical examination, etc.
- Progress notes : observations of the medical staff who are in direct contact and responsible for the care of the patient.

3.3 EXAMPLES OF EXISTING HIS

In this section, we will describe two hospital information systems which are considered as exemplary by professionals knowledgeable in the computer and medical fields [ESTR 79].

3.3.1 COSTAR

COSTAR (Computer-Stored Ambulatory Record) is a centralized medical information system for ambulatory care. Besides providing a computerized patient record for the outpatients, it supports administrative/financial functions such as billing and scheduling. A modular design approach was used for implementing COSTAR. This allows functions to be added or deleted after the system is in operation. The core of the COSTAR system includes the following basic modules: security, registration and medical record system. Other additional modules such as scheduling, accounts receivable module, etc. are available. However, COSTAR does not satisfy all the information activities of the ambulatory group practices. For example no modules are available for the clinical laboratory and the pharmacy.

This system is written in MUMPS (Massachusetts General Hospital Utility Multi-Programming System) language and can be implemented on different hardware configurations which

support this language. MUMPS is a high-level interpretive file management system oriented to medical applications. Because of the storage and processing requirements of COSTAR, the minimal computer configuration is in general a minicomputer system. Note that COSTAR user interface is via non intelligent terminals.

3.3.1.1 Medical record organization

To improve the efficiency of the computer for retrieving and organizing the medical information in a patient medical record, some standardization of the vocabulary used in the medical record is needed. In COSTAR, this standardization is provided through the "Directory": a dictionary containing all medical terms allowed in the medical record. Every entry of the COSTAR directory corresponds to a code standing for a descriptive term such as name of disease or a problem, laboratory test, procedure, medication, etc. Associated with each directory code entry is the actual name and type of the code. For instance, the code 'MHAB1' corresponds to the term hypertension and consists of a problem. This is the minimal information that is required for a directory entry . Additional parameters may be defined for each code such as fees for a procedure or a test, clinical status (e.g abnormal and normal test results) and parameters specifying the structure of the stored data. A basic set of terms

comes with the package. Practices can add or delete terms to suit their special needs.

The COSTAR medical record is divided into two sections: demographic data and medical information. The medical information is stored in a time-oriented format, where time corresponds to the different visit encounters. This information is made up of a number of entities called events. An event consists of a single code recorded at a single encounter. More specifically, the event information consists of the code name with the information related to the code and the encounter information as shown in the status report of figure 12. For example, the information associated to a medication code would consist of a short textual directions, the number of refills, the dose, etc. The encounter information usually consists of the date of the encounter and the name of the provider. Therefore, the medical information can be viewed as a collection of events that link codes to encounters. In its simple format, a patient medical record is formatted as shown in figure 11.

3.3.1.2 User interface

COSTAR provides two types of user interfaces. First, a high level query language is provided for research and statistical purposes. Secondly, for the clinicians access to

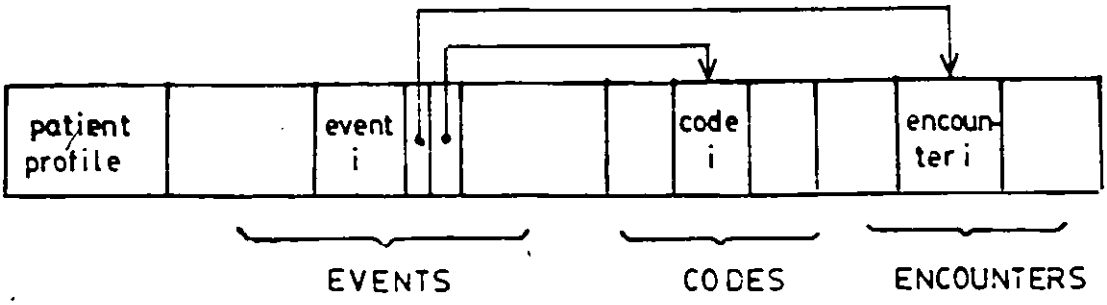

Figure 11: COSTAR record format

the medical records, three basic formats for displaying the patient's medical data are provided:

(1) Encounter report: it displays all information collected during a patient encounter.

(2) Status report: this is a current summary of a patient record. It corresponds to the most current information of a patient's diagnosis, physical findings, medications, laboratory tests, ect. The organization of this report is shown in figure 12.

(3) Flowchart: this display consists of a chronological listing of all occurrences of a particular coded item together with the associated code information.

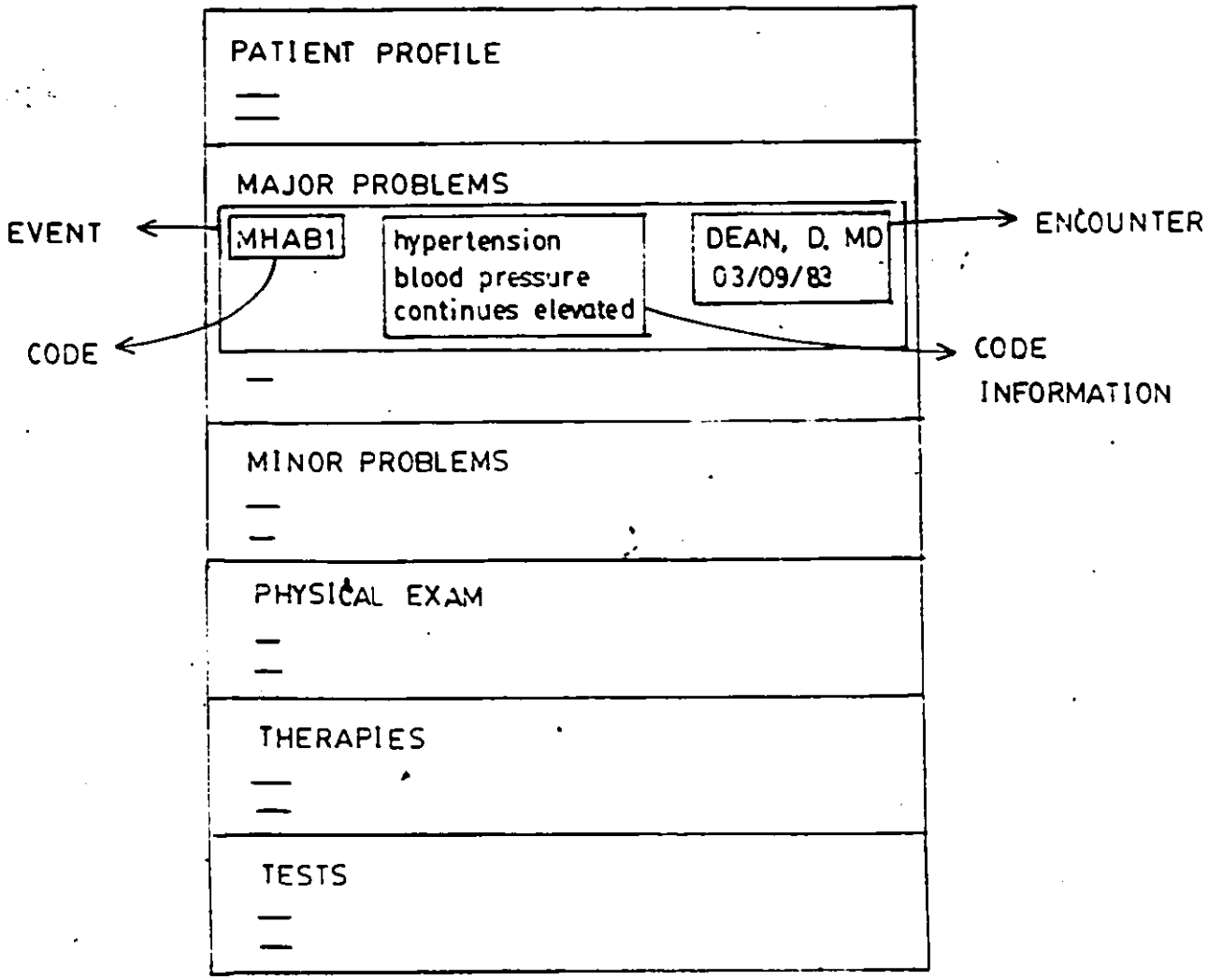

Figure 12: COSTAR status report

As seen above, all user accesses are very rigidly controlled, with the user being able to select only from a very limited set of display modes. In addition, because of the highly encoded format of the information, all input data to the system must be recorded on special encounter forms which are entered into the system via clerical personnel.

Coding of information is advantageous, as it saves space and reduces processing, but it makes the system less flexible; information must be entered into the system in a rigid format. Thus, coding is appropriate for the storage of long term records but not for the storage of short term records.

3.3.2 PROMIS

The major objectives of PROMIS (Problem-Oriented Medical System) is to provide a computerized medical record system and a guidance system for patient care. It provides little support for other applications. As its name implies, PROMIS automates the problem-oriented medical record format.

3.3.2.1 Medical record organization

The main format for displaying a patient medical record is the problem-oriented medical record format. This format consists of (1) a problem list which includes all past and present problems, (2) a data base (e.g. medical history, profile), and written out separately for each problem, (3) a plan (e.g. diagnostic, plan for collecting additional information, ordering treatment), and (4) progress notes which include all data collected after the initial plan has been established [WEED 71, BENJ 80]. Other formats are also sup-

ported by the PROMIS system. For example, it is possible to retrieve directly all laboratory tests for a given patient.

The physical structure of the data base which contains all the patient medical records is divided into three sections: a data section, a problem section and an index section. All these sections are stored in different files. The data section contains, for each patient in a chronological order, all information included in his/her medical record. The records in the data section are divided into simple entries (variable-length logical units), which are usually small and can be addressed individually. The problem section contains, for each patient all his/her problems. Associated with each problem is a list of ordered procedures and the reasons for issuing them. The problem section is stored in a separate file in order to optimize retrieval efficiency because this patient data is frequently accessed. The index section includes a number of index tables pointing to the data section in order to allow the various accessing methods described above.

3.3.2.2 Guidance system

Guidance refers to the giving of medical advice during the inquiry process. The guidance system of PROMIS is implemented by confronting the user with a set of structured

displays. A display is made up of selectable and/or non-selectable text. Choosing selectives choices causes branching to other displays, or actions such as adding data into the patient record, enabling procedure ordering. Thus, by making the selection from the options presented on the display the user progresses through the network of displays, where each display becomes more and more specific. The use of this branching network integrates the patient data and medical knowledge data to arrive at a decision or carrying out an action. In many situations, choosing corresponds to directly writting into patient record. The most general types of displays are: Data base, problem list, initial plan and progress notes. These displays serve as high-level indices to the other displays. According to Weed [Weed 71], they represent the four phases of medical actions. More specifically, the first action in patient care is the collection of the data base (history, profile). From this , the problem list is formulated and for each active problem, initial plans are written. Then each problem is followed by progress notes. Figure 13 illustrates a data base display.

In order to provide such a guidance system, PROMIS contains a special data base that includes knowledge (E.G. defined problems) that has traditionnaly resided in medical libraries, journals and textbooks. Example of such information are: medically defined problems, drugs, radiology procedures, physical therapies, patient history questionnaire,

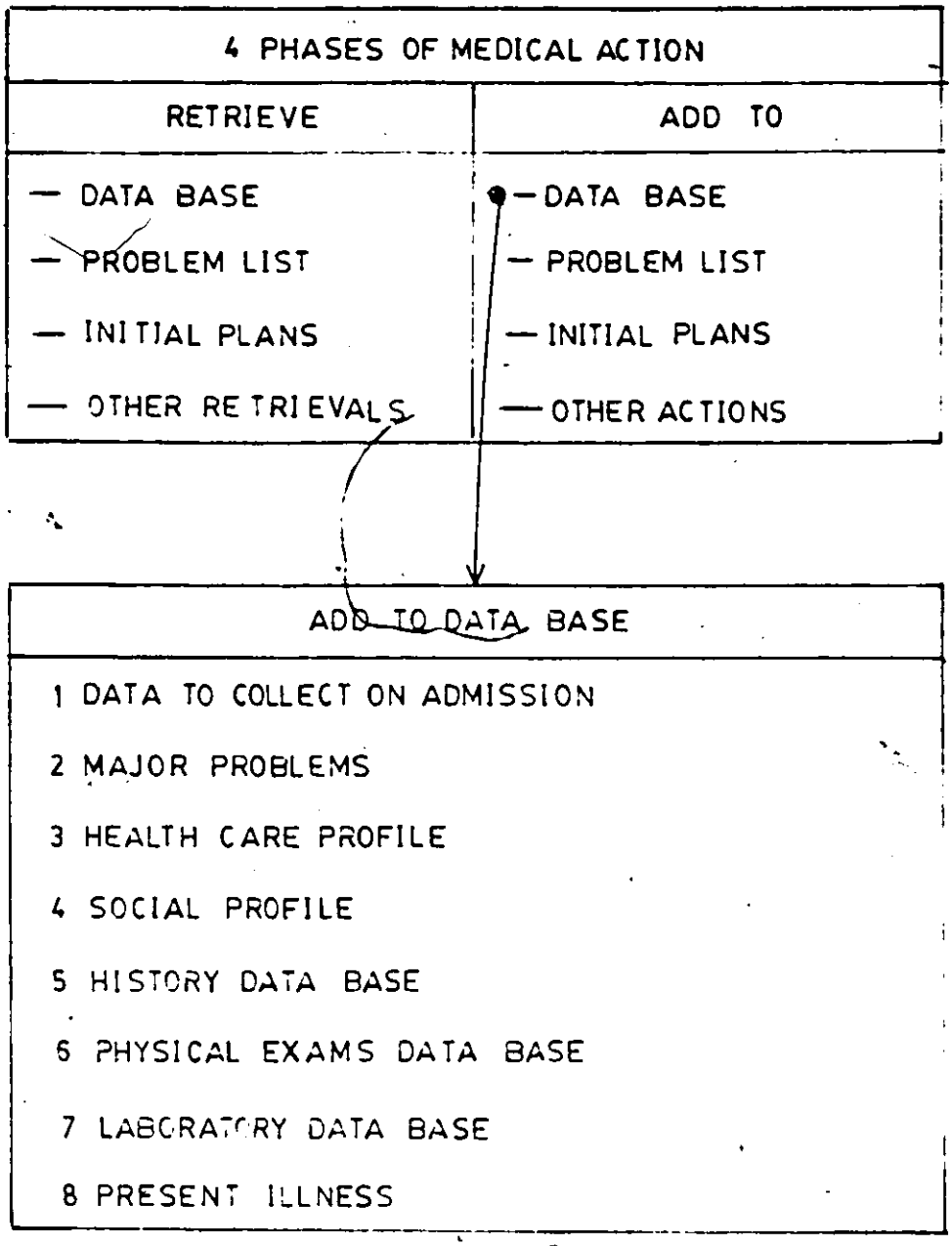

Figure 13: Data base display

ect. Furthermore, the above information must be integrated with the patient medical record.

3.3.2.3 System implementation

PROMIS is implemented on a network of minicomputer nodes connected by a high speed CATV bus as shown in figure 14. The network contains a master node called the network node and one or more common nodes. The network node houses super-nodal files such as a master patient location file. Each common node is a general-purpose node dedicated to specific wards and/or services which can serve approximately 20 to 30 terminals. These common nodes contain a copy of the medical guidance system and application software. Each common node also maintains the medical records for its patients. Furthermore, all PROMIS application programs are written in a tailor made programming language PPL (PROMIS Programming Language).

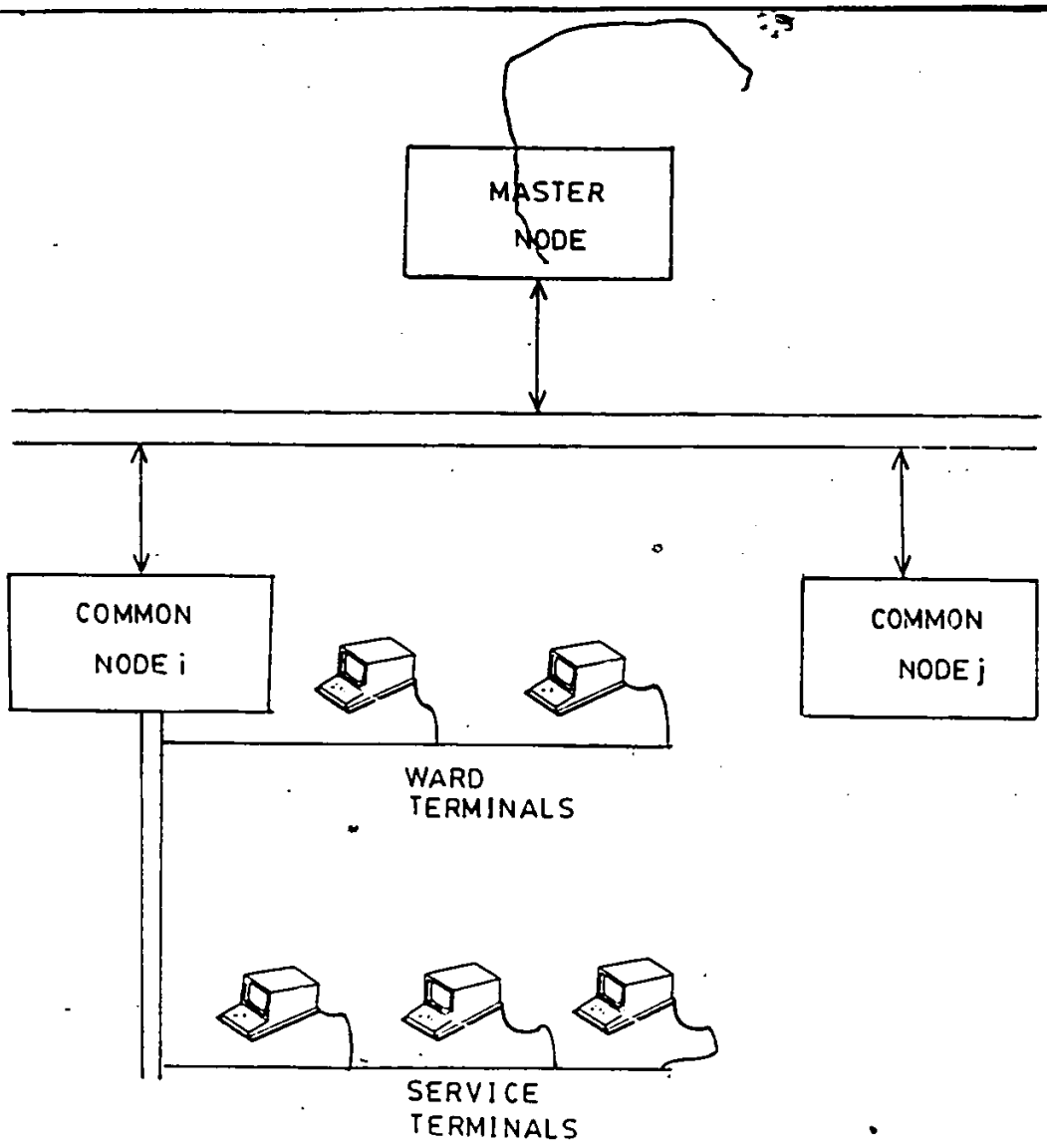

Figure 14: Architecture of PROMIS

3.4 CONCLUSION

Most of the HIS systems are implemented as maxi-computer centralized systems. The major problems associated with these systems are [QUIN 81]:

(1) It shares all problems of a centralized data base system such as cost, reliability, upgradability (high degradation with the number of users increased), etc.

(2) Integration Problems: many hospitals have installed or will installed special-purpose information processing systems such as pharmacy, automated laboratories and others. These stand-alone systems must be integrated into the overall system. Interfacing these modules into a centralized HIS is a difficult and costly task because of the incompatibility of the hardware and software [MISH 81].

(3) Lack of adaptability: if one uses a general-purpose package (e.g. COSTAR), then it is unlikely that a commercial information system can support efficiently all the information functions of all the hospital [GUEM 82]. This problem is due to the fact that the information activities of the various departments varies from one hospital to the other, and that vendor offers only one system for each functional area. For example only one pharmacy system would be supplied.

(4) Lack of flexibility: since most HIS provides limited amount of pre-specified reports, they don't allow different users to have access to the general data according to their specific needs [BLUM 82].

Other examples of available centralized HIS are: The TECHNICON medical information system [ESTR 79] which is implemented on a IBM 370 system or respective compatible systems and IBM's Medical Information System [MISH 81] called "Patient Care System".

There are also a limited number of available distributed HISs such as PROMIS which are implemented on a network of minicomputer . Because of their distributed nature, they benefit from some of the advantages of distributed systems such as fail soft capability, upgradability, etc., but they still carry many of the problems of centralized HIS. For instance, in the case of PROMIS each node has a complete copy of the system which serves a geographically area of the hospital (i.e. a group of wards and services). Thus, whether there is one or more nodes, the facilities provided by the system are always the same. Another example of a distributed system is CHAMPS [CARR 75]. In this system, each minicomputer performs only a specific task such as terminal interfacing, file management. Also, an intelligent local area communication network has been suggested for integrating stand-alone systems [TOLC 81].

In summary, up to the present, most successful medical computer applications consist of executing one specific task (e.g. computed tomographic scanner) [LINC 83]. However, large information systems have proved to be impractical be-

cause they are too difficult, too rigid, or too intensive to be used well [LINC 83].

Chapter IV

A DISTRIBUTED QUERY ARCHITECTURE FOR A HOSPITAL INFOMATION SYSTEM

This chapter presents the design outline of a hospital information system (HIS) using a layered data base organization. Specifically, we shall consider the design of one facet of an HIS, namely an integrated medical record system. Furthermore, we will indicate how it can be integrated with other computerized hospital systems. In the first section, the specification of the scope of application for the system is given. Secondly, the medical record data base organization is specified. Next the system architecture which supports the above data base organization is presented along with its operation. Finally each element of the system is discussed in detail.

4.1 SCOPE OF APPLICATION

In this system, we confine "patient care" to be that which is delivered to inpatients (i.e. patients confined to beds) and to emergency patients. Although clinical outpatient care (.i.e care delivered to a scheduled patient having no bed), is not included, the system could be easily expanded to support it. In the first portion of this section, we will identify the various system users and briefly describe their system access requirements. The remainder of this section will be devoted to a description of the information flow between the system users.

4.1.1 User types

In the system there are four types of users:

Admission - The admission user performs all the actions required for admitting scheduled patients. Furthermore it controls all requests for releasing and transferring inpatients or emergency patients.

Wards - The wards users (e.g., cardiology, maternity, pediatry), require all medical information relating to patients assigned to them.

Services - The hospital services users (e.g., Pharmacology, pathology, clinical laboratory), require specific information on all patients currently in the hospital; that is emergency patients and inpatients.

Emergency - The emergency user requires immediate and priority access to any data in a patient's medical record.

4.1.2 System information flow

The system information flow is described by outlining the user's interactions with the system during a patient cycle. Whenever a scheduled patient is admitted to the hospital, admission must generate the "admitting information record (includes present problems)" and combine this record with the patient profile and relevant medical history to form the 'current medical record' (CMR). If the patient has been previously admitted to the hospital then the patient profile and the relevant medical history can be obtained from the patient 'permanent medical record' (PMR). In the case of a new patient, admission is responsible for generating the patient profile and medical history. To remove any confusion, the CMR consists of the temporary medical record (TMR) with the addition of the patient profile and relevant medical history. Once the CMR is created, it is transferred to the ward to which the patient has been assigned. Subsets of the CMR are passed on to the various services.

The process of admitting an emergency patient is complicated by the nature of an emergency situation; that is, im-

mediate medical attention must be given to an emergency patient. Therefore, when admitting an emergency patient, his/her CMR should be restricted to only what is relevant to the emergency situation. Thus, the CMR is composed of the pertinent admitting information, and of the patient profile and relevant medical history from the PMR. If there is no PMR, as in the case of a new patient, then the emergency must generate a complete enough CMR in order to satisfy the needs of the emergency and the various services. Once the CMR is available, subsets of the CMR is passed on to the various services. After the emergency's critical moment has passed, emergency must obtain the information missing from the CMR (and PMR in the case of a new patient). Subsets of this information may also be required by the various services. Note that once an emergency patient is admitted, emergency interacts with the system in the same fashion as a ward in delivering care to its patients. In the discussion below, the term ward also refers to emergency.

Given that a patient is admitted (from emergency or admission), let us now examine the everyday system operation. Service requests are initiated by a ward where a user (usually a physician), after consulting a patient's CMR, may request a number of actions from the various services. These requests are then transmitted to the appropriate service. If warranted (e.g. as a safeguard), before servicing the request, the service user may request additional information

from the CMR or PMR. Once the ward request has been serviced, the service station transmits the results to the ward that initiated the request. After the results are received, and verified, the ward initiates an update to the patient's CMR.

In this system, all patient transfers (i.e. from a ward or emergency to another ward) are initiated by the patient's present ward. Every patient transfer must be validitated (for safeguard and administrative purposes) by admission before they can proceed. When a transfer has been accepted by admission, the initiating ward must provide an update to the PMR and to the CMR (including the CMR portions of the various services). Furthermore, the targetted ward may request additional history information to be included in the patient's CMR.

When the patient is released, the patient's ward uses the CMR to update the PMR. Afterwards the CMR is deleted.

4.2 DATA BASE ORGANIZATION

In many information systems, users only require access to a small, well defined portion of the data base [QUIN 81]. The philosophy behind the data base organization is driven by this fact. Essentially, each user has a local (private) copy of his portion of the data base. A copy of the total

data base is kept at a central location primarily for downloading to the local data bases and for reliability.

In this application two distinct data bases are identified: (1) the "on-line archival data base" (OADB) containing all PMRs and (2) the "current data base" (CDB) containing all CMRs. The OADB provides the patient profile and the relevant medical history to be downloaded to the CDB. In addition, the OADB provides a data source for research studies. The CDB is accessed by the wards and services since it is their source of working data. As stated in the previous section, user demands on the CDB are small and well defined. For instance, each ward has under its care a small number of patients and each service only requires a limited amount of the patient's CMR. Therefore, the CDB is distributed over the wards and services, with each ward and service having a local data base and specialized tailor made query processing system. By distributing the data in this fashion, the following functions are necessitated:

(1) Cooperative algorithms are required for controlling concurrent updates,

(2) schemes are required for locating data which is not locally available, and,

(3) security mechanisms are required at each node.

Performing these functions at each work station (ward or service), complicates their implementation, and requires considerable communications between the work stations. In an alternate approach, all global transactions (e.g. access for additional information, update) are channelled to a central node where a complete copy of the CDB is kept. The justification for the cost of data duplication is counterbalanced by simplicity of access, the ease of the maintenance of the local data bases and by having a more reliable system.

The data base organization is a layered system with user access being restricted to the outermost layer as shown in figure 15. This layer is subdivided into a number of dedicated user-oriented data bases referred to as local data bases. At the innermost layer is the OADB. In between these two layers is the CDB.

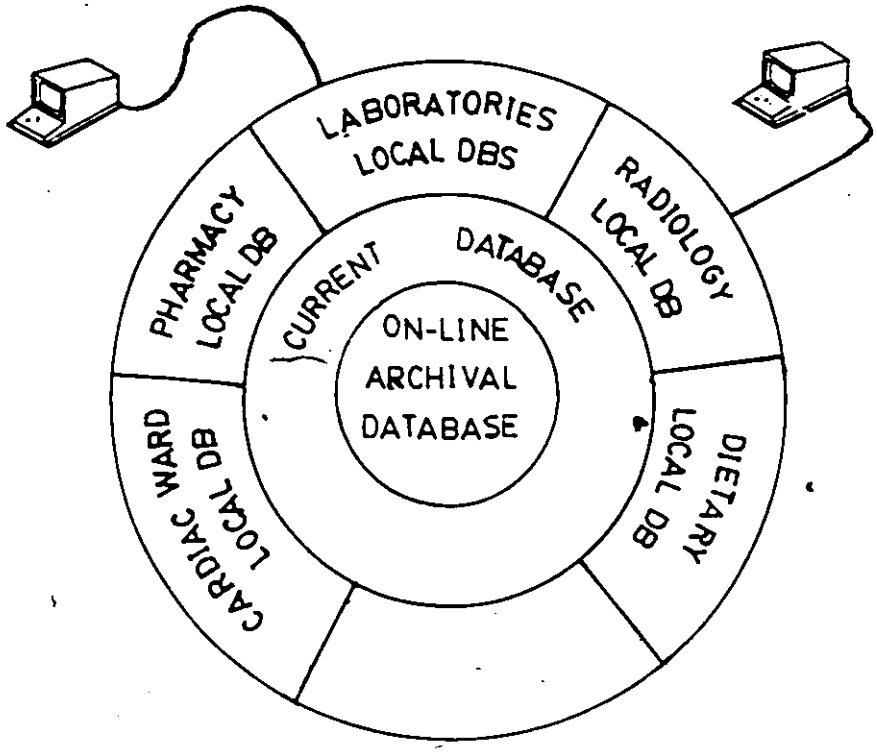

Figure 15: Data base organization

4.3 SYSTEM ARCHITECTURE AND OPERATION

The HIS architecture to support the layered data base organization is illustrated in figure 16. Three basic system elements are identified: (1) data base manager subsystem which controls all accesses to the CDB and to the OADB, and the interactions (local data bases) between the work stations and admissions, (2) the service, ward and emergency work stations (local data bases), and (3) admission. The data base manager subsystem and the work stations are inter-

connected by a common bus referred to as the "central bus". All information on the bus is transmitted in the form of a tagged message. Each transmission is a broadcast from one element to all others. The admission is directly connected to the data base manager subsystem mainly because the actions for admitting, transferring and releasing a patient require in general access to the OADB.

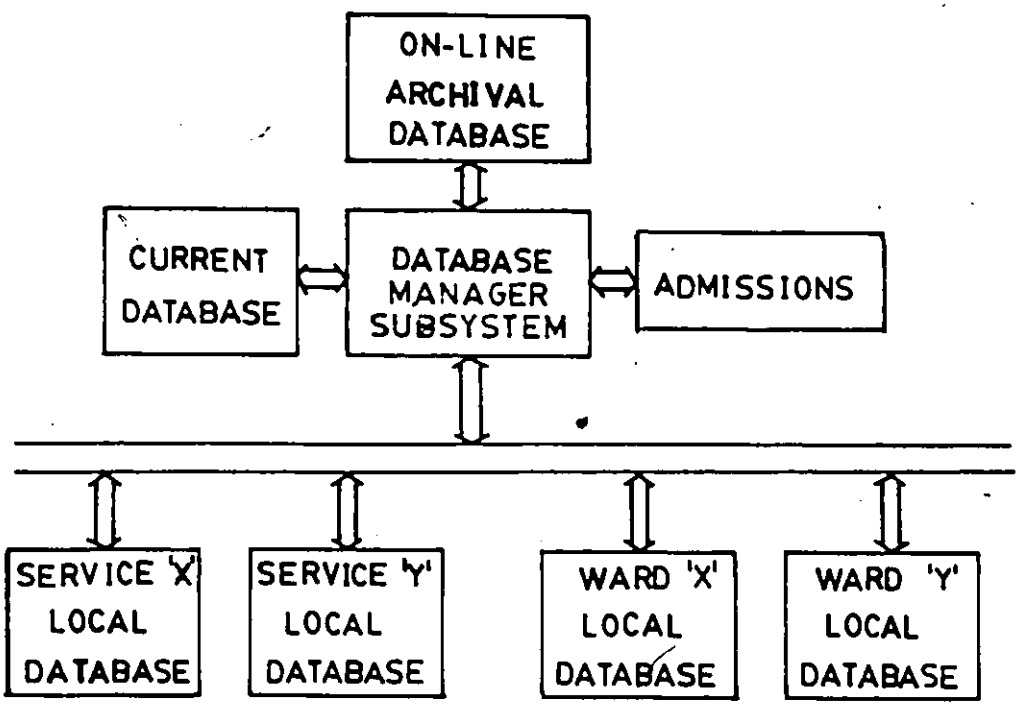

Figure 16: System architecture

The organization of the data base manager subsystem as shown in figure 17, is based on the concept of separating processing and coordination activities [KRIE 83] in order to provide a reliable, efficient and simple to design system. In this system, the coordinator is the data base manager and the processing elements are: CDB controller, the OADB controller, the admission interface and the bus controller.

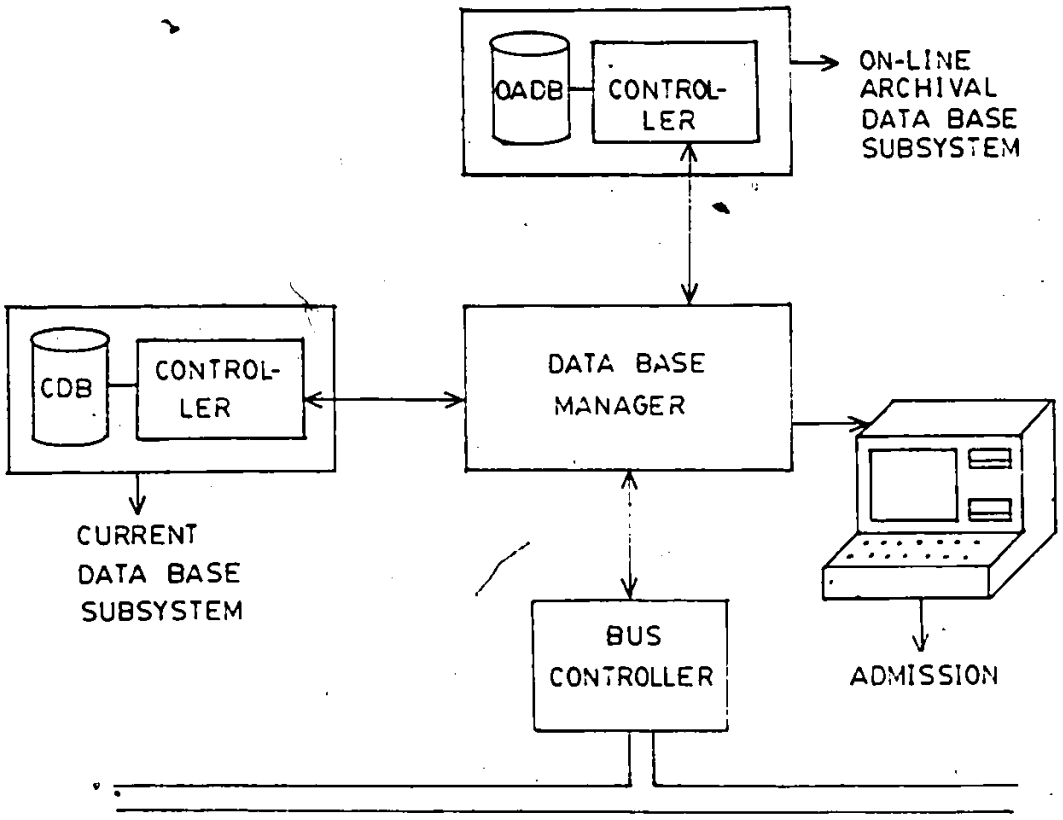

Figure 17: Data base manager subsystem organization

The functions provided by this HIS system can be divided into three classes:

(1) Functions that can be performed locally in the work stations such as requests for retrieving locally available data and producing various terminal displays.

(2) Periodic functions such as transferring from the OADB the less pertinent history information to an off-line memory system and executing garbage collection routines for each data bases (including local data bases).

(3) Functions requiring interaction between the various elements of the system such as data base downloading when a patient is admitted.

These functions classes can be categorized as classes that do require interactions between system elements during execution and those that do not. The first two functions classes fall into the first category; this implies that their implementation is fairly straightforward. Functions of the last class does require interactions; therefore, these types of functions are inherently more difficult to implement. The functions of this category are enumerated and describe as follows:

A) Admitting a patient from admission. This process is initiated by admission and involves the following actions:

(1) In the case of a former patient, the OADB is accessed in order to retrieve the patient profile and the relevant medical history according to the retrieval strategy specified by admission. The retrieval strategies are discussed later in this chapter. The retrieved data is combined with the admitting information to form the CMR. In the case of a new patient, admission must generate the CMR by using a special questionnaire to determine patient profile and medical history.

(2) The CMR is transferred to the CDB. At the same time, this record is broadcasted over the central bus so that each work station can store relevant sections of the CMR in their local data bases.

B) Admitting a patient from emergency. This process involves the following actions:

(1) The OADB is accessed to determine if there is a record for this particular patient.

(2) If there is such a record, then the patient profile and relevant medical history is retrieved and combined with the pertinent admitting information to

form the CMR. In the case of a new patient, emergency must generate the CMR by using a special questionnaire to obtain the pertinent patient profile and medical history.

(3) This initial CMR is transferred to the CDB. At the same time, it is broadcasted over the central bus to be picked up by the appropriate service work stations.

(4) When possible, information left out of the CMR (because of the urgency in admitting the patient), is added. This involves sending an update to the CDB and to the various service work stations.

C) Service request. These requests are initiated by wards and involve the following actions:

(1) All demands for a service request must be validated by the data base manager subsystem.

(2) If valid, then the ward transmits the actual service request on the central bus where it is picked-up by the data base manager subsystem and the appropriate service work station. For proper system operation, each ward maintains a list of all outstanding service requests which is updated whenever the request has been serviced. Furthermore, the data base manager subystem also maintains a list of all outstanding service requests for each patient in the CDB.

D) Update request. These requests are initiated by wards and can be of two types: (1) those resulting from the data produced by a ward (e.g., doctors, nurses), and (2) those resulting from data produced by the completion of a service request. This update process involves the following actions:

(1) All update requests must be validated by the data base manager subsystem before they can be initiated.

(2) If valid, then the ward broadcast the update over the central bus where it is picked up by the data base manager subsystem to update the CDB. Furthermore, if the update resulted from a service request, then the update must also be picked up by the appropriate service work stations to update their local data bases.

(3) IF the update is the result data of a service request, then the list of outstanding service requests in both the ward local data base and CDB must be updated.

E) Additional information request. For each work station, all requests for accessing data which are not available in its local data base are directed to the data base manager subsystem. This involves the following actions:

(1) The data base manager subsystem checks the validity of the request.

(2) If valid, the data base manager subsystem obtains the information from the appropriate data base (i.e. CDB or OADB).

(3) The data base manager subsystem transfers the required information to the work station that requested the information.

F) Transferring a patient. This process is initiated by a ward or emergency by sending a tranfer request to the data base manager subsystem. The following actions take place:

(1) The data base manager subsystem checks for proper authorization, and if valid transfers the request to admission.

(2) When admission accepts the transfer, the data base manager subsystem signals the initiating work station to proceed with the transfer. The initiating ward must generate an update to the CDB and OADB. Simultaneously, if additional history information is required by the targetted ward, a retrieval request to the OADB is issued. We assume that the retrieval request was transmitted (by the initiating ward) along with the transfer request.

(3) When the data base manager subsystem receives the update to the CDB, this update is combined with the ad-

ditional history information to form the new CMR. This CMR is transferred to the CDB. Concurrently, the CMR is broadcasted over the central bus to be picked-up by the targetted ward and the various services.

G) Releasing a patient. This process is initiated by a ward and involves the following actions:

(1) The data base manager subsystem checks the validity of the request. If valid, the request is directed to admission.

(2) When admission accepts the release request, the data base manager subsystem verifies that all outstanding service requests are completed.

(3) Then data base manager subsystem signals the patient's ward to transmit the required data for updating the OADB. Concurrently, it signals all work stations to delete this patient record from their local data bases.

(4) When the OADB is updated, the patient's CMR is deleted from the CDB.

4.4 SYSTEM ELEMENTS

4.4.1 On-line archival data base subsystem

The on-line archival data base subsystem consists primarily of a storage device and an intelligent controller. The controller services all requests to access the OADB. Functions performed by this unit are similar to the functions provided by a file system, but in this case they are implemented on a dedicated unit. The following subsections will be described: (1) the accessing requirements for the OADB, (2) a record organization that permits efficient processing of the accessing operations, and (3) the issues involved in the implementation of the archival data base subsystem are discussed.

4.4.1.1 Accessing requirements

As stated before, the OADB holds the PMR of all former patients. The PMR includes the patient profile and medical history. The patient profile is always retrieved as a whole, thus this part of the record is manipulated as a single entity in the OADB. On the other hand, one must be able to access specific information within the history section.

There are three fundamental strategies for retrieving the medical history information of a patient. The first strategy is retrieving information according to its relevance to the present medical problem. The second strategy is to retrieve the past information according to time. The third strategy is retrieving information according to its level of general importance. Alternatively, a mixture of any two or all three of the above strategies is possible. The format of the record organization should reflect the chosen retrieval strategy in order to make searching more efficient.

4.4.1.2 Record organization

The on-line archival record (PMR) is divided into two parts: patient profile and history. The complete record organization is illustrated in figure 18. The patient profile is generally fixed length while the medical history is variable length. To access specific information within the patient medical history, this subrecord must be well structured into well defined addressable fields. Furthermore, because of the nature of medical information, the fields must also be of variable length. In most HIS, the retrieval of the past history is done in a problem-oriented mode or in a time-oriented mode. In this system, the patient medical history is organized in a problem-oriented format. The medical history can also be retrieved in a time-oriented format, but the operational efficiency would diminish.

The medical history is divided into a number of variable length problem sections. Each problem section consists of all data related to a particular problem. Each problem section is further subdivided into a problem field descriptor and a number of variable length encounter sections. The problem field descriptor includes the name of the problem. The encounter section consists of the data collected during an encounter visit for a particular problem. In turn, each encounter section is subdivided into an encounter field descriptor and a number of data fields. The encounter field descriptor includes the date of the encounter. Each data field is made up of data field descriptor and a variable list of data items representing information such as lab results, a diagnosis, and so on. The data field descriptor includes: tags indicating the importance of the information (i.e status flag), user access security tags and the type of information (e.g. laboratory results, diagnosis). The status field is used for filtering irrelevant data during retrieval. Note that the security tags could be included in the problem field descriptor instead of being included in the field data descriptor.

4.4.1.3 Implementation aspects

An estimation of the size of the OADB will yield some insight on the system requirements. It is very difficult to

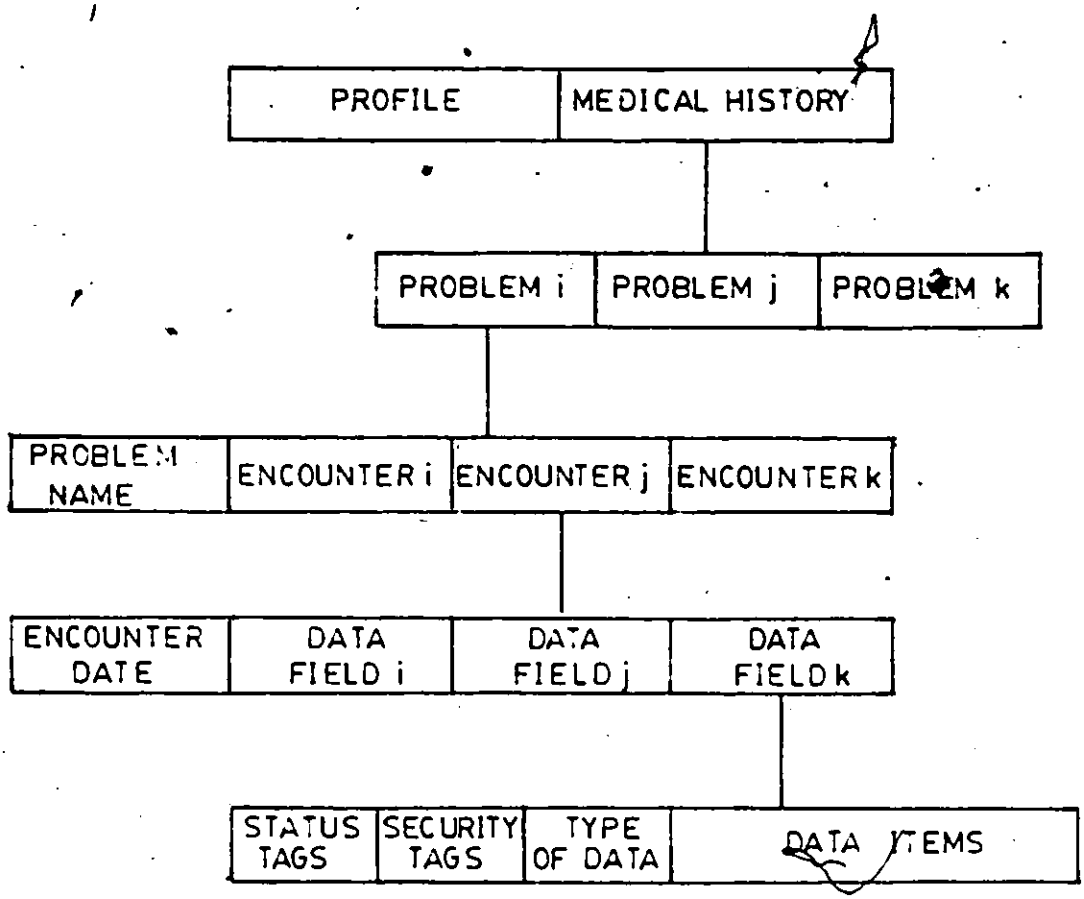

Figure 18: Archival medical record organization

determine the size of a PMR for several reasons: (1) the wide variation in the contents of a PMR, (2) the period of time that the information is kept on-line, and (3) the recording format: encoded or narrative text. From the literature [MCDO 83, SCHU 79, BEAM 79], it was concluded that an average of 5K to 10K bytes for the on-line archival record (PMR) is an appropriate estimate. For example, a medium size hospital with 50000 former patients will require a OADB of approximately 500M bytes.

Based on this assessment, the OADB can be implemented as one of the following memory systems:

- A magnetic tape based system: This has the advantage of storing off-line information at a low cost [CHI 82]. However, because a tape drive is a sequential device, its response time degrades rapidly with the number of accesses.

- A moving-head disk system: the average access time is superior to that of a magnetic tape, system but the cost for maintaining off-line information is much higher. In addition, a more complex backup procedure is required.

If one chooses a moving-head disk memory system, various index tables can be included to reduce retrieval time. If a magnetic tape system is used, all searches must be performed sequentially. Currently, most large data base systems use two separate technologies for on-line and archival storage: magnetic disks and tapes [COPE 82].

In the following, we will briefly outline the implementation of a disk based archival data base. In a disk system, there are two fundamental approaches for information retrieval. The first consists of processing the information as it is being retrieved (i.e. 'on-the-fly'). In the second approach, first the data block is transferred in a buffer

where it is subsequently processed. The first technique requires a faster processor since it must process data without the benefit of buffering. For our system, the second approach is more appropriate because the OADB is accessed infrequently.

Let us now examine the influence of the buffer in the "block-buffered" retrieval technique. The minimal size of this buffer is determine by the file blocksize. The blocksize in turn affects the amount of disk storage needed and the number of input/output operations for record retrieval. System performance can be enhanced through the use of double buffering.

Next, a suggested directory structure for the OADB is as follows. Patient medical records are accessed through a patient directory. The patient directory contains a set of record pointers for each patient. This set of pointers references : (1) the patient profile record, and (2) the problem subrecords of the medical history. Overflow pointers are included to facilitate expansion of the medical history.

An off-line memory system must be integrated to the OADB in order to support back-ups and the archival of irrelevant outdated information. This system is design for patient care, thus to allow the inclusion of additional functions, such as enabling various queries for research purposes, other index tables could be included.

4.4.2 Current data base subsystem

The current data base subsystem consists of a storage device and an intelligent controller. The controller processes all accesses to the CDB. In the first subsection, the accessing requirements for the CDB are described. From this a record organization that permits efficient processing of the accessing operations is presented. Finally, the issues involve in the implementation of the CDB are briefly described.

4.4.2.1 Access requirements

The CDB contains all CMRs. The CMR consists of the admitting data, patient profile and relevant medical history, and information gathered during the patient's hospital stay. Most of accesses to the CDB are to update the CMR. There are also occasional requests from services to access information not locally available.

4.4.2.2 Record organization

The CMR is divided into two parts: (1) the patient profile and relevant medical history, and admitting information, and (2) all data gathered during the patient's hospital stay. The first part is static, seldom accessed, and is

stored primarily for reliability. The second part is dynamic; this information should be structured according to information source (e.g. service or ward) in order to facilitate retrieval and updating. The complete organization of a CMR is shown in figure 19.

The dynamic subrecord is divided into two sections: ward and service. The ward section consists of the information entered by the ward staff (e.g. nurse notes, doctor notes). The service section consists of the results produced by the various services. This information is structured according to the service who entered it. Furthermore, each service subsection is subdivided into a number data fields. Each data field is composed of a data field descriptor indicating the date of the request, a tag indicating if the request has been serviced, and variable lentgh list of data items.

4.4.2.3 Implementation aspects

It is very difficult to obtain an estimate of the size of the CDB. The justification is similar to that presented in section 4.4.1.3 for the OADB. From the literature [MCDO 83, SCHU 79, BEAM 79], it was concluded an average of 10K to 15K bytes for a CMR is an appropriate estimate. For example, a medium size hospital with 1000 current patients will require a CDB of approximately 20M bytes. The current data base is

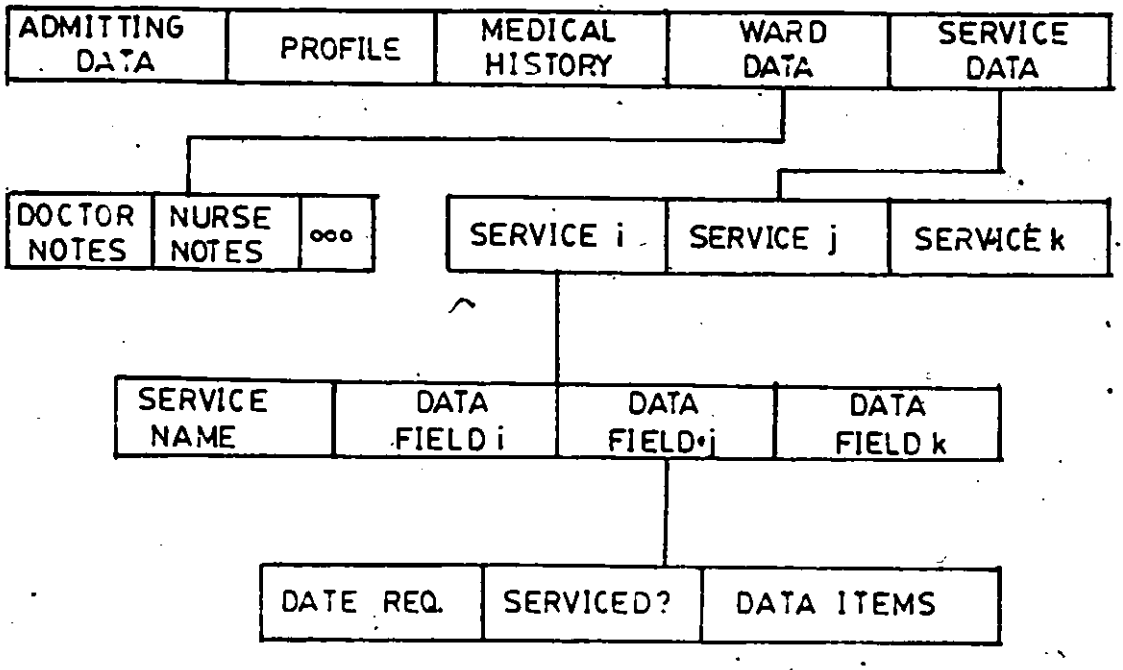

Figure 19: Current medical record organization

accessed frequently therefore a hard disk system could be choosen.

The CDB is mainly accessed to update the CMR, therefore using the "block-buffer" retrieval technique is a reasonnable design choice.

The CDB can be partionned into three different files:

(1) The "static" file includes the admitting data and patient profile. These fixed length records are seldomly accessed. Thus a simple file organization is required for this collection of records.

(2) The "ward" file includes information entered by the ward staff. These variable length records are also accessed seldomly. Again a simple file organization is sufficient.

(3) The "service" file includes information provided by the services. These variable length records are accessed frequently for updates, for storing outstanding service requests and for retrieving additional information. In this case, the desire to maintain an acceptance level of performance justifies the use of a more complex file organization.

4.4.3 Work stations

One of the major difficulties in the design of a multi-user information system is to provide a user-friendly interface tailored to specific user needs. In the suggested system, each of the work stations has its own data base, therefore an information retrieval system tailored to specific user needs can be provided. More specifically, all applications programs (e.g. generating menus and reports), are part of the work station, since they are tasks that operate on local data. programs can be tailored to user needs. In addition, various dedicated functions can be included, for example the automatic scheduling of service operations.

There are three types of work stations: ward, emergency and service work stations. As stated previously, except for the case of admitting a patient, emergency is functionally similar to a ward. Each ward work station contains the complete CMR information of the patients assigned to their ward. Each service work station has a limited amount of information on all patients currently in the hospital. This information consists of (1) a selected section of the patient profile, admitting data and relevant medical history, and (2) the service section containing data produced by various services which is relevant to them. In general, a work station would be implemented on a single microcomputer-based system. Considering present day cost of hard disk system and the advisability of preventing users to handle disks, small non removable hard disk system should be integrated in all work stations.

4.4.4 Data base manager

The data base manager is the control element of the data base manager subsystem. Its main duty consists of coordinating the various processing elements.

From the description of the system functions that are controlled by the data base manager subsystem, it is apparent that the data base manager must be a highly interactive

system. This can lead to severe bottlenecks. The design of the data base manager will be dealt within the next chapter.

4.4.5 Bus controller

In this system, the bus controller is an intelligent communication device. The structure of the bus controller consists of two levels. The inner level provides the proper communication interface with the central bus. The outer layer provides high level functions (tasks) to the data base manager. 'Broadcast a CMR' is an example of such tasks. These tasks are defined in the next chapter. From the system operation requirements, it can be seen that the traffic on the central bus is very low and thus serial data transmission via a pair of twisted wire could be used with a simple protocol to resolve bus contention. The following two bus approaches can be consisdered:

- passive media with handshaking protocol;
- centrally controlled bus where all requests for accessing the bus are serviced by a central controller.

The first strategy requires more processing power at each station while the second would require more lines and may be less reliable. The decision should depend on the physical layout of the hospital and the estimated traffic.

4.5 SYSTEM INTEGRATION

In many large institutions, one of the major problems is to integrate the various special purpose information processing systems such as patient billing, scheduling of services, inventory system, automated laboratories and other specialized information retrieval systems (e.g. diagnostic aids, pharmacology decision aids) [MISH 81].

The distributed patient information system suggested in this chapter can easily be interfaced with most of the above mentionned systems. In the suggested architecture, system integration can be done in the following ways:

- Connecting directly some of the work stations to other information systems or computerized medical instruments.
- Connecting some special-purpose systems as listeners on the bus so they can capture information relevant to them.

4.6 CONCLUSION

A complete distributed system solves the major problems (e.g. adaptability, flexibility, reliability) associated with centralized HIS [QUIN 81]. However, the main problem with this distributed approach is the lack of integration [QUIN 81]. In addition to providing the general advantages of distributed systems, the suggested system overcomes the

integration problem. This was achieved by implementing the nucleus of an HIS (i.e. medical record system) as special-purpose system using the layered data base approach. Yet to be discussed is the cost of such a system; that is the hardware, software and maintenance costs. This is done by considering the cost relative to a general-purpose centralized data base system. We estimate that there are no major differences in the hardware costs. The main difference in the cost of a work station as opposed to an intelligent terminal, is the cost of the storage device. This added cost is counterbalanced by the use of a simple multi-microprocessor based system instead of a minicomputer (e.g. VAX system) for accessing the on-line data base. However, one can tailor special-purpose user-friendly interfaces at much lower cost in the described system. Maintenance costs should also be much lower because of the distributed nature of the system.

Chapter V

IMPLEMENTATION ASPECTS OF THE DATA BASE MANAGER

The primary function of the data base manager is to initiate the controller tasks for executing the system-wide functions. These controller tasks and their sequencing relations are defined in the first part of this chapter. Next an implementation of the data base manager is given.

5.1 DATA BASE MANAGER FUNCTIONS

In this section we will specify the data base manager actions that support the data base manager subsystem functions outlined in the previous chapter. To facilitate the discussion, a special graphical notation is introduced. This graphical notation illustrates the data flow between the various controllers and the data base manager. In addition, it also illustrates the sequencing relation of the controller tasks.

5.1.1 Graphical notation

This notation was inspired in part from the notation suggested by Buhr [BUHR 84], for the represention of the data flow between concurrently executing modules. In our case, the modules correspond to the data base manager and the various controllers. This pictorial notation was also inspired from the data flow [RUMB 85] for representing task sequencing relationships.

An element 'A' requesting an element 'B' to execute task 'K' with parameters 'P' is represented in our notation as shown in figure 20. The response from element 'B' to 'A' can be one of the following three: data, status (yes,no), and/or task completed. Graphically they are denoted as depicted in figure 20 .

The task sequencing relationship is represented by a directed graph as shown in figure 21. The nodes correspond to the execution of a task, the trigger arcs represent the control signals and the arrows emerging from the center of a node illustrate the initiation of a task. A node can initiate a task when all of its input trigger arcs are set. When the initiating task is completed, the corresponding node sets all of the trigger arcs leaving this node.

Conditional control is introduced through two special operators: branch and merge operators. These special opera-

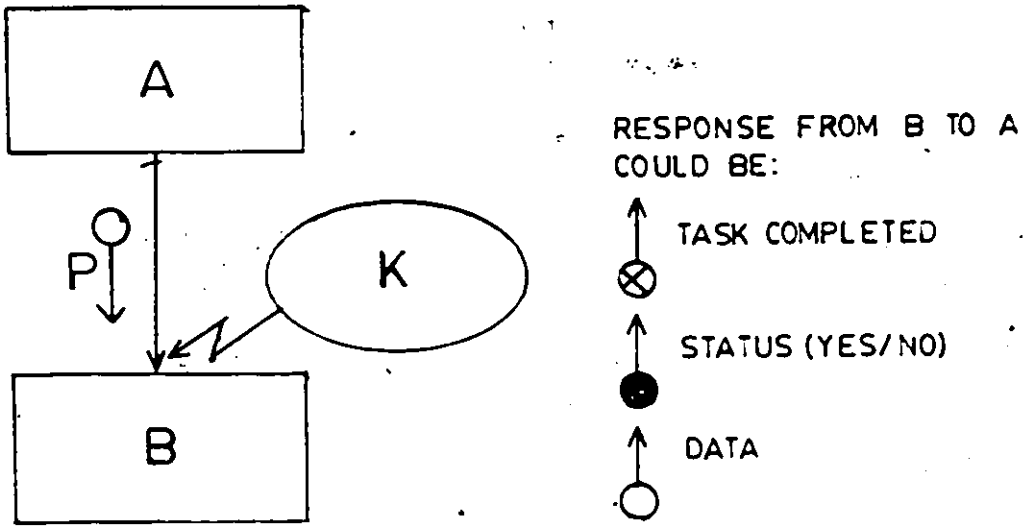

Figure 20: Data flow

tors are illustrated in figure 22. The branch and merge operators can have any arbitrary number of input and output trigger arcs. The branch operator functions as follows: when all of the input control signals are set, then the branch node sets the output trigger arcs that correspond to the evaluated condition. The merge operator operates as follows: whenever an input trigger arc is set, then all trigger arcs leaving this node are set. Figure 23 shows the construction of a conditional control using these operators.

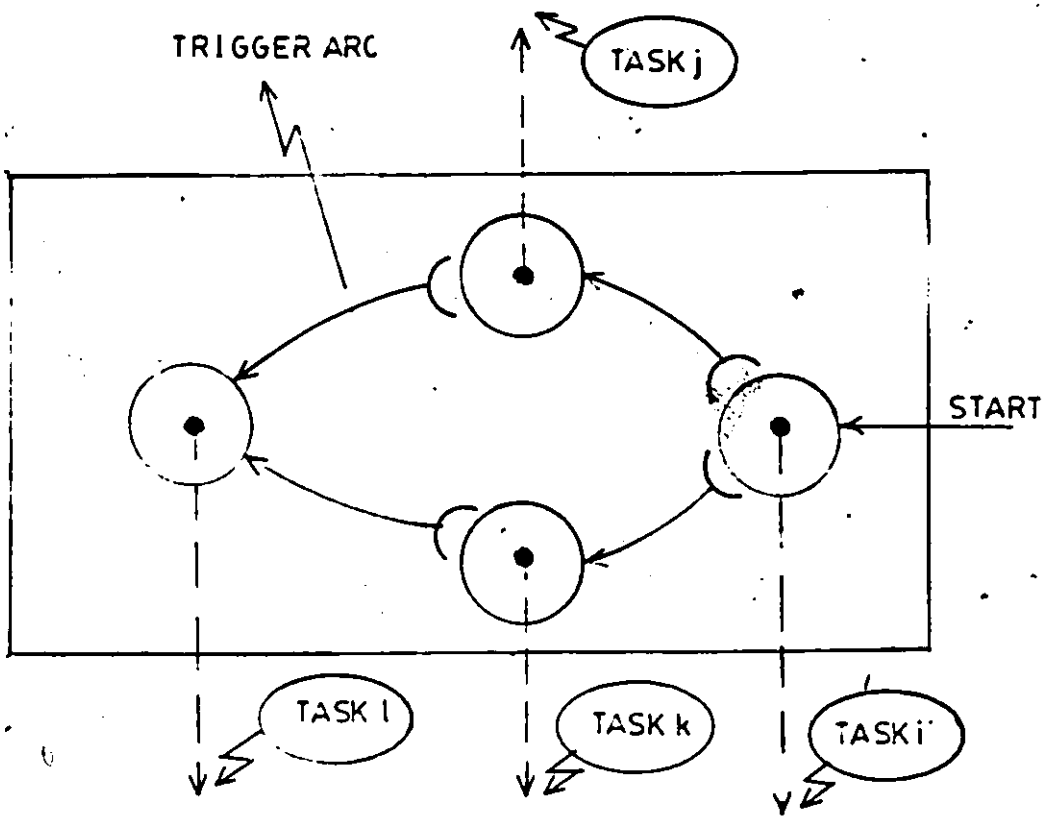

Figure 21: Control directed graph

For the completeness of this graphical notation, three additional operators were introduced: wait, timer/counter and gate operators. These operators were not used in the thesis. The wait operator works as follows: when all input

trigger arcs become active, then all its output trigger arcs become active. This node is defined to simplify drawing. For the timer/counter operator, when its input trigger arc becomes active, the set up value is entered. The occurrence on an event/time input will decrement the count by 1. Whenever the count becomes zero, the output trigger arc is activated. Anytime the disable signal becomes active, the count is frozen and the node does not respond to the event/time input. The input trigger arc can set up the node only if the count is zero or if the node is disabled. The gate operator operates as follows: when enabled, the gate transmits, otherwise the gate blocks the propogation of the input trigger arc signal.

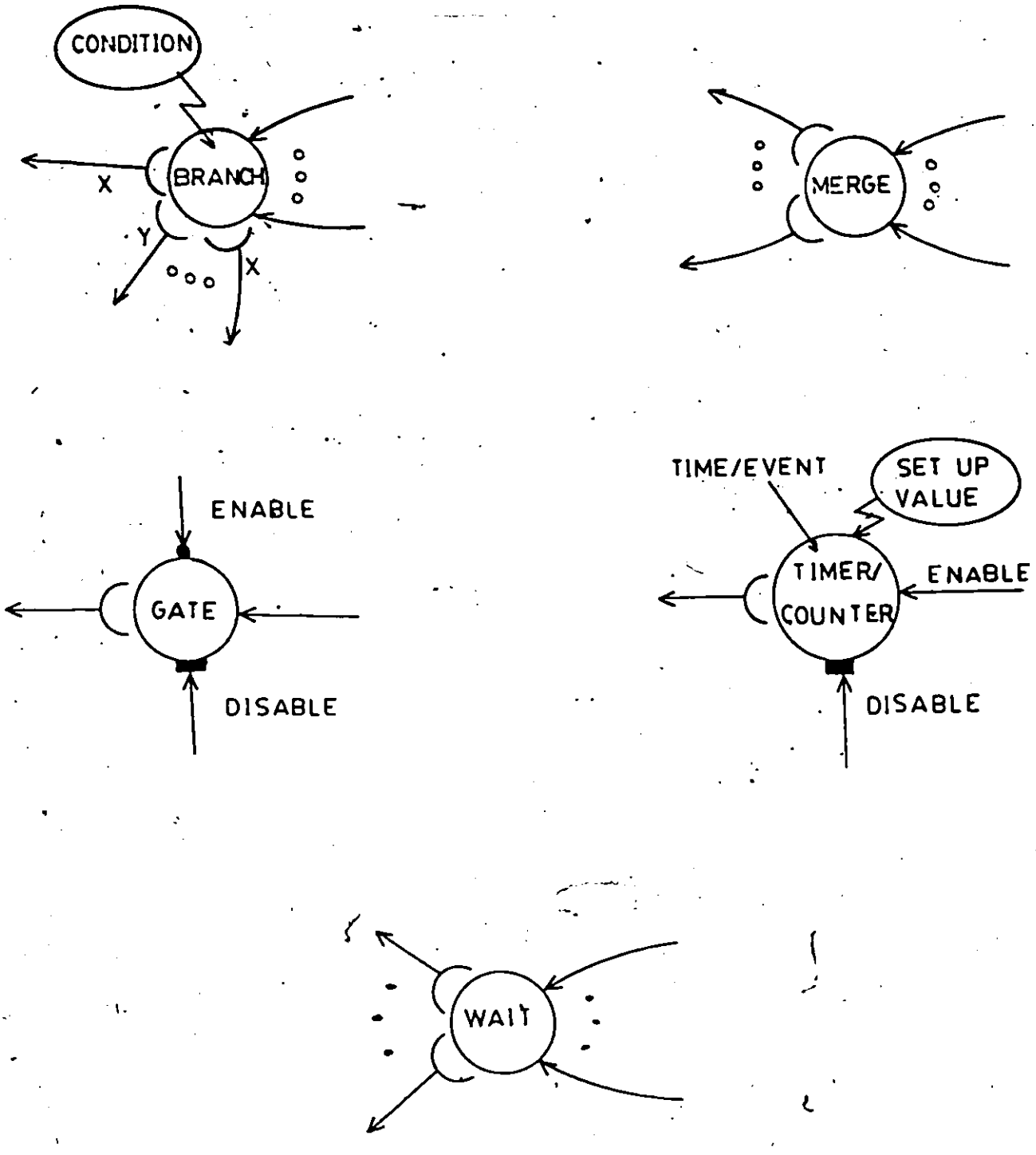

Figure 23: Sequencing operators

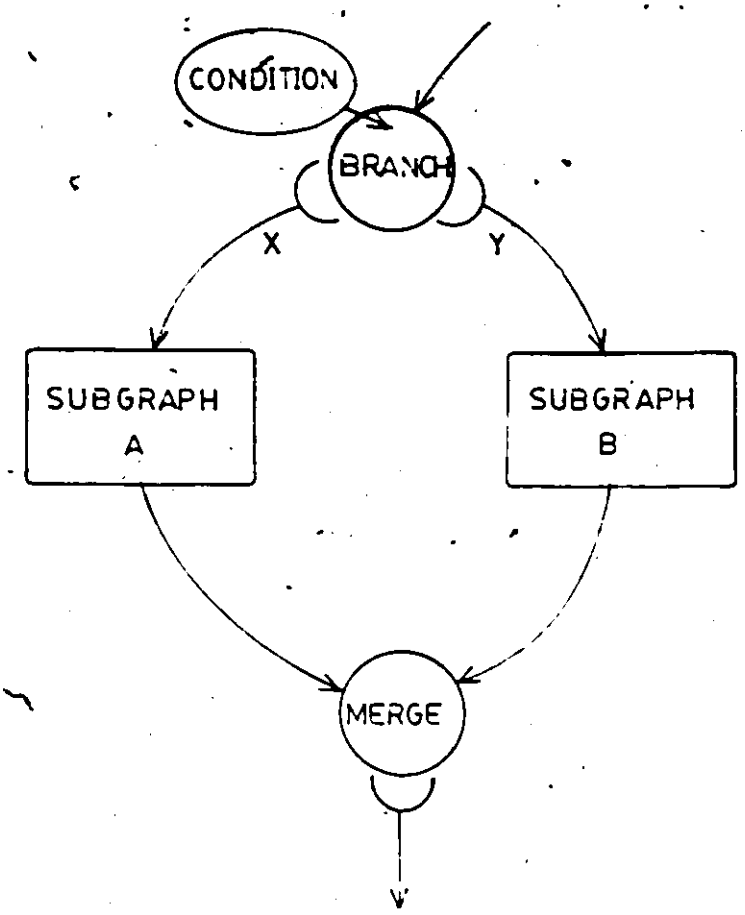

Figure 23: Conditional

5.1.2 Definition of the data base manager operation

In this subsection, the control actions of the data base manager are described. The descriptions are presented in terms of the system functions they represent. Pictorial descriptions using the previous defined graphical notation are used as the vehicle of presentation.

(1) Admitting a patient from admission

There are two types of admission: admitting a former patient or a new patient. In the case of a former patient, the history retrieval strategy must be transmitted along with the admitting request. While the data base manager waits for the admitting data, it sends a request to the on-line archival data base (OADB) controller to retrieve the patient profile and the sections of the medical history specified by the retrieval strategy. When both the retrieved data and the admitting data are received, they are combined to form the CMR. In the case of a new patient (i.e. no PMR exists for the admitting patient), the admission must also produce the patient profile and medical history. The data base manager transmits the CMR to the current data base (CDB) controller where it is inserted. Concurrently, the CMR is passed to the bus controller to be broadcasted over the central bus. The controller task structure diagrams for admitting a former patient and new patient are shown in figures 24 and 25 respectively.

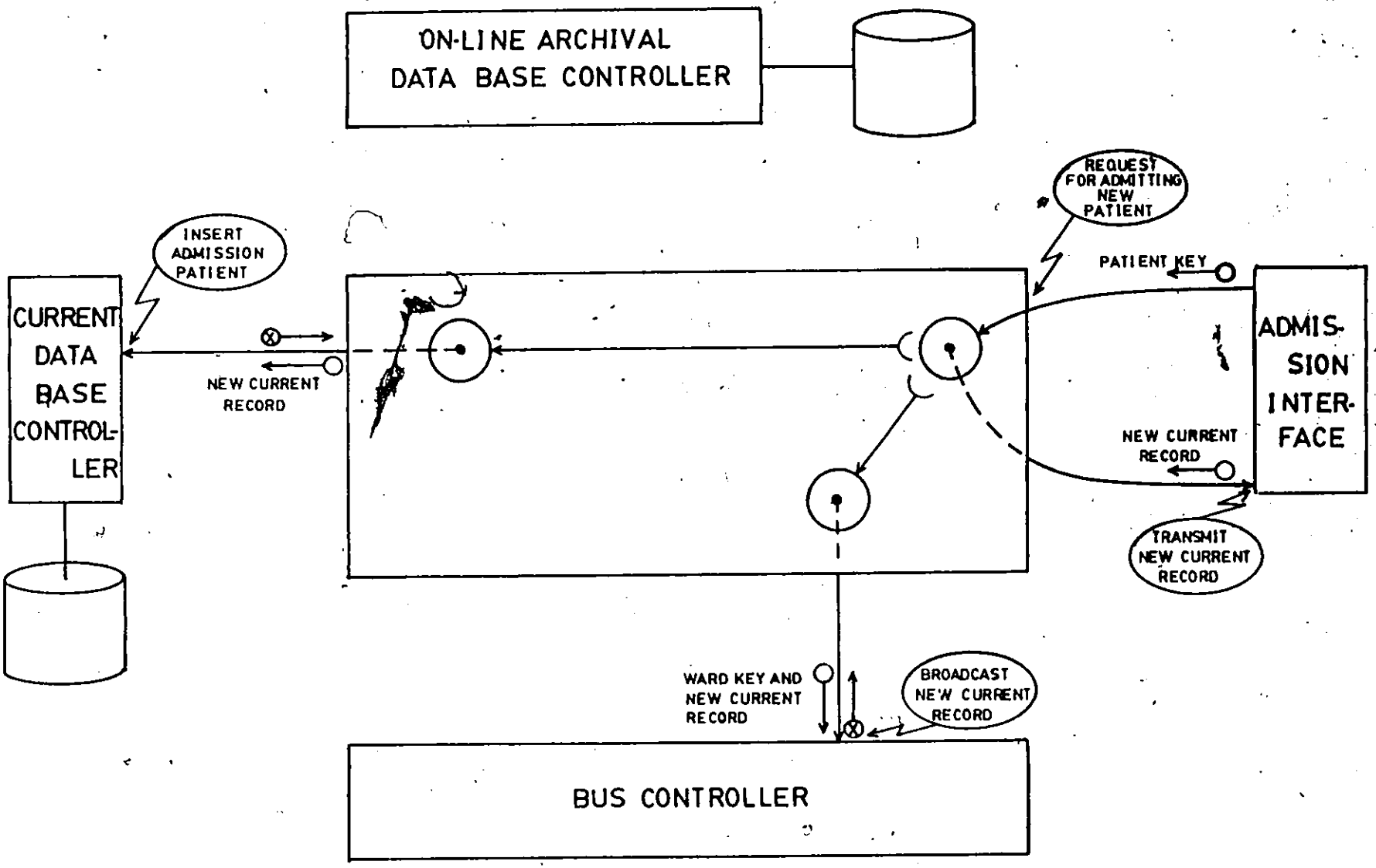

Figure 24: Admitting a new patient from admission

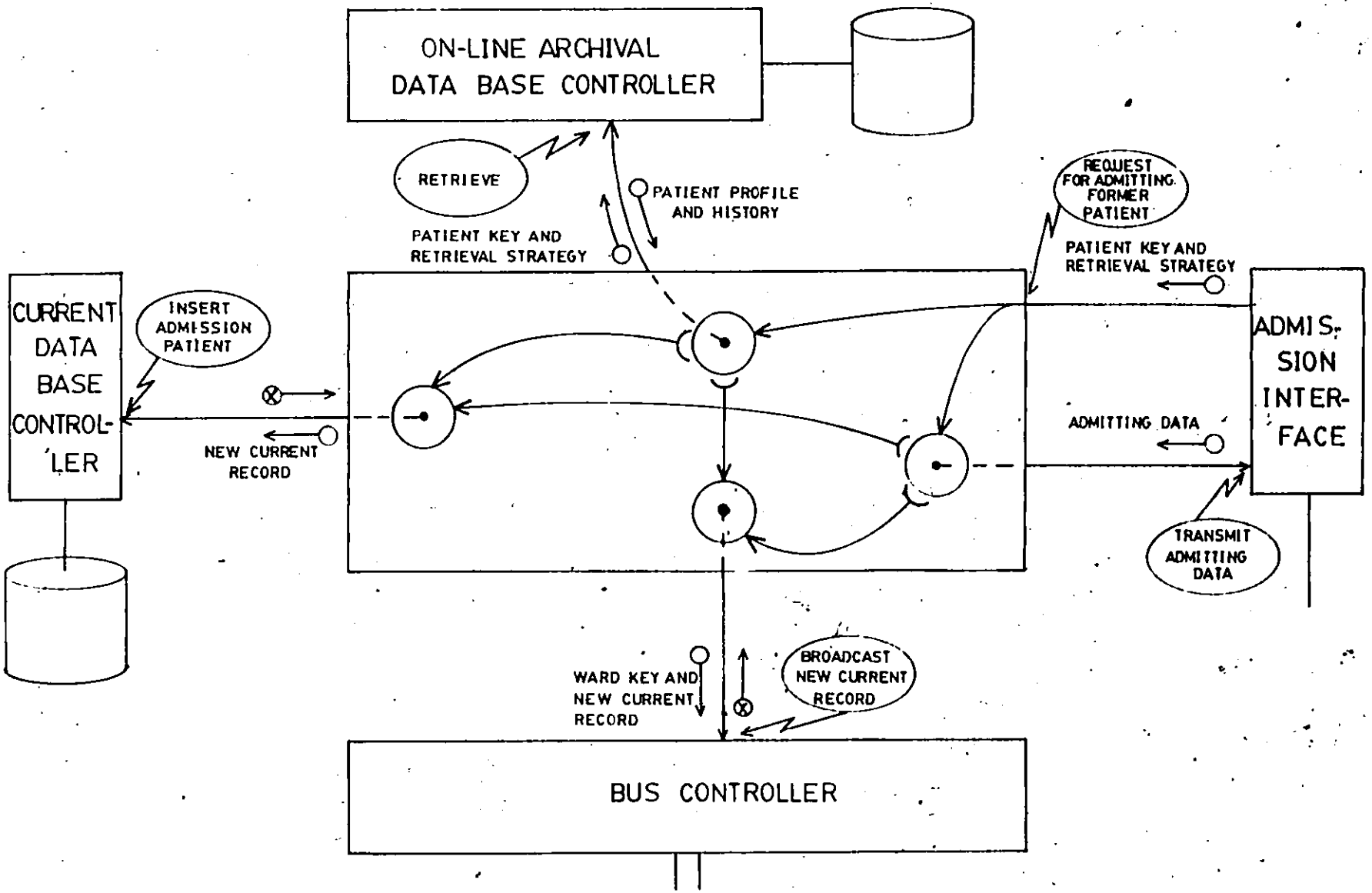

Figure 25: Admitting a former patient from admission

(2) Admitting a patient from emergency

Upon receiving an admission request from emergency, the first action of the data base manager is to send a request to the OADB controller to retrieve the patient profile and the sections of the medical history specified by the retrieval strategy. The retrieval strategy is transmitted along with the admission request over the central bus.

If the patient has an on-line archival record (PMR), then the data base manager sends the retrieved data to the bus controller to be broadcasted over the central bus and to the CDB controller where it is inserted. Whenever the admitting data becomes available from emergency, the bus controller transmits the admitting data to the data base manager so that the CDB be updated. Note that the bus controller must broadcast this data so that the local data bases can be updated.

If no PMR was found, the data base manager signals the bus controller to obtain from emergency the critical admitting information. This information is immediately transmitted to the CDB controller where it is inserted. Whenever the data base manager receives the remaining CMR information from the bus controller, it transmits this record to the data base manager to update the patient's CMR. Note that the bus controller must broadcast both the initial CMR and the remaining CMR information so that the local data bases

be properly updated. The corresponding structured graphs for admitting a former patient and a new patient are shown in figures 26 and 27 respectively.

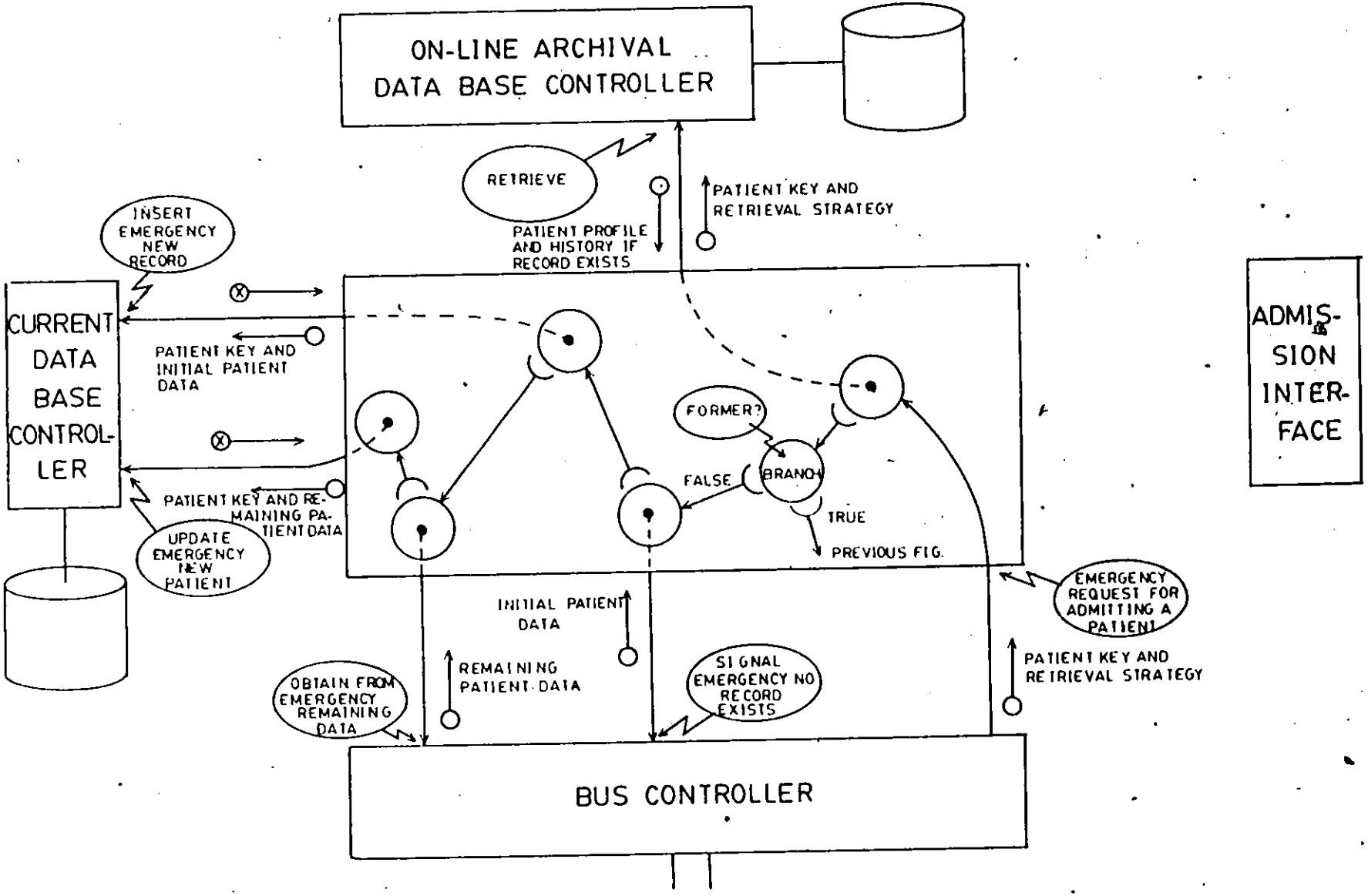

Figure 26: Admitting a new patient from emergency

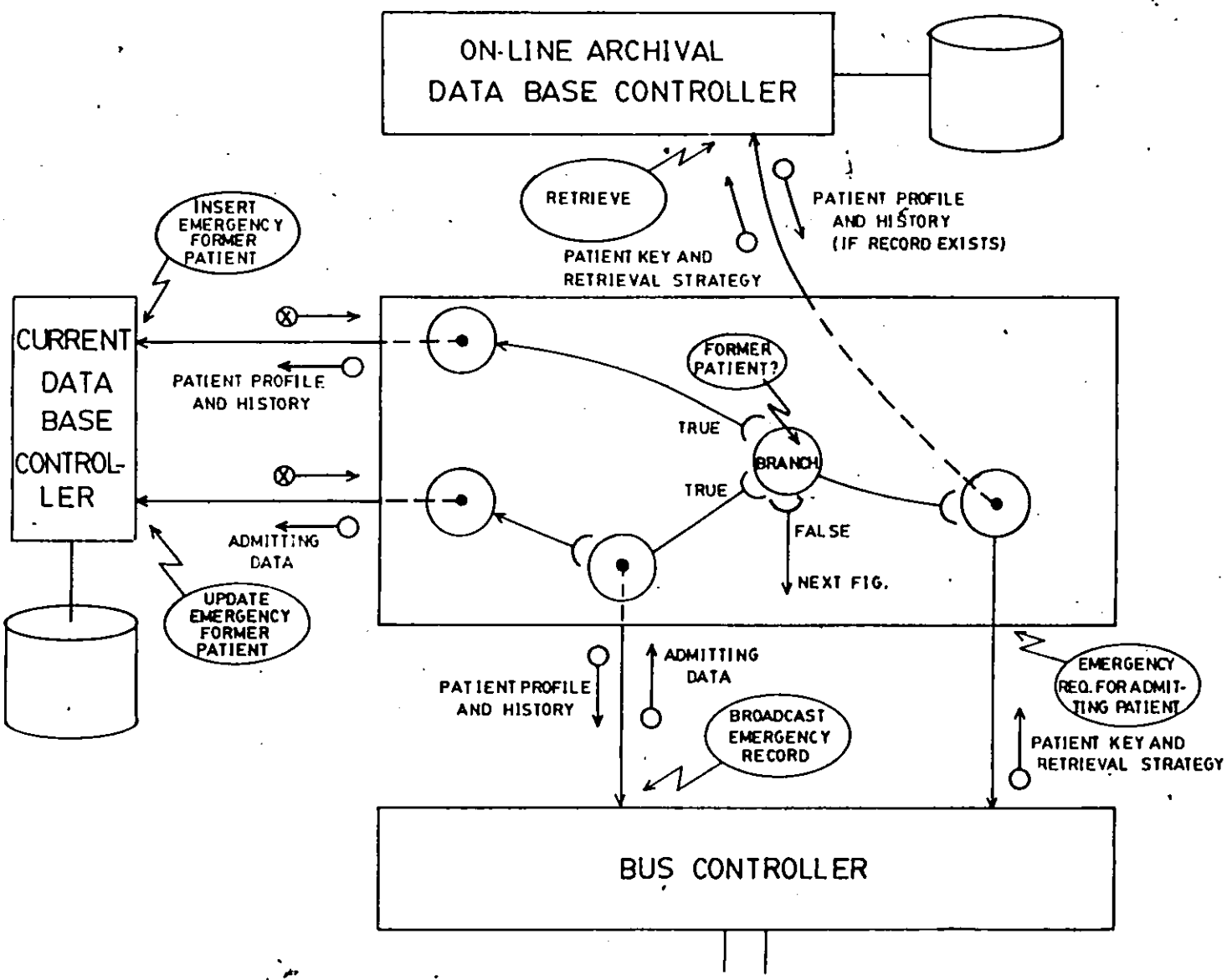

Figure 27: Admitting a former patient from emergency

(3) Service request

Whenever the data base manager is signaled from a ward for a service request, its first action is to check the validity of the request. If valid, the data base manager requests the bus controller to inform the initiating ward to broadcast the service request. When the bus controller receives the service request, it transmits it to the data base manager who sends an update request to the CDB controller to stored the service request. If the service request is not valid, an error message is sent to the initiating ward. The pictorial description of this process is illustrated in figure 28 .

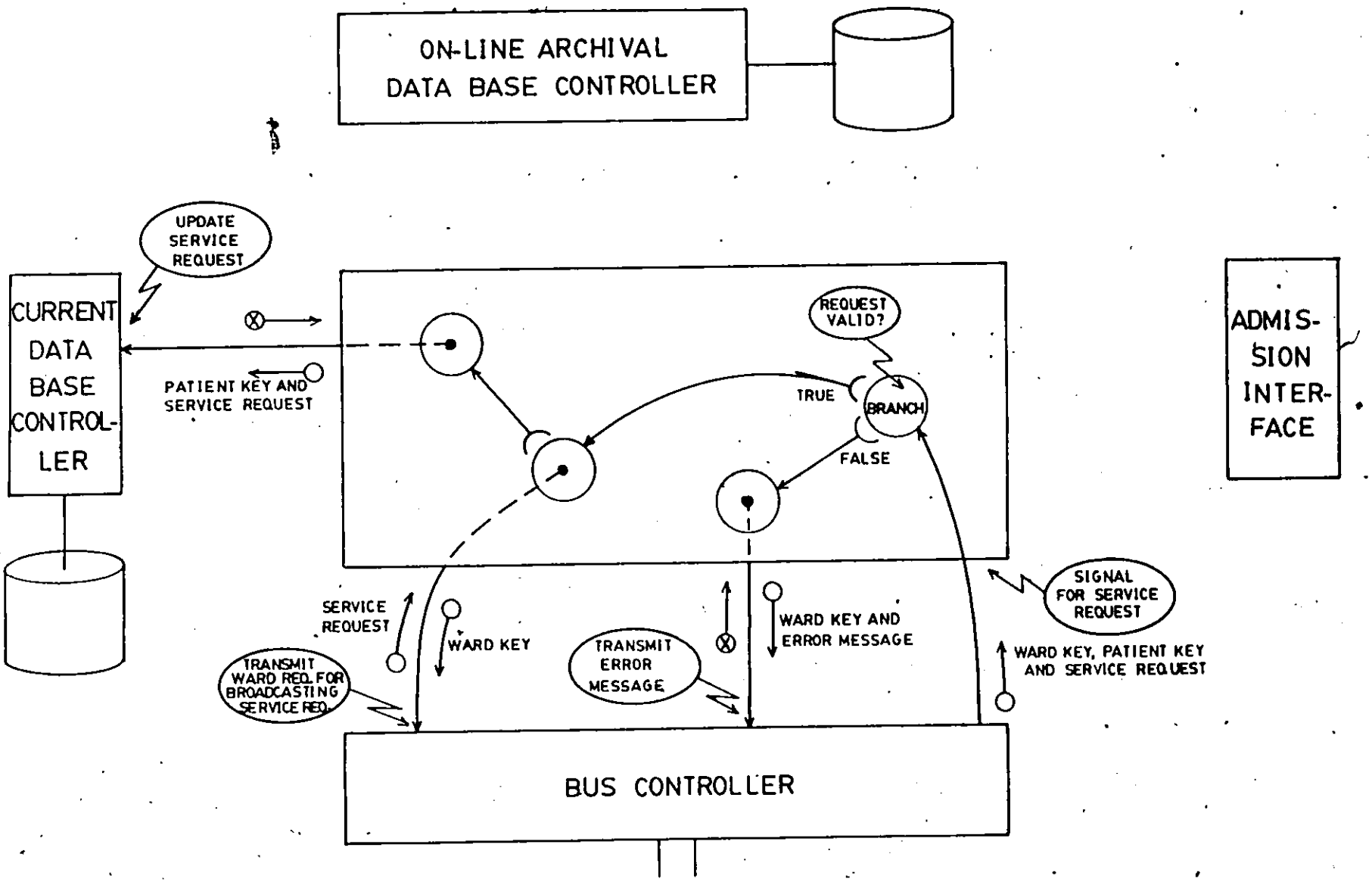

Figure 28: Service request

(4) Update Request

The data base manager's first action consists of determining if the request is valid (i.e. to make certain that the person who made the request is allowed to do so). If the request is valid, the data base manager signals the bus controller to request the update data from the initiating ward. If it is a service update, then this data is also broadcasted to the service stations. When the update data (ward or service response) is received by the data base manager (via the bus controller), it transmits this data to the CDB controller to update the patient record. The corresponding structured graph is illustrated in figure 29 .

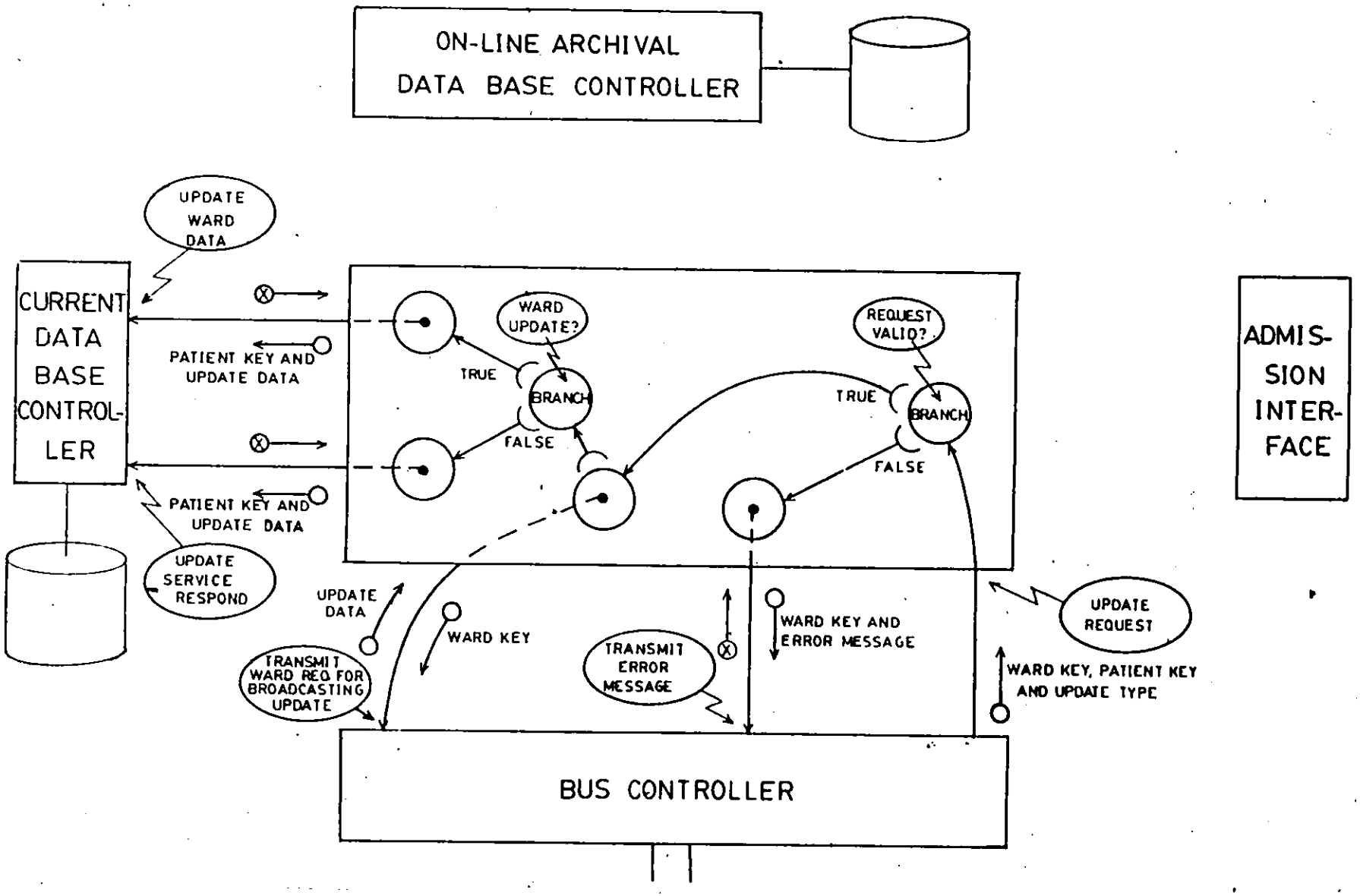

Figure 29: Update request

(5) Additional Information Request

The additional information request signal is transmitted from a local work station (ward or service) to the database manager via the bus controller.

Before submitting the request to the CDB or the OADB controller, the data base manager verifies the validity of the query request.

If valid, then it determines to which data base (OADB or CDB) to issue the query request. Furthermore, each data base controller performs verification to protect against unauthorized accesses. If the query request is accepted, the requested patient data is sent to the data base manager who signals the bus controller to transmit this data to the requesting work station.

If the request has been rejected by the data base manager or by the data base controller, then an error message is sent to the requesting work station. The corresponding structured graph is shown in figure 30 .

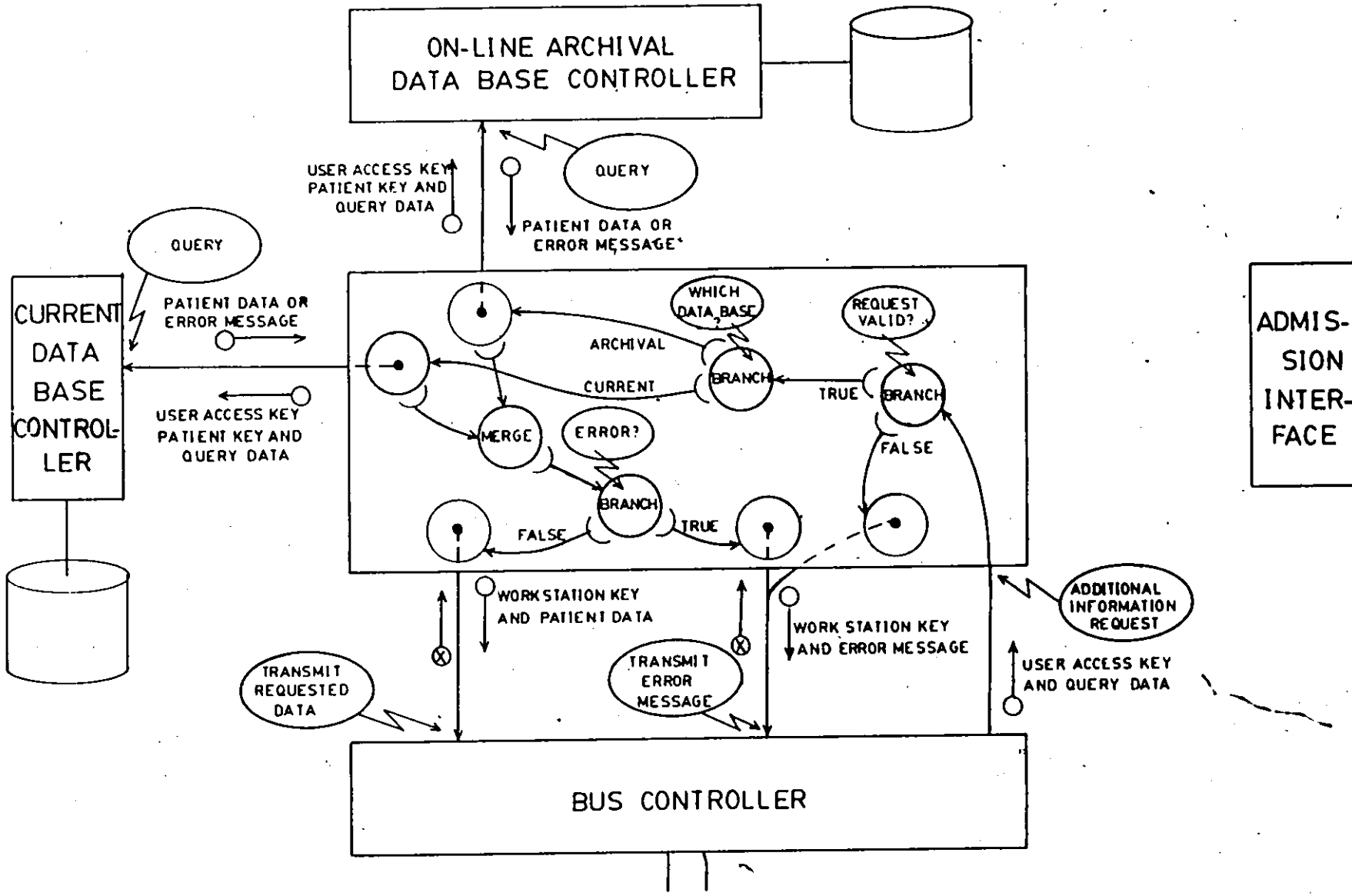

Figure 30: Additional information request

(6) Releasing a patient

This process is initiated by a ward. The ward sends a request to the data base manager to release a patient over the central bus. First, the database manager checks for proper authorization. If valid, it transfers the request to admission. When admission signals the data base manager to proceed with the release process, the first step is to determine whether there are outstanding service requests. The database manager transmits a query to the CDB controller and concurrently signals the bus controller to broadcast a request to check for outstanding service requests.

If no outstanding service requests exist, the data base manager signals the bus controller to broadcast a command to delete the CMR of the patient who is being released. When the bus controller obtains the on-line archival update record from the patient's ward, it transmits it to the data base manager who sends the on-line archival update data to the OADB controller to update the patient's PMR. For reliability, the patient record in the CDB is not deleted until the update of the OADB has taken place. The pictorial description of such a process is shown in figure 31 .

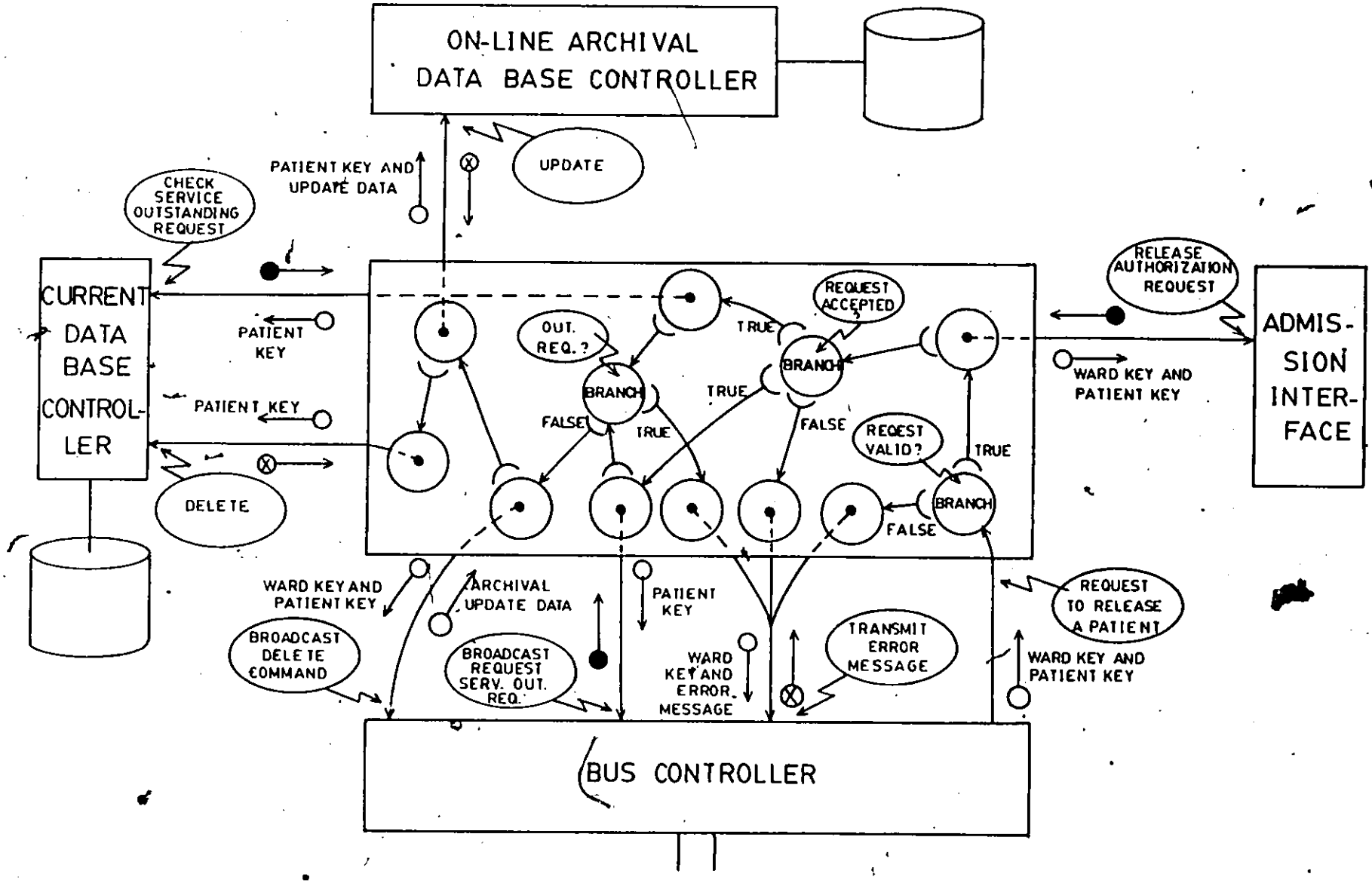

Figure 31: Releasing a patient

(7) Transferring a patient

Upon receiving a transfer request from a ward, the data base manager checks the validity of the request. If valid, it transfers the request to admission. Whenever the data base manager is requested by the admission to proceed with the transfer process, it sends a request to the bus controller to inform the originating ward to transmit the update data and the on-line archival data. Concurrently, it sends a request to the OADB controller to retrieve the additional history information specified by the transfer request. If the transfer is found not valid by the database manager or not approved by the admission, an error message is sent to the initiating ward.

When the on-line archival updated data is received, the data base manager issues an update request to the OADB controller. Furthermore, when both the retrieved patient history and the updated current data are available, they are combined to form the new CMR. This CMR is sent to the CDB controller to update the patient's record. Concurrently, the data base manager issues a request to the bus controller to broadcast the CMR. The corresponding graph is shown in figure 32 .

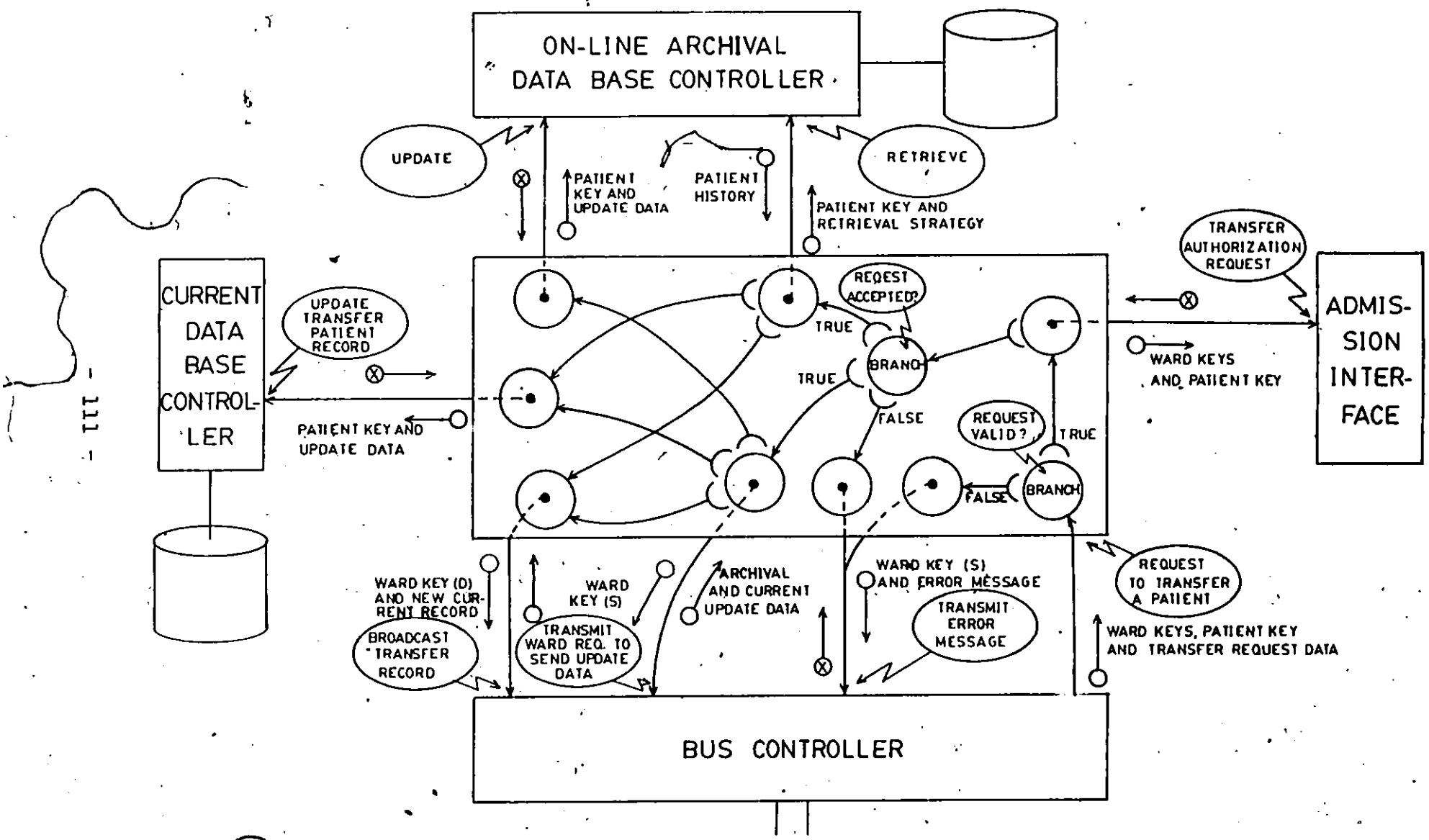

Figure 32: Transferring a patient

5.2 IMPLEMENTATION OF THE DATA BASE MANAGER

The data base manager must posses the following properties. First, the data base manager must be efficient in order to provide a fast response for critical requests. Secondly, the data base manager must be reliable since its failure can stop partially (or completely) the operation of the system. Finally, the data base manager should be upgradable in order to adapt the system to new demands (e.g. the addition of new work stations).

The data base manager will be implemented as an oligarchical system [KRIE 82]. An oligarchical system is a multilayered organization whose lowest level is composed of entities (processing elements) and all other layers are comprised of coordinators (system control elements). There are n>1 control levels where each coordinator at level i is directly connected to several coordinators at level i-1 (in the case of i=1, the coordinators are connected to the entities). Furthermore, each coordinator is connected to at least one coordinator at level i+1 (except in the case of i=n which corresponds to the top level). In this model, there are three kinds of communication primitives : requests, control signals, and transfers. All requests are sent to a higher level, control signals are sent to lower levels and data transfers are sent within the same level.

5.2.1 Data base manager organization

In this subsection, a possible mapping of the data base manager based on oligarchical control is presented. The first design step consists of partitioning the entities into groups and associating a coordinator to each group. The partitionning of the entities schould be done according to their level of interactions (i.e entities having a high degree of interactions between them should be connected to the same coordinator).

As outlined in chapter four, the overall data base is organized in a layered structure where the information transfers between the different layers are controlled by the data base manager. From this specification, two entity groups can be indentified. One group consists of those entities implicated in the data transfers between the CDB and the local data bases. These entities are the CDB controller and the bus controller. The other group consists of those entities implicated in the data transfers between the CDB and the OADB. These entities are the CDB controller, the OADB controller and the admission interface. Coordinators, referred to as task driver "1" and task driver "2", are assigned to each group. Task driver "1" controls all accesses to the CDB from work stations. It also acts as the controller for interactions between the work stations. In addition, all data transfers between work stations and task driver "2"

are controlled by task driver "1". Task driver "2" controls all accesses to the OADB. It also controls the admission interface since most admission requests involve access to the OADB. Furthermore, all data transfers between the CDB and OADB, and the admission are coordinated by this task driver.

At the above level, a coordinator (the task manager) is allowed to manage both task drivers. The task manager will (1) provide controlled interaction between both task drivers, (2) control accesses to shared resources (e.g. global memory, CDB). The organization of the data base manager is illustrated in figure 33 .

In order to provide fail soft capability, the task manager should be able to take over the duties of both task drivers, and each task driver should be capable of taking over the functions of the task manager and the other task driver.

The information flow in the data base manager is now described. In an oligarchical system, all information transfers occur at the bottom level. More specifically, all data involved in any transaction are stored in the initiating processing element until the transfer to the target resource can be completed. All requests are initiated by the processing elements and are directed to the task drivers. Depending on the actions necessary to service the request, the task driver will either transmit the proper control signals to the bottom level or send a request to the task man-

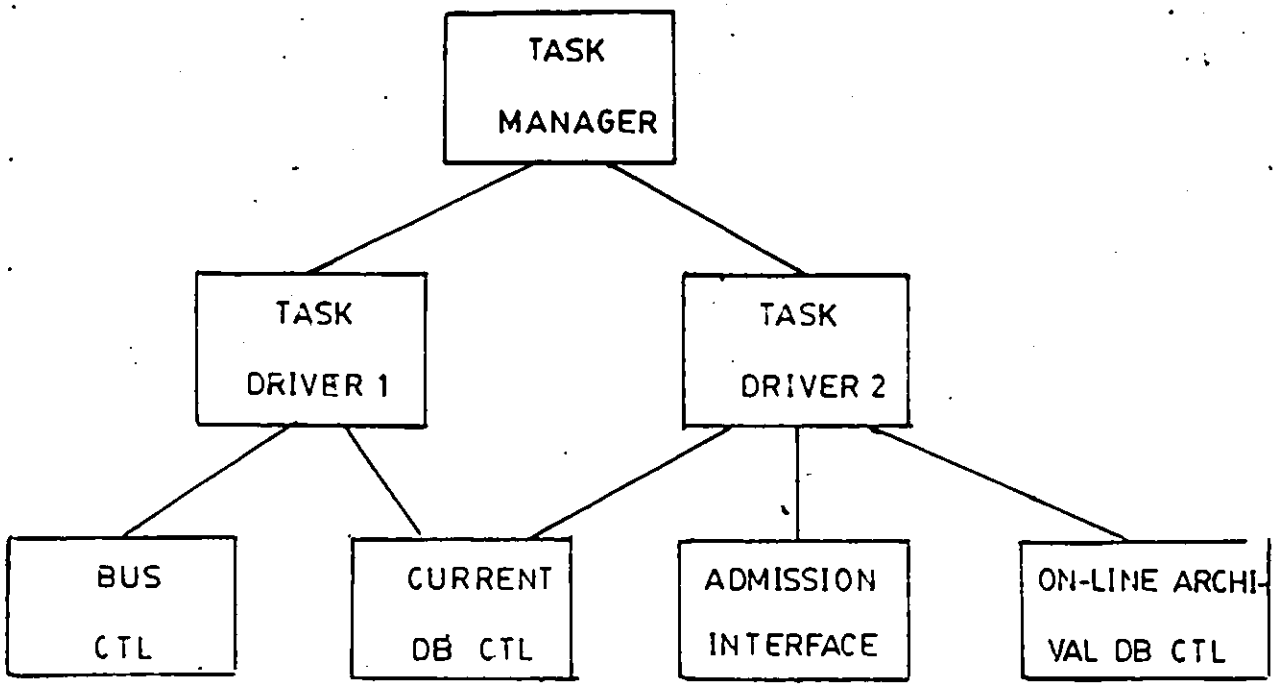

Figure 33: Data base manager organization

ager. In a normal mode of operation, the task manager only receives requests from the task drivers and only sends control signals to the task drivers. Task drivers "1" and "2", and the task manager must maintain a queue for all requests involving the system resources that they control (e.g. controllers, memory).

Next, a suggested bus organization for the bottom level is described. In order to take advantage of the parrallelism existing between the processes, each task driver has a separate bus interconnecting itself to the elements it controls. The bus associated with the task driver "1" connects the bus controller and the CDB controllers. The bus associated to task driver "2" connects the CDB and OADB controller, and the admission interface. In order to store temporary data, each bus has a buffer connected to it. Furthermore, a dual port memory is connected to both buses. This will allow data transfers between the bus controller, and either the OADB controller or the admission interface. For the other levels, the elements of the data base manager (task manager, task driver "1", task driver "2") must be interconnected in such a way that each unit can take the duties of the other in the instance of a failure. The complete block diagram of the data base manager is illustrated in figure 34 .

Access to buffers must be coordinated since they are shared by concurrently executing processes. Comunications between the units of the data base manager can be reduced: (1) by having each task driver direct control over the buffer associated with it, and (2) having the task manager direct control over the common memory.

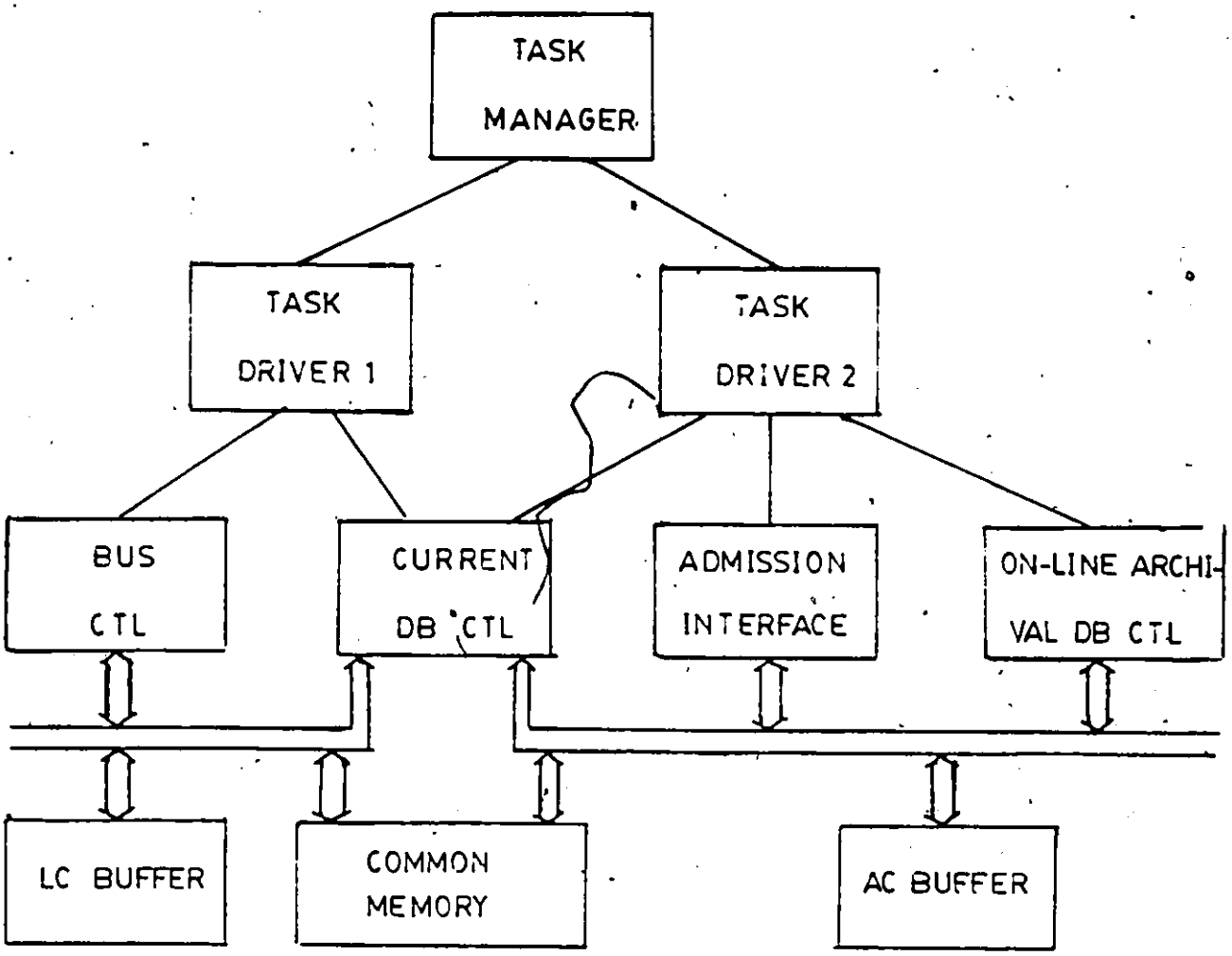

Figure 34: Bus organization

5.2.2 Implementation aspects of the data base manager's units

In this section, the implementation aspects of each element of the data base manager are presented.

Both task drivers are similar in the sense that they have the same functional requirements. The general organization of a task driver is shown in figure 35 . The look-up table essentially stores finite state machines for each possible request. These machine states can be of four types:

(1) Initiating a controller task

(2) Sending a request to the task manager

(3) Executing internal tasks such as verifying the validity of requests, determining to which data base to issue a query request.

(4) Initiating data transfers between the buffers and the common memory. One method of accomplishing this is to include a DMA channel between the common memory and the buffers.

The list of currently executing requests contains the request identifier, request parameters and request execution status for each request that is being executed. Corresponding to each type of activity, there is a queue listing the

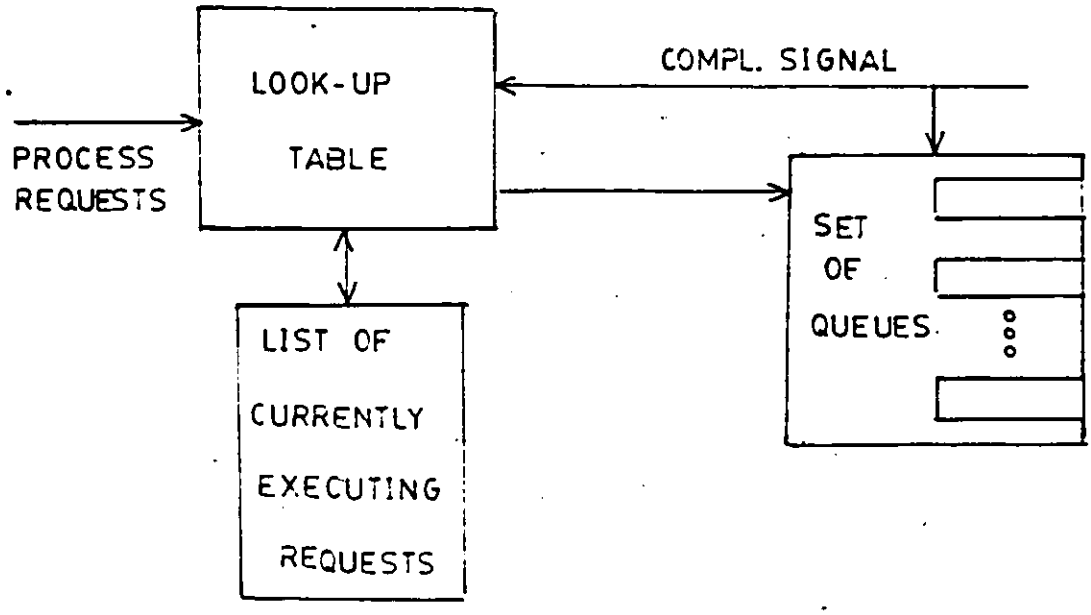

Figure 35: Task driver organization

activities ready to be executed. Each query entry includes the activity parameters and an identifier to indicate the initiating request. Semaphores will provide synchonization between units over shared resources. A controller semaphore is set whenever a task is activated on the corresponding controller and is reset whenever the task driver receives a signal indicating that the task is terminated.

To describe the operation of a task driver, we will describe the cycle of activities that occur in the execution of a request. In the case of task driver "1", a request to a a task driver to start a set of activities can be initiated either by the task manager or the bus controller. In the case of task driver "2", the admission interface or the task manager ca originate the request.

Upon receiving such a request, an entry is created in the list of currently executing requests and the first activity from the table of actions is transmitted (with the appropriate parameters and request identifier) to the corresponding queue.

The execution of each activity requires the availability of one or more resources (where a resource can be internal or external). Whenever a resource is no longer required, the following actions must take place: (1) the resource semaphores must be signal, (2) the look up table is checked to determine the next activities to be initiated and (3) the status field of the list of currently executing requests is updated. The next activities waiting on the reset semaphores must be initiated. This cycle continues until all queues are empty.

In the above discussion, we have assume that all activities have the same level of priority. A priority mechanism can be easily implemented in this organization. For in-

stance, critical tasks can be inserted at the front of the queue instead of the end of the queue.

The task manager coordinates accesses to the CDB and the common memory. Furthermore, communications between both task drivers is done through the task manager. The task manager can be organized in the same fashion as the task drivers. In a normal mode of operation, it can only receive requests from the task drivers. These requests can be of two types: communication requests and requests for access to resources under its control. Note that the assignment of the common memory and the CDB can be done according to a priority scheme.

Because of the complex requirements of the task drivers, each task driver could be implemented as a task driven multi-processor system [KRIE 81]. A task-driven multi-processor system consists of a number of processors each having its own local memory and some I/O capability. In addition, the system may have global resources such as global memory for passing messages and for storing common variables. All resources available to the system, including the processors, are controlled by a dedicated unit call the executive.

5.3 CONCLUSION

The purpose of this chapter was to present the design of the of the data base manager. This was done by defining for each of the system proccesses, the various controller tasks and the sequencing relationships between these tasks. A graphical notation was introduced to describe the operation of the system.

The data base manager being the critical element of the suggested layered data base system, has to be implemented as a highly reliable multi-processor system. This was achieved by the defining the data base manager as an oligarchical system. Roughly speaking, this required the definition of the coordination of task interactions in a hierarchical fashion. This was done by identifying the function of two task drivers and a task manager.

Chapter VI

SIMULATION

In order to show the feasibility of the system, we have simulated the operation of the data base manager. The simulation demonstrates that the data base manager allows the various controllers to operate concurrently to execute simultaneous processes. The first section presents a brief description of how the simulation was performed. Then, the actions of both task drivers to coordinated the execution of each system process are defined. Finally, examples of the simulation are presented.

6.1 SIMULATION DESCRIPTION

To provide concurrent operation between both task drivers, each task driver was allowed to run sequentially using equal time-slices. In this system, the major functions of the task manager consist of controlling accesses to the current data base and to the common memory. These functions

were simulated by using semaphores for controlling accesses to those shared resources.

Each task driver has a buffer for storing temporary data belonging to different processes which can be executed in parallel. This means that at any moment, the buffers can have data belonging to different processes and flowing in different directions. In order to avoid deadlocks that can result from the fact that data in the buffer can flow in two different directions, the following allocation scheme is used. Each buffer is segmented into two parts where each segment stores data flowing in one direction. This is a very simple scheme and one could develop a more sophisticated scheme that could achieve higher system performance. With respect to the common memory, the only restriction is with respect to size. Only data belonging to two processes are allowed to reside simultaneously in the common memory. The simulation listing is included in appendix a.

6.2 ACTIVITIES OF TASK DRIVERS 1 AND 2

In this section, we will indicate for each task driver, the sequence of activities required to process each system function. The above is achieved by introducing a special graphical notation referred to as an "activity graph". Basically, this graphical notation consists of a flowchart

with the addition of a "wait" operator as shown in figure 36. In this graphical notation, the execution box, conditional box and collecting point are represented in the same fashion as in a flowchart. A link (i.e. control path) that has a wait operator as its destination is represented by a dotted line. Otherwise, the links are represented by a solid line. Whenever a process reaches (through a dotted line) a wait operator, it enters into a wait state. The process is allowed to continue when all input trigger arcs of the wait operator are set. Using the above notation, the following subsections present the sequence of activities required to process each system function.

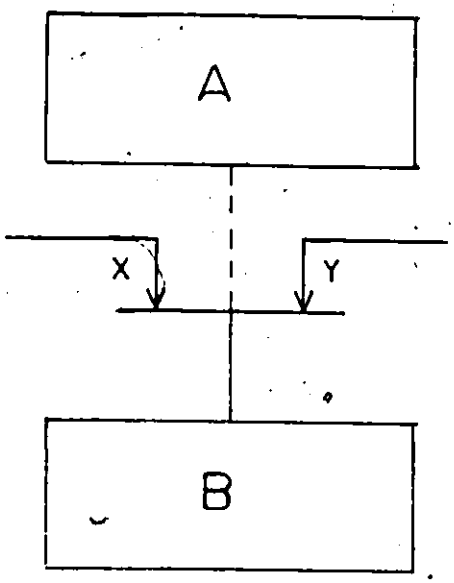

Figure 36: Wait operator

6.2.1 Activity graphs of task driver "1"

The actions performed by the task driver "1" to coordinate the execution of each system process, are presented separately by the following activity graphs.

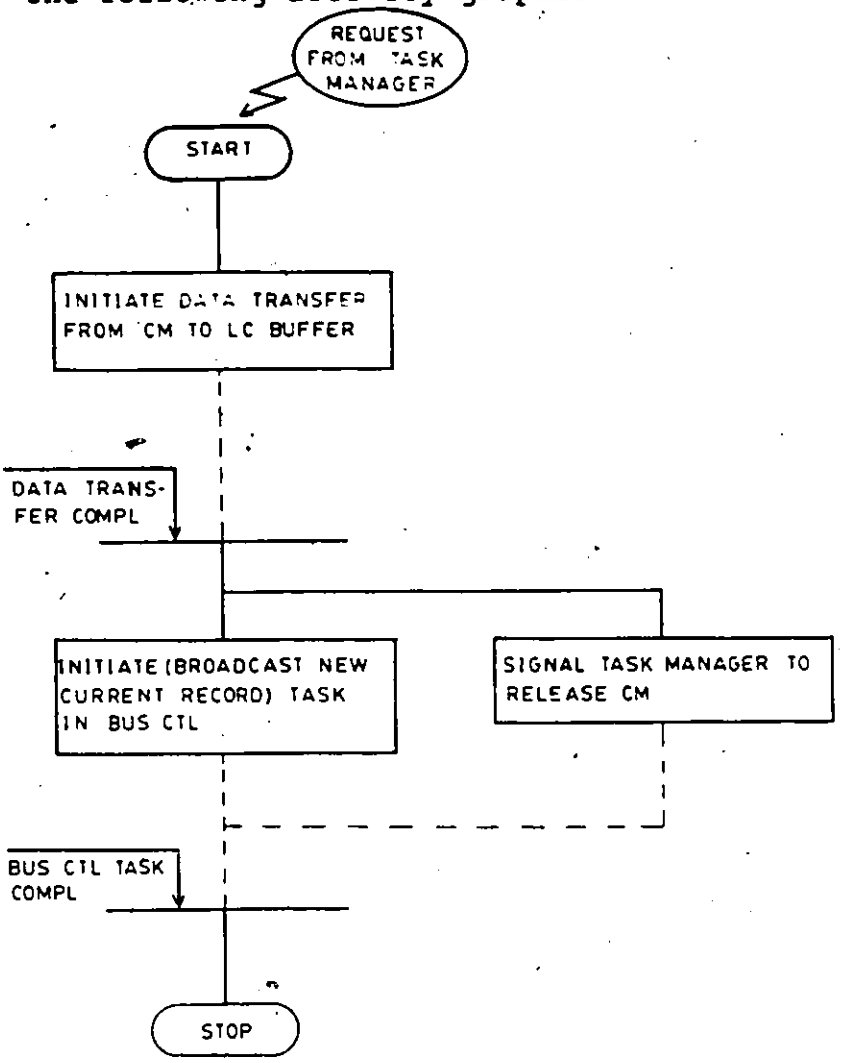

Figure 37: Admitting a patient from admission

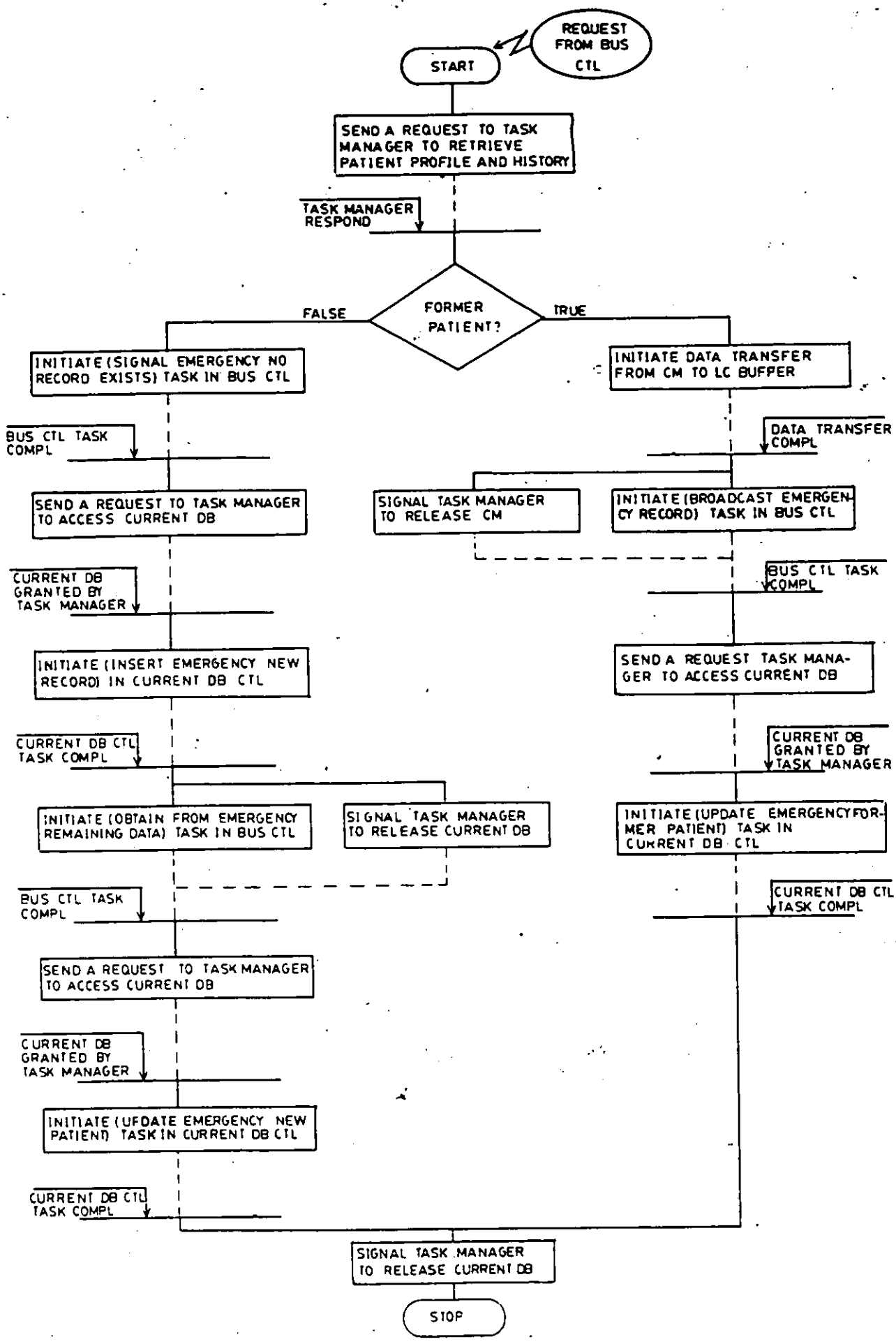

Figure 38: Admitting a patient from emergency

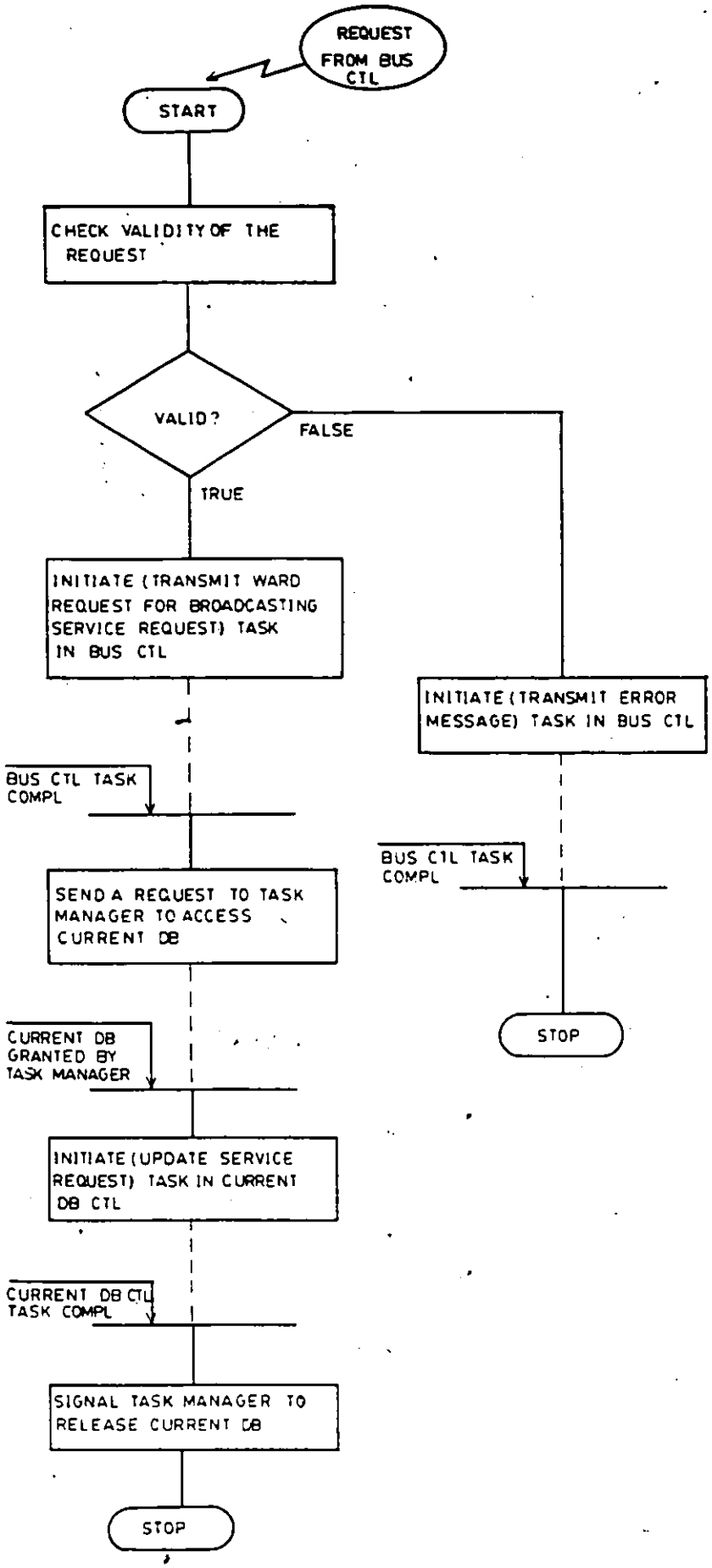

Figure 39: Service request

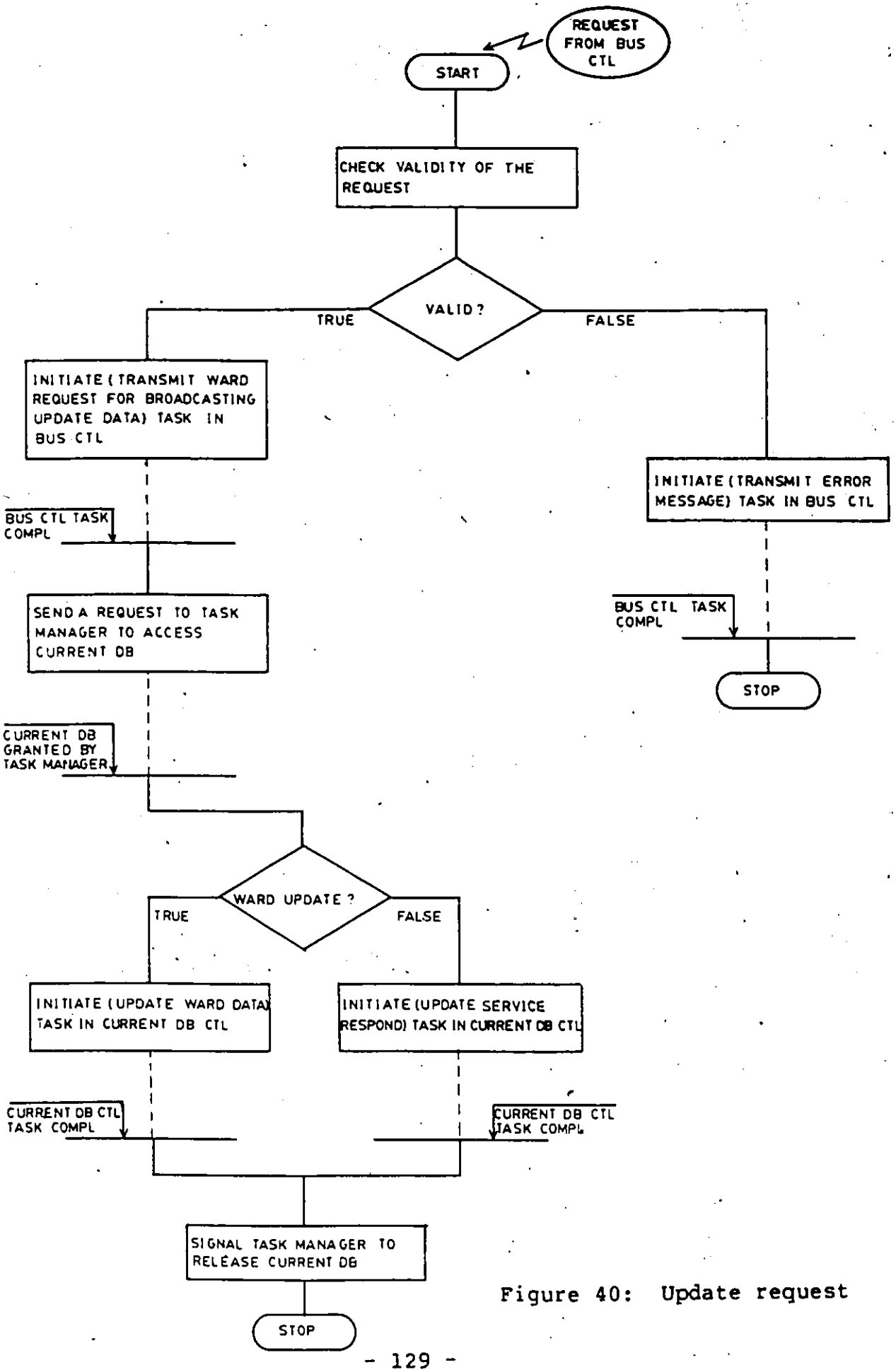

Figure 40: Update request

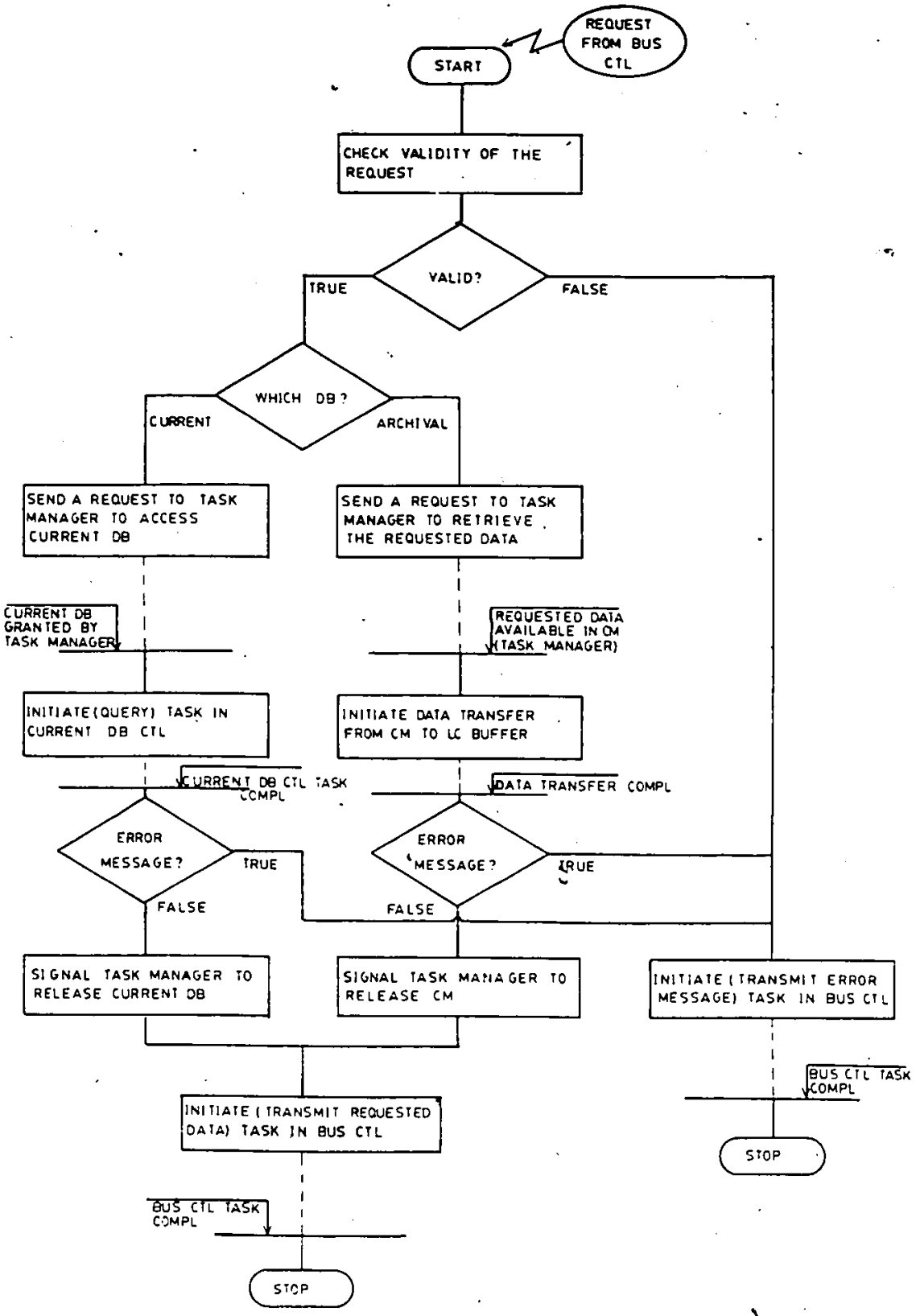

Figure 41: Additional information request

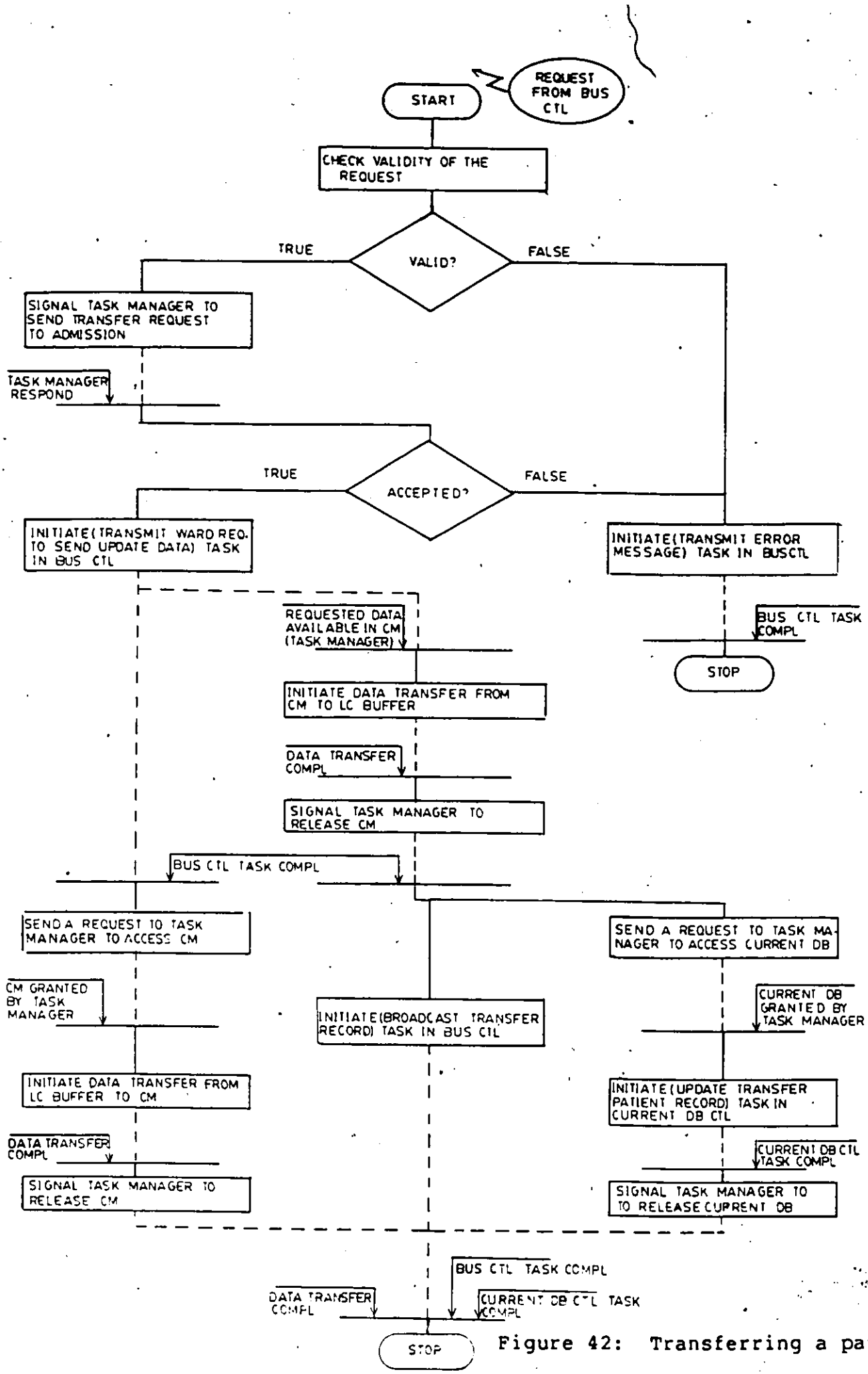

Figure 42: Transferring a patient

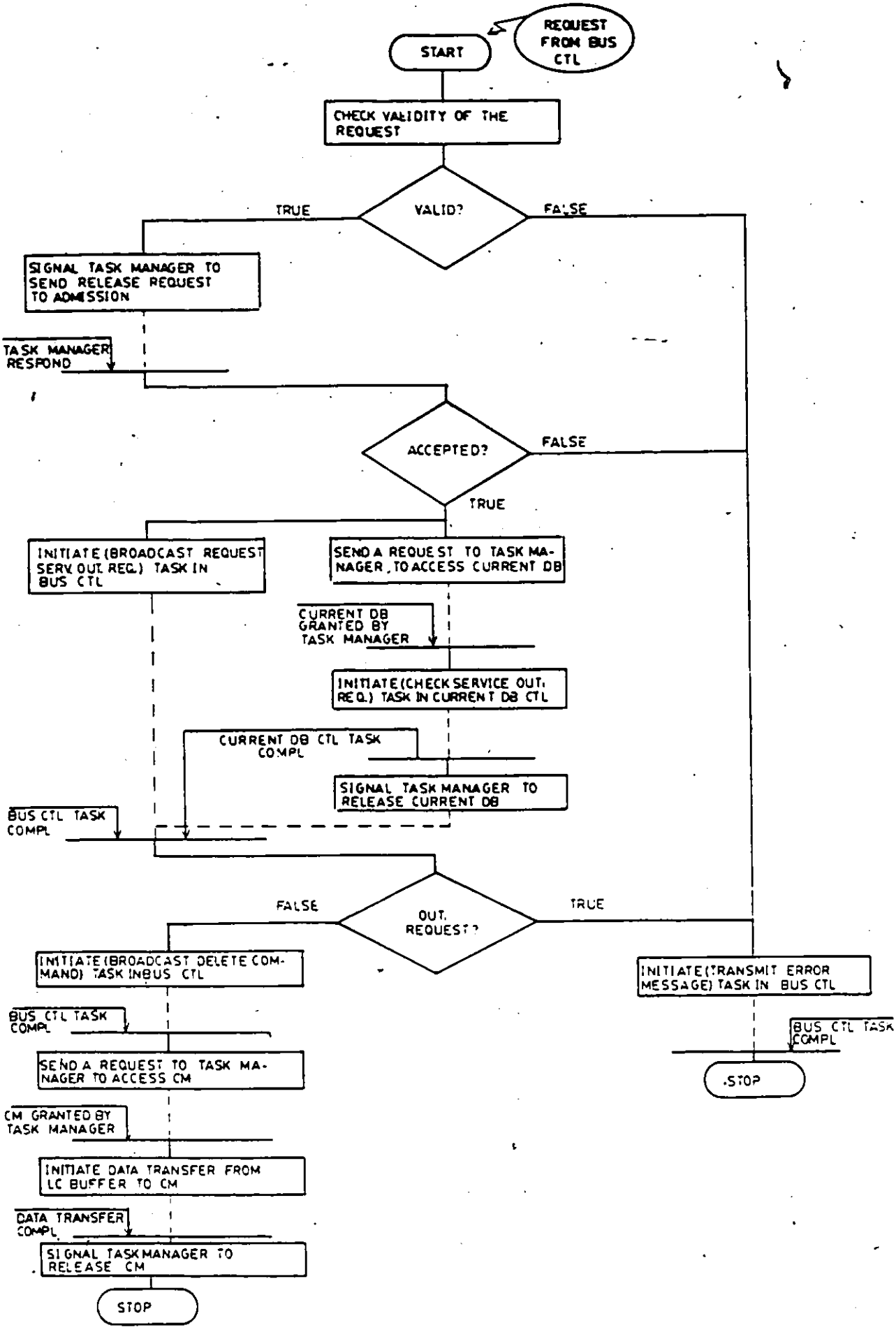

Figure 43: Releasing a patient

6.2.2 Activity graphs of task driver "2"

The actions performed by task driver "2" to coordinate the execution of each system process, are presented separately by the following activity graphs.

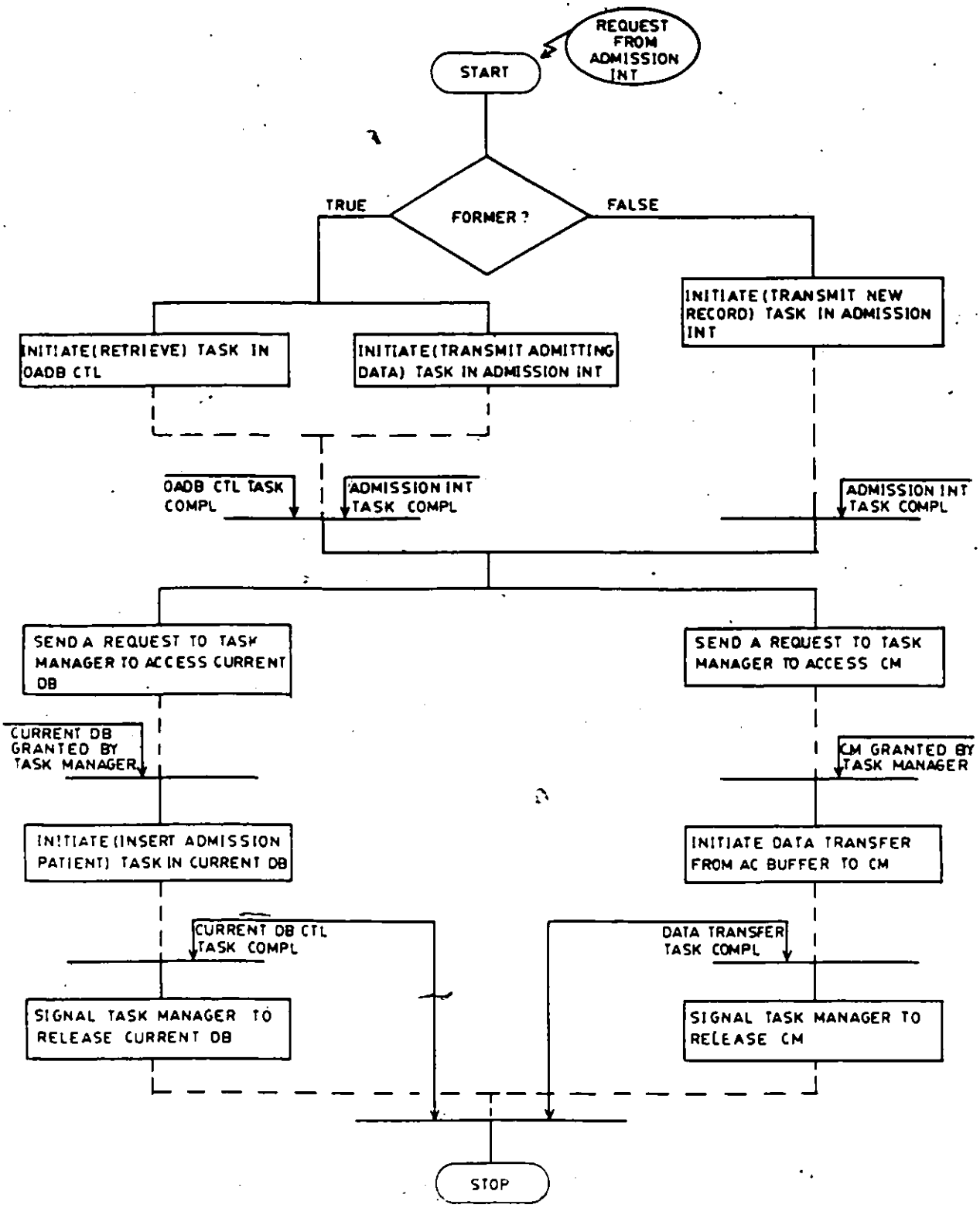

Figure 44: Admitting a patient from admission

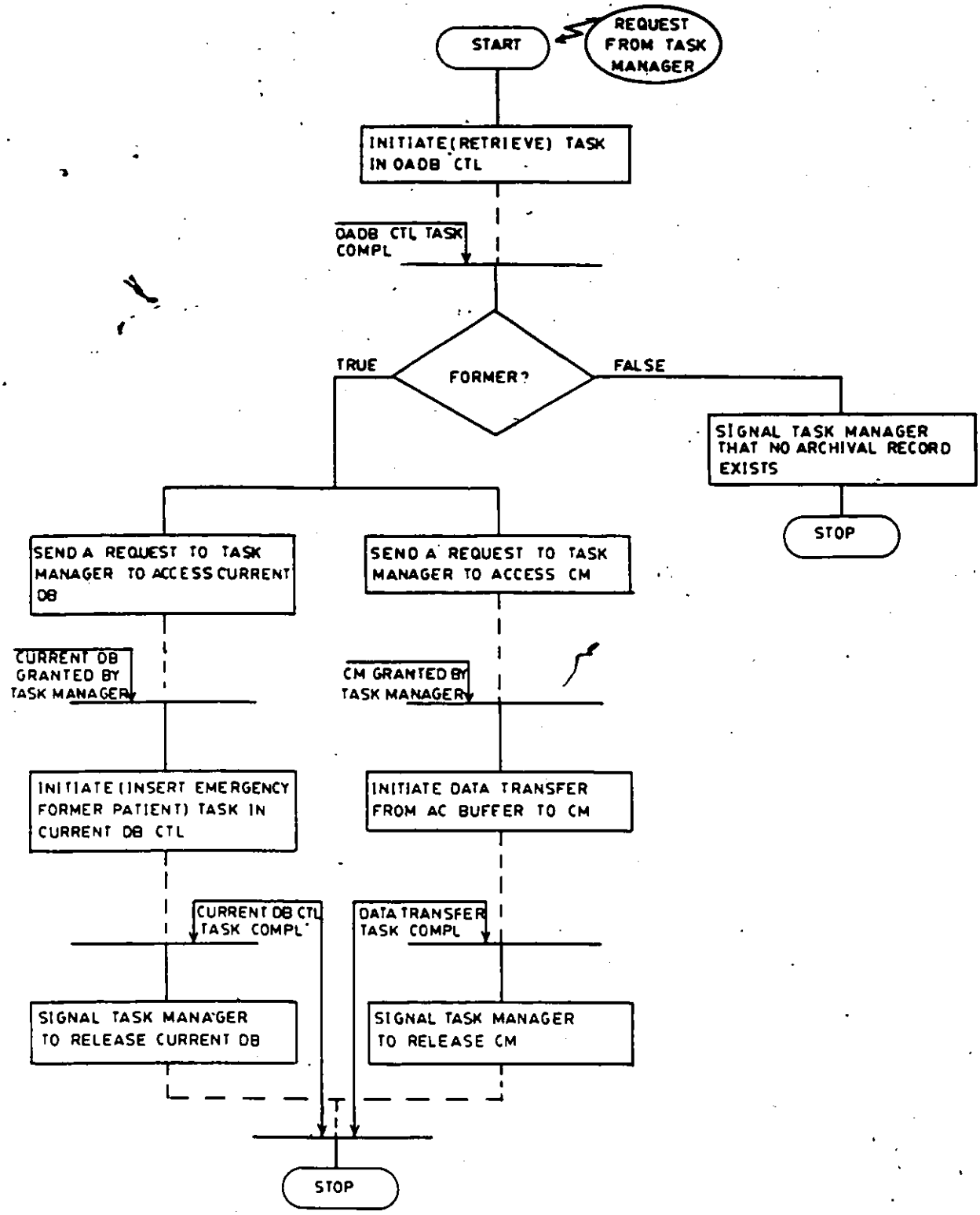

Figure 45: Admitting a patient from emergency

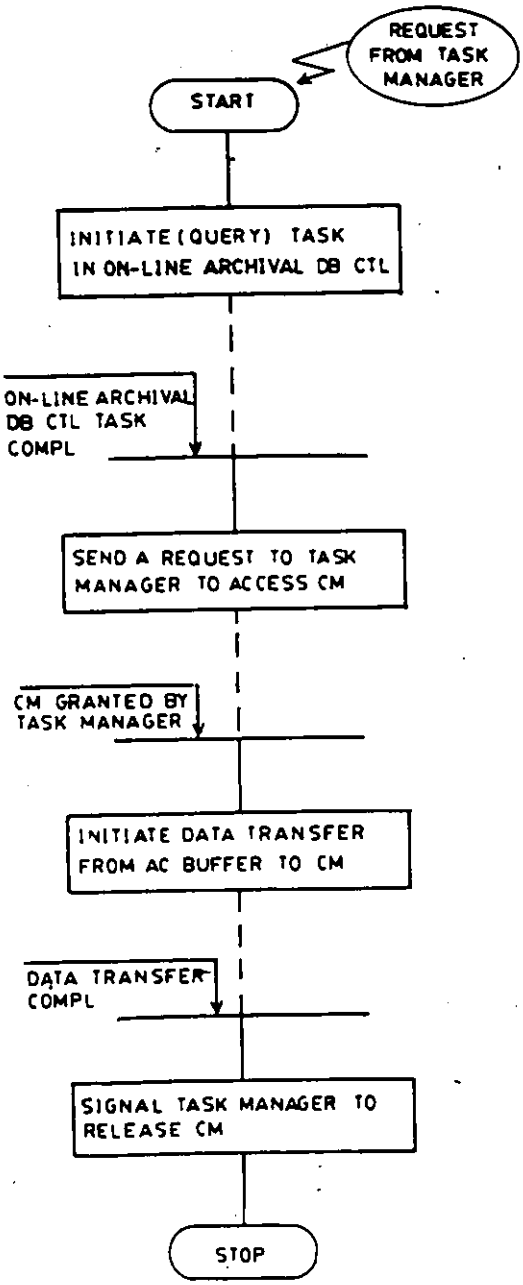

Figure 46: Additional information request

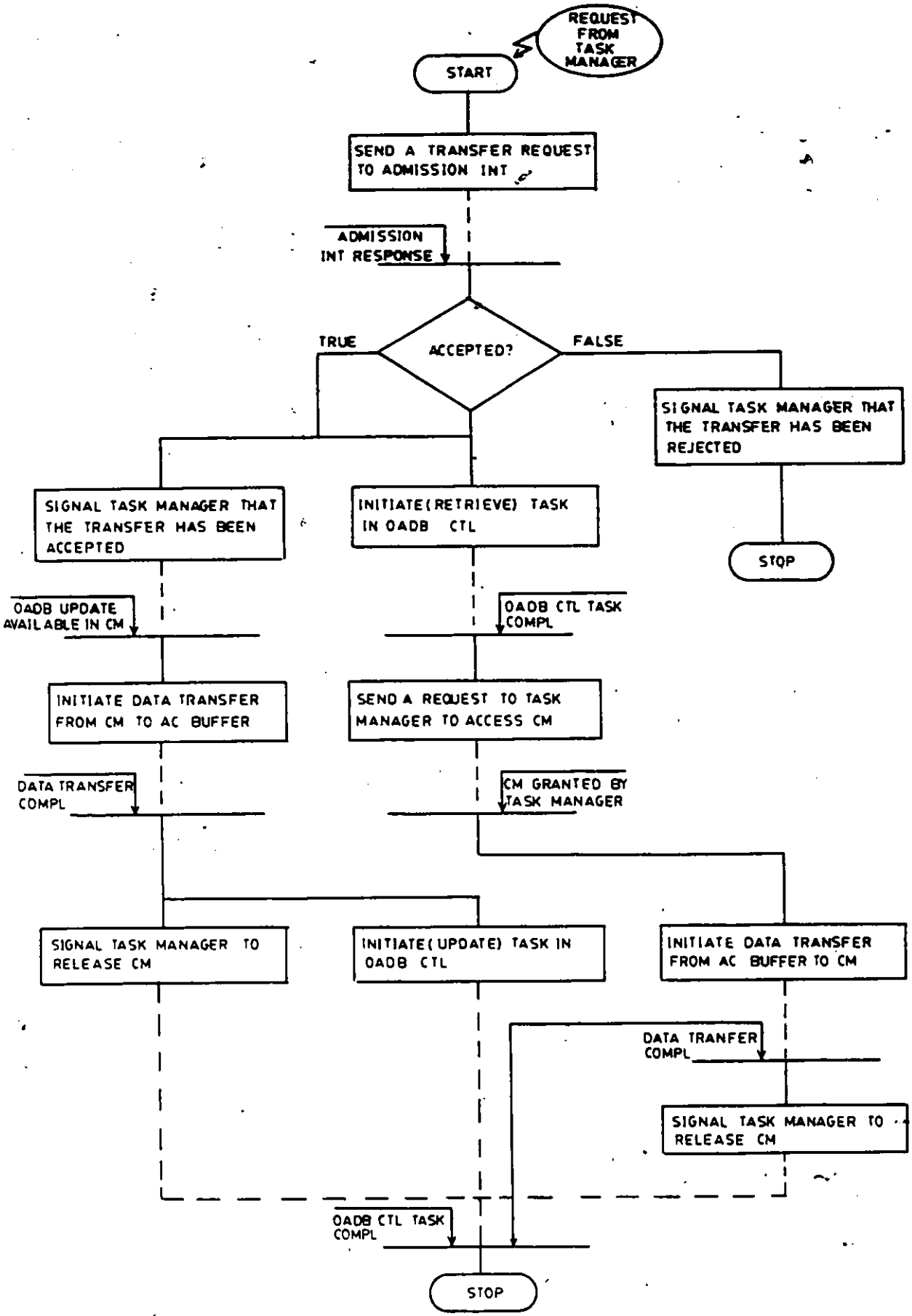

Figure 47: Transferring a patient

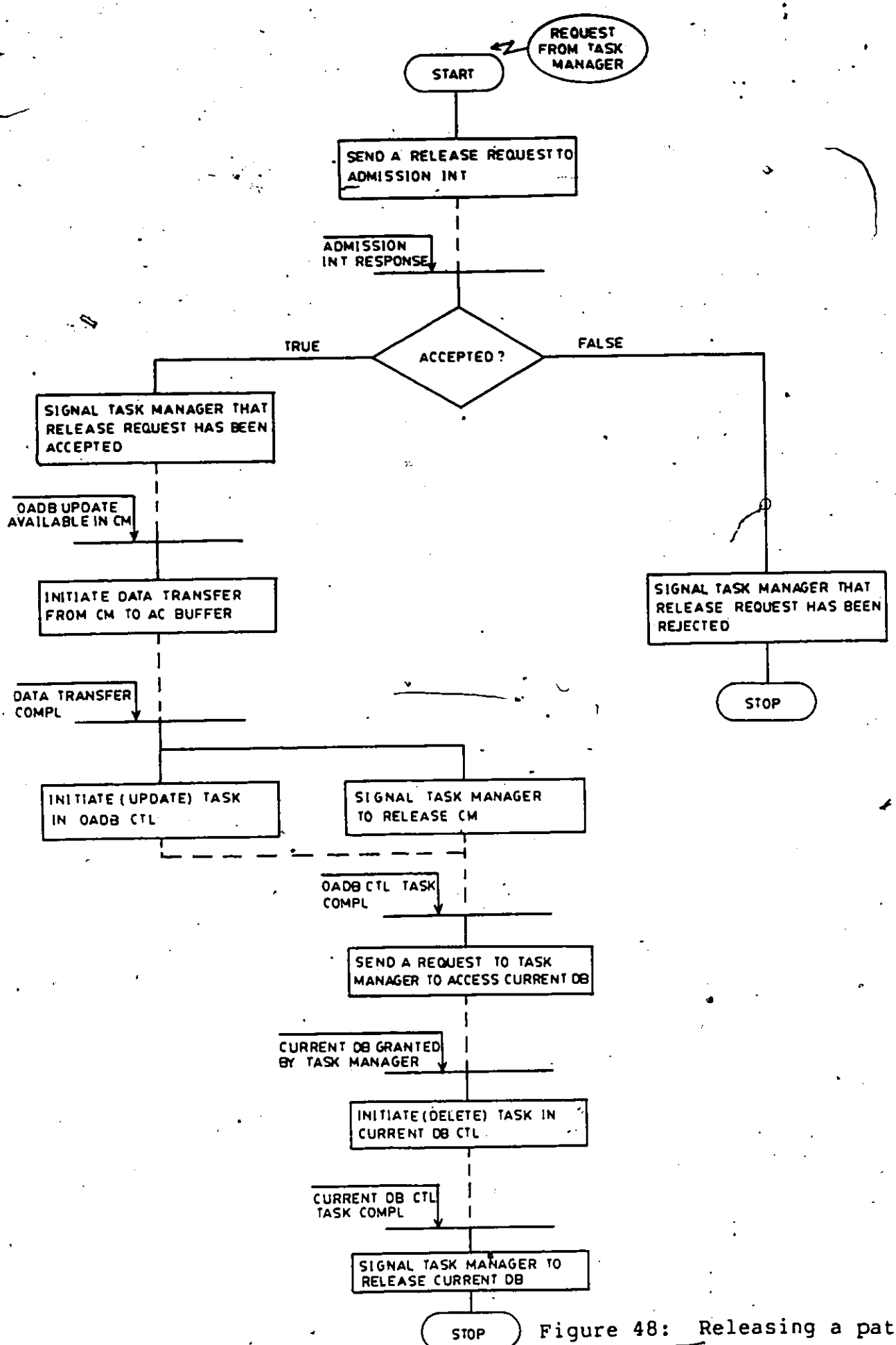

Figure 48: Releasing a patient

6.3 SIMULATION EXAMPLES

The first set of examples represent the system processes being executed separately. In the other set of examples, multiple process requests are initiated simultaneously.

6.3.1 Single request

Process: Admitting a former patient from emergency

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0		Retrieve request	Archivaldb
9		Insert emergency former patient	Currentdb
9		Transfer from ACB to common memory	CM and ACB
11	Transfer from com mon memory to LCB		CM and LCB
13	Broadcast emergen cy former record		Busctl
23	Update emergency former patient		Currentdb

Process terminated at clock: 29

Initial data:
patient profile: wwwwwwwwww
patient history: xxxxxxxxxx
patient admitting data: yyyyyyyyyy

Data broadcasted by busctl: wwwwwwwwwwxxxxxxxxxx

Data received by the currentdb:
patient profile: wwwwwwwwww
patient history: xxxxxxxxxx
patient admitting data: yyyyyyyyyy

Process: Admitting a former patient from admission

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0		Retrieve request	Archivaldb
0		Transmit admitting data	Admission
12		Insert admission patient	Currentdb
12		Transfer from ACB to common memory	CM and ACB
14	Transfer from com mon memory to LCB		CM and LCB
16	Broadcast new record		Busctl

Process terminated at clock: 22

Initial data:
patient profile: wwwwwwwww
patient history: zzzzzzzzzz
patient admitting data: kkkkkkkkk

Data broadcasted by busctl: wwwwwwwwwwzzzzzzzzzzkkkkkkkkkk

Data received by the currentdb:
patient profile: wwwwwwwww
patient history: zzzzzzzzzz
patient admitting data: kkkkkkkkk

Process: Admitting a new patient from admission

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0		Transmit new record	Admission
14		Insert admission patient	Currentdb
14		Transfer from ACB to common memory	CM and ACB
16	Transfer from com mon memory to LCB		CM and LCB
18	Broadcast new record		Busctl

Process terminated at clock: 24

Initial data:
patient profile: 1111111111
patient history: mmmmmmmmmm
patient admitting data: nnnnnnnnnn

Data broadcasted by busctl: 1111111111mmmmmmmmmmnnnnnnnnnn

Data received by the currentdb:
patient profile: 1111111111
patient history: mmmmmmmmmm
patient admitting data: nnnnnnnnnn

Process: Service request

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0	Transmit ward req to broad ser req		Busctl
5	Update service request		Currentdb

Process terminated at clock: 11

Process: Update request

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0	Transmit ward req to broad update		Busctl
9	Update service respond		Currentdb

Process terminated at clock: 14

Initial data:
service respond update: ddddddddd

Data received by currentdb:
service respond update: ddddddddd
Process: Update request

Process: Additional information request

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0		Query request	Archivaldb
4		Transfer from ACB to common memory	CM and ACB
5	Transfer from com mon memory to LCB		CM and LCB
6	Transmit addition al information		Busctl

Process terminated at clock: 11

Initial data:
additional history data: zzzzzzzzzz

Data transmit by busctl:
additional history data: zzzzzzzzzz

Process: Release a patient

CLOCK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0	Broadcast request for outstanding rq		Busctl
0	Check for outstanding req		Currentdb
5	Broadcast delete command		Busctl
13	Transfer from LCB to common memory		CM and LCB
14		Transfer from Com mon memory to ACB	CM and ACB
15		Update request	Archivaldb
21		Delete request	Currentdb

Process terminated at clock: 25

Initial data:
update archival data : ooooooooo

Data received by archivaldb:
update archival data: ooooooooo

6.3.2 Multiple requests

PROCESSES:

- Admitting new patient "Z" from admission
- Update patient "X"
- Update patient "Y"

CLK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0	UPDATE X -1 Transmit ward req to broad update (9)		Busctl
0		ADMT-AD Z -1 Transmit new record (14)	Admission
9	UPDATE X -2 Update service respond (5)		Currentdb
14	UPDATE Y -1 Transmit ward req to broad update (9)		Busctl
14		ADMT-AD Z -2 Insert admission patient (6) -2 Transfer from ACB to common memory (2)	Currentdb CM and ACB
16	ADMT-AD Z -3 Transfer from com mon memory to LCB (2)		CM and LCB
23	ADMT-AD Z -4 Broadcast new record (6)		Busctl
23	UPDATE Y -2 Update service respond (5)		Currentdb

```
Process UPDATE "X" terminated at clock:          14


Initial data:
  Service respond update: ppppppppppp


Data received by currendb:


  Service respond update: ppppppppp

Process UPDATE "Y" terminated at clock:          28

Initial data:
  service respond update: ddddddddd


Data received by currentdb:
  service respond update: ddddddddd

Process ADMT-AD "Z" terminated at clock:         29


Initial data:
  patient profile: 1111111111
  patient history: mmmmmmmmmm
  patient admitting data: nnnnnnnnnn

Data broadcasted by busctl: 1111111111mmmmmmmmmmnnnnnnnnnn

Data received by the currentdb:
  patient profile: 1111111111
  patient history: mmmmmmmmmm
  patient admitting data: nnnnnnnnnn
```

PROCESSES:

- Admitting new patient "X" from admission
- Release patient "Y"

CLK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0	RELEASE Y -1 Broadcast request for outstanding rq(5) -1 Check for outstanding req (3)		Busctl Currentdb
0		ADMT-AD X -1 Transmit new record (14)	Admission
5	RELEASE Y -2 Broadcast delete command (8)		Busctl
13	RELEASE Y -3 Transfer from LCB to common memory (1)		CM and LCB
14		RELEASE Y -4 Transfer from Com mon memory to ACB (1)	CM and ACB
14		ADMT-AD X -2 Insert admission patient (6)	Currentdb
15		ADMT-AD X -3 Transfer from ACB to common memory (2)	CM and ACB
15		RELEASE Y -5 Update request (6)	Archivaldb
17	ADMT-AD X -4 Transfer from com mon memory to LCB (2)		CM and LCB
19	ADMT-AD X -5 Broadcast new record (6)		Busctl
21		RELEASE Y -6 Delete request (4)	Currentdb

```
Process ADMT-AD "X" terminated at clock:          25

Initial data:
  patient profile: lllllllll
  patient history: mmmmmmmmmm
  patient admitting data: nnnnnnnnn

Data broadcasted by busctl: lllllllllmmmmmmmmmmnnnnnnnnn

Data received by the currentdb:
  patient profile: lllllllll
  patient history: mmmmmmmmmm
  patient admitting data: nnnnnnnnn

Process RELEASE "Y" terminated at clock           25


Initial data:
  update archival data : ooooooooo


Data received by archivaldb:
  update archival data: ooooooooo
```

PROCESSES:

- Admitting former patient "Z" from emergency
- Update patient "X"
- Additionnal information (patient "A")
- Update patient "Y"

CLK	ACTIVITIES INITIATED		RESSOURCE
	TASK DRIVER 1	TASK DRIVER 2	
0	UPDATE X -1 Transmit ward req to broad update (9)		Busctl
0		ADMT-EM Z -1 Retrieve request (9)	Archivaldb
9	UPDATE X -2 Update service respond (5)		Currentdb
9		ADMT-EM Z -2 Transfer from ACB to common memory (2)	CM and ACB
11	ADMT-EM Z -3 Transfer from com mon memory to LCB (2)		CM and LCB
14	UPDATE Y -1 Transmit ward req to broad update (9)		Busctl
14		ADMT-EM Z -4 Insert emergency former patient (4)	Currentdb
18		ADDINF A -1 Query request (4)	Archivaldb
22		ADDINF A -2 Transfer from ACB to common memory (1)	CM and ACB
23	UPDATE Y -2 Update service respond (5)		Currentdb
28	ADMT-EM Z -5 Broadcast emergen cy former record (10)		Busctl

38	ADDINF A -3 Transfer from com mon memory to LCB (1)		CM and LCB
38	ADMT-EM Z -6 Update emergency former patient (6)		Currentdb
39	ADDINF A -4 Transmit addition al information (5)		Busctl

Process UPDATE "X" terminated at clock: 14

Initial data:
Service respond update: ppppppppp

Data received by currendb:
Service respond update: ppppppppp

Process UPDATE "Y" terminated at clock: 28

Initial data:
service respond update: ddddddddd

Data received by currentdb:
service respond update: ddddddddd

Process ADMT-EM "Z" terminated at clock: 44

Initial data:
patient profile: wwwwwwwww
patient history: xxxxxxxxx
patient admitting data: yyyyyyyyy

Data broadcasted by busctl: wwwwwwwwwxxxxxxxxx

Data received by the currentdb:
patient profile: wwwwwwwww
patient history: xxxxxxxxx
patient admitting data: yyyyyyyyy

Process ADDINF "A" terminated at clock: 44

Initial data:
additional history data: zzzzzzzzz

Data transmitted by busctl:
additional history data: zzzzzzzzz

Chapter VII

CONCLUSION

Present day research as indicated in chapter two is mainly focused on the design of very large general-purpose data base systems. A major research area is the development of the various data base processors. An alternate research area considers the integration of existing data bases into a simple unified data base system. This research area is motivated by the following facts: (1) the recognition of the benefits of distributed systems, and (2) the large investments in existing data base systems.

With the declining cost of processing hardware and storage devices, computerized information systems are becoming cost effective in a variety of applications. Many of these applications require the design of simple user oriented cost effective data base system. In the case of small number of users, this can implemented by using a general-purpose minicomputer system. This approach is not cost effective in the case of more complex multi-user system as it requires a large amount of software overhead.

This thesis introduces a layered data base architecture applicable to multi-user information systems where user ac-

cess requirements are restricted to a small portion of the total data base. In such cases by duplicating part of the data base, one can isolate the various users and provide a highly reliable user oriented distributed data base system. To show the applicability and feasibility of such layered data base, the implementation of a patient medical record system was presented. To provided the required background, a survey of existing HIS was done.

In chapter four, a top down design of the proposed system was undertaken. In this system, the critical element is the data base manager which has to control the information flow between the different layers. Using the oligarchical approach, in chapter five, the implementation of the data base manager as a multi-processor system was presented. The operation of the data base manager was simulated in order to show the feasibility of the design.

In this thesis, we demonstrate that it is possible to implement a cost effective special-purpose systems in applications where general-purpose system have failed to work. The next step is to undertake a detailed simulation to check for possible bottlenecks, and its overall integration into an HIS.

This layered data base is applicable to a number of other special-purpose data base systems. It is relatively straight forward to map the requirements of such applications into a two or multilayer data base system.

Appendix A

SIMULATION LISTING

A simulation in Pascal was done to show the feasibility of the data base manager's design. The main section of the program is first described. Then a list of the variables used is given. Along with this, the relevant data structures used are described. Finally, each procedure called is described separately. Note that the program section for printing the results is not included in this program listing.

```
PROGRAM simulation (input,output);
```

This program simulates the concurrent operation of task drivers 1 and 2. This is achieved by calling repeatedly the two procedures that simulate the operations of task drivers 1 and 2. The task manager is implemented as a passive resource; i.e. accesses to CDB and the common memory are controlled by a settable flag. The bus controller, the CDB controller, The OADB controller and admission interface are also implemented as passive resources.

```
BEGIN
  initialize;
```

This procedure initializes variables such as counters, flags, linked lists.

```
  iniptl;
  inipt2;
```

These two procedures initialize respectively the task static list of each task driver.

```
  inireq;
```

This procedure initiates the system requests (e.g. update request, release request). This procedure is not included in this listing.

```
  WHILE bo DO
    BEGIN
      taskdr1;
      taskdr2;
      taskdr1;
```

```
    counter := counter + 1;
    fini;
END; END.
```

The COUNTER represents the clock of the system. Procedures TASKDR1 and TASKDR2 correspond respectively to task drivers 1 and 2. During each clock cycle, procedure TASKDR1 and TASKDR2 are called. Procedure TASKDR1 is recalled after procedure TASKDR2 to check if a request was send by procedure TASKDR1. The procedure FINI is not included in this program listing. The program terminates when all tasks are completed. This condition is evaluated by the procedure FINI.

The following data structures are required for this simulation. These data structures are listed according to type of object they represent.

1) System buffers

LCBUFFER,ACBUFFER,COMMONM: These variables represent respecttively the lc buffer, the ac buffer and the common memory. Each one is implemented as an array of type character.

2) Data bases

CDB: The current data base is implemented as an array of records. Each field consists of an array of type character representing a section of a CMR.

ADB: The on-line archival data base is implemented in the same fashion as the current data base.

3) Control buses

All communications between the elements of the data base manager subsystem are done through variables.

CTL1,CTL2: These are the control buses which interconnect the task drivers to their controllers. These buses are implemented as linked lists.

CTLTN: This control bus interconnects the task manager to the two task drivers and is implemented as a record.

4) Lists of task drivers

TD1PTLST,TD2PTLST: The task static lists are implemented as an array of record. The record format is defined as follows:

```
    (ressource): The name of the resource on which the
                 task resides.
    (TIMEOUT): A count representing the execution time.
    (BUFFER): Buffer segment affected during the execution
              of the task.

   READYQ1,READYQ2: The ready lists (dynamic) list
   the task waiting to be executed. Those lists are
   implemented as an array of record. The record fields
   are self explanatory.

   ACT1LST,ACT2LST: The execution lists contain all
   currently executing tasks. Those lists are implemted
   as linked lists.

TYPE

  dix = PACKED ARRAY(.1..10.) OF char;
  sept = PACKED ARRAY(.1..7.) OF char;
  dixhuit = PACKED ARRAY(.1..18.) OF char;
  resstype = (rcurrent,rarchival,rbusctl,radmission,dma1,
              dma2);
  processtype = (adadm,ademer,serreq,updreq,addinf,transf,
                 release);
  tasktype = (inadmp,inemerfp,inemernp,upemernp,uptransf,
              upemerfp,upserreq,upserres,upward,queryreq,
              checkoutreq,deletereq,retrieve,query,update,
              bnwr,btrr,breour,bdec,bemr,temncr,term,twasr,
              twaup,taddinf,twatr,arr,art,atradm,atrnr,
              dmalcom,dma2com,dmalcb,dmaacb) ;

  link = @object;

  object =
    RECORD
      next : link;
      data : integer;
    END;


  bus =
    RECORD
      curtask : tasktype;
      arctask : tasktype;
      bustask : tasktype;
      admtask : tasktype;
      dmatask : tasktype;
      base : link;
    END;

  bus1 =
    RECORD
      request : (cdbr,comr,cdbd,comd);
      resp : integer;
```

```
    address : integer;
  END;

current =
  RECORD
    patprofile : dix;
    pasthistory : dix;
    admdata : dix;
    wardinf : dix;
    service1 : dix;
    service2 : dix;
  END;

archival =
  RECORD
    patprofile : dix;
    pbl1 : dix;
    pbl2 : dix;
    pbl3 : dix;
  END;

ready =
  RECORD
    ind : integer;
    process : processtype;
    task : tasktype;
  END;

ptlst =
  RECORD
    ressource : resstype;
    timeout : integer;
    buffer : (wlcfs,wlcbs,wacfs,wacbs,rlcfs,rlcbs,racfs,
              racbs,wcom,nobuffer);
  END;

point = @activetype;

activetype =
  RECORD
    ind : integer;
    process : processtype;
    task : tasktype;
    timeout : integer;
    prochain : point;
  END;

cn =
  RECORD
    ck : integer;
    no : integer;
    pr : integer;
  END;
```

```
VAR

  t,i : integer;
  bo : boolean;
  cdb : ARRAY(.1..5.) OF current;
  adb : ARRAY(.1..5.) OF archival;
  lcbuffer,acbuffer,commonm : PACKED ARRAY(.1..100.)
                              OF char;
  ctl1,ctl2 : bus;
  ctltm : busl;
  p : link;
  test2,test3,result4 : text;
  l,act2lst,act1lst : point;
  readyq1,readyq2 : ARRAY(.0..10.) OF ready;
  back1,back2,front1,front2 : integer;
  td1ptlst,td2ptlst : ARRAY(.processtype,tasktype.) OF
                      ptlst;
  ress1sem,ress2sem : ARRAY(.resstype.) OF integer;
  acbf,acff,cdbf,combf,comff,lcff,lcbf : integer;
  counter : integer;
  finish : ARRAY(.processtype,1..5.) OF integer;
  flag1,flag2,admf : integer;
  work : ARRAY(.processtype,1..5.) OF integer;
  indent : ARRAY(.processtype,1..5.) OF integer;
  num : ARRAY(.processtype,1..5.) OF cn;

PROCEDURE initialize;
```

This procedure initializes the following variables: (1) all flags, (2) COUNTER variable, (3) CDB and ADB variables, (4) link lists; i.e. CTL1.BASE, CTL2.BASE, ACT1LST, ACT2LST, (4) pointers of the READYQ1 and READYQ2 queues.

```
  VAR
    i,x : integer;

  PROCEDURE listini (VAR source : link);

    BEGIN
      new(p);
      p@.next := NIL;
      p@.data := 0;
      source := p
    END;

  PROCEDURE actini (VAR source : point);

    BEGIN
      new(l);
      l@.prochain := NIL;
      source := l;
    END;

  BEGIN
```

```
    listini(ctl1.base);
    listini(ctl2.base);
    actini(act1lst);
    actini(act2lst);
    bo := true;
    front1 := 1;
    back1 := 1;
    front2 := 1;
    back2 := 1;
    ress1sem(.rcurrent.) := 0;
    ress1sem(.rbusctl.) := 0;
    ress1sem(.dma1.) := 0;
    ress2sem(.rcurrent.) := 0;
    ress2sem(.rarchival.) := 0;
    ress2sem(.radmission.) := 0;
    ress2sem(.dma2.) := 0;
    lcff := 0;
    lcbf := 0;
    cdbf := 0;
    comff := 0;
    acff := 0;
    acbf := 0;
    combf := 0;
    counter := 0;
    flag1 := 0;
    flag2 := 0;
    admf := 0;

    FOR x := 1 TO 5 DO
      WITH cdb(.x.) DO
        FOR i := 1 TO 10 DO
          BEGIN
            patprofile(.i.) := ' ';
            pasthistory(.i.) := ' ';
            admdata(.i.) := ' ';
            wardinf(.i.) := ' ';
            service1(.i.) := ' ';
            service2(.i.) := ' ';
          END;
    FOR x := 1 to 5 DO
      WITH adb(.x.) DO
        FOR i := 1 to 10 DO
          BEGIN
            patprofile(.i.) := 'w';
            pbl1(.i.) := 'x';
            pbl2(.i.) := 'y';
            pbl3(.i.) := 'z';
          END;
  END;

PROCEDURE iniptl;
```

This procedure initializes the task static list of task driver 1.

```
BEGIN
  WITH tdlptlst(.adadm,dmalcb.) DO
    BEGIN
      ressource := dmal;
      buffer := wlcbs;
      timeout := 2;
    END;
  WITH tdlptlst(.adadm,bnwr.) DO
    BEGIN
      ressource := rbusctl;
      buffer := rlcbs;
      timeout := 6;
    END;
  WITH tdlptlst(.serreq,twasr.) DO
    BEGIN
      ressource := rbusctl;
      buffer := wlcfs;
      timeout := 5;
    END;
  WITH tdlptlst(.serreq,upserreq.) DO
    BEGIN
      ressource := rcurrent;
      buffer := rlcfs;
      timeout := 6;
    END;
  WITH tdlptlst(.updreq,twaup.) DO
    BEGIN
      ressource := rbusctl;
      buffer := wlcfs;
      timeout := 9;
    END;
  WITH tdlptlst(.updreq,upserres.) DO
    BEGIN
      ressource := rcurrent;
      buffer := rlcfs;
      timeout := 5;
    END;
  WITH tdlptlst(.updreq,upward.) DO
    BEGIN
      ressource := rcurrent;
      buffer := rlcfs;
      timeout := 5;
    END;
  WITH tdlptlst(.addinf,queryreq.) DO
    BEGIN
      ressource := rcurrent;
      buffer := wlcbs;
      timeout := 10;
    END;
  WITH tdlptlst(.addinf,taddinf.) DO
```

```
  BEGIN
    ressource := rbusctl;
    buffer := rlcbs;
    timeout := 5;
  END;
WITH tdlptlst(.addinf,dmalcb.) DO
  BEGIN
    ressource := dmal;
    buffer := wlcbs;
    timeout := 1;
  END;
WITH tdlptlst(.ademer,dmalcb.) DO
  BEGIN
    ressource := dmal;
    buffer := wlcbs;
    timeout := 2;
  END;
WITH tdlptlst(.ademer,bemr.) DO
  BEGIN
    ressource := rbusctl;
    buffer := wlcfs;
    timeout := 10;
  END;
WITH tdlptlst(.ademer,upemerfp.) DO
  BEGIN
    ressource := rcurrent;
    buffer := rlcfs;
    timeout := 6;
  END;
WITH tdlptlst(.release,breour.) DO
  BEGIN
    ressource := rbusctl;
    buffer := nobuffer;
    timeout := 5;
  END;
WITH tdlptlst(.release,checkoutreq.) DO
  BEGIN
    ressource := rcurrent;
    buffer := nobuffer;
    timeout := 3;
  END;
WITH tdlptlst(.release,bdec.) DO
  BEGIN
    ressource := rbusctl;
    buffer := wlcfs;
    timeout := 8;
  END;
WITH tdlptlst(.release,dmalcom.) DO
  BEGIN
    ressource := dmal;
    buffer := wcom;
    timeout := 1;
  END;
```

```
  END;

PROCEDURE inipt2;

   This procedure initializes the task static list of task
driver 2.


  BEGIN
    WITH td2ptlst(.adadm,retrieve.) DO
      BEGIN
        ressource := rarchival;
        buffer := wacbs;
        timeout := 9;
      END;
    WITH td2ptlst(.adadm,atradm.) DO
      BEGIN
        ressource := radmission;
        buffer := wacbs;
        timeout := 12;
      END;
    WITH td2ptlst(.adadm,atrnr.) DO
      BEGIN
        ressource := radmission;
        buffer := wacbs;
        timeout := 14;
      END;
    WITH td2ptlst(.adadm,inadmp.) DO
      BEGIN
        ressource := rcurrent;
        buffer := racbs;
        timeout := 6;
      END;
    WITH td2ptlst(.adadm,dma2com.) DO
      BEGIN
        ressource := dma2;
        buffer := wcom;
        timeout := 2;
      END;
    WITH td2ptlst(.ademer,retrieve.) DO
      BEGIN
        ressource := rarchival;
        buffer := wacbs;
        timeout := 9;
      END;
    WITH td2ptlst(.ademer,inemerfp.) DO
      BEGIN
        ressource := rcurrent;
        buffer := racbs;
        timeout := 4;
      END;
    WITH td2ptlst(.ademer,dma2com.) DO
      BEGIN
        ressource := dma2;
```

```
        buffer := wcom;
        timeout := 2;
      END;
    WITH td2ptlst(.addinf,query.) DO
      BEGIN
        ressource := rarchival;
        buffer := wacbs;
        timeout := 4;
      END;
    WITH td2ptlst(.addinf,dma2coms) DO
      BEGIN
        ressource := dma2;
        buffer := wcom;
        timeout := 1;
      END;
    WITH td2ptlst(.release,dmaacb.) DO
      BEGIN
        ressource := dma2;
        buffer := wacfs;
        timeout := 1;
      END;
    WITH td2ptlst(.release,update.) DO
      BEGIN
        ressource := rarchival;
        buffer := racfs;
        timeout := 6;
      END;
    WITH td2ptlst(.release,deletereq.) DO
      BEGIN
        ressource := rcurrent;
        buffer := nobuffer;
        timeout := 4;
      END;
  END;

FUNCTION removectl (source : link) : integer;
```

This function removes an element from the linked list of pointer type LINK. This function is used for linked lists CTL1.BASE and CTL2.BASE.

```
  BEGIN
    p := source@.next;
    removectl := p@.data;
    IF p@.next = NIL THEN
      source@.next := NIL
    ELSE
      source@.next := p@.next;
    dispose(p);
  END;

PROCEDURE insertctl (source : link; a : integer);
```

This procedure inserts an element into a linked list of pointer type LINK.

```
VAR
  trans : link;

BEGIN
  new(p);
  p@.data := a;
  p@.next := NIL;
  trans := source;
  WHILE trans@.next <> NIL DO
    trans := trans@.next;
  trans@.next := p
END;
```

```
PROCEDURE insertact (source : point; w : integer;
                     x : processtype;
                     y : tasktype; z : integer);
```

This procedure inserts an element into a linked list of pointer type POINT. This procedure is used by linked lists ACT1LST AND ACT2LST.

```
VAR
  trans : point;

BEGIN
  new(l);
  l@.ind := w;
  l@.process := x;
  l@.task := y;
  l@.timeout := z;
  l@.prochain := NIL;
  trans := source;
  WHILE trans@.prochain <> NIL DO
    trans := trans@.prochain;
  trans@.prochain := l;
END;
```

```
PROCEDURE remact (source1,source2 : point);
```

This procedure removes an element from a linked list of pointer type POINT.

```
BEGIN
  WHILE source1@.prochain <> source2 DO
    source1 := source1@.prochain;
  source1@.prochain := source2@.prochain;
END;
```

```
PROCEDURE insertql (w : integer; x : processtype;
                    y : tasktype);
```

This procedure is called to insert an element into the the ready list READYQ1.

```
BEGIN
  readyq1(.back1.).ind := w;
  readyq1(.back1.).process := x;
  readyq1(.back1.).task := y;
  back1 := (back1 + 1) MOD 10;
END;

PROCEDURE insertq2 (w : integer; x : processtype;
                    y : tasktype);
```

This procedure is called to insert to insert an element into the ready list READYQ2.

```
BEGIN
  readyq2(.back2.).ind := w;
  readyq2(.back2.).process := x;
  readyq2(.back2.).task := y;
  back2 := (back2 + 1) MOD 10;
END;


PROCEDURE busctl;
```

This procedure represents the bus controller. The bus controller is simulated as a passive resource. Its function in this program, is to initialize the lc buffer for the bus controller tasks which send data to the data base manager.

```
VAR
  loc,i : integer;

BEGIN
  loc := removectl(ctl1.base);
  CASE ctl1.bustask OF
    bnwr: ;
    btrr: ;
    breour: ;
    bemr:
      FOR i := 1 TO 10 DO
        lcbuffer(.i.) := 'y';
    bdec:
      FOR i := 1 TO 10 DO
        lcbuffer(.i+(loc-1).) := 'o';
    temncr:
      FOR i := loc TO loc+9 DO
        BEGIN
          lcbuffer(.i.) := 'a';
          lcbuffer(.i.) := 'b';
```

```
        END;
      term: ;
      twasr:;
      twaup:
        FOR i := loc TO loc+9 DO
          lcbuffer(.i.) := 'd';
      taddinf: ;
      twatr:
        FOR i := loc TO loc + 9 DO
          BEGIN
            lcbuffer(.i.) := 'e';
            lcbuffer(.i+10.) := 'f';
            lcbuffer(.i+20.) := 'g';
            lcbuffer(.i+30.) := 'h';
            lcbuffer(.i+40.) := 'i';
            lcbuffer(.i+50.) := 'j';
          END;
    END;
  END;

PROCEDURE curdbctl (cf : integer);
```

This procedure represents the CDB controller. In this program, it is implemented as a passive resource. This procedure executed all accesses to the CDB data structure for all CDB controller tasks called.

```
  VAR
    x,patno : integer;
    c1,c2 : char;

  PROCEDURE tdriver1;
    VAR
      x,i : integer;

    BEGIN
      CASE ctl1.curtask OF
        inemernp:
          BEGIN
            i := removectl(ctl1.base);
            FOR x := 1 to 10 DO
              BEGIN
                cdb(.patno.).admdata(.x.) := lcbuffer(.i.);
                i := i + 1
              END;
          END;
        upemerfp:
          BEGIN
            i := removectl(ctl1.base);
            FOR x := 1 TO 10 DO
              BEGIN
                cdb(.patno.).admdata(.x.) := lcbuffer(.i.);
                i := i + 1
              END;
```

```
  END;
upemernp:
  BEGIN
    i := removectl(ctl1.base);
    FOR x := 1 TO 10 DO
      BEGIN
        cdb(.patno.).patprofile(.x.) :=
          lcbuffer(.i.);
        cdb(.patno.).pasthistory(.x.) :=
          lcbuffer(.i.);
        i := i + 1
      END;
  END;
upserreq:
  CASE removectl(ctl1.base) OF
    1:
      cdb(.patno.).service1(.10.) := 'o';
    2:
      cdb(.patno.).service2(.10.) := 'o';
  END;
checkoutreq:
  BEGIN
    c1 := cdb(.patno.).service1(.10.);
    c2 := cdb(.patno.).service2(.10.);
    IF (c1 = 'o') OR (c2 = 'o') THEN
      insertctl(ctl1.base,1)
    ELSE
      insertctl(ctl1.base,0);
  END;
upserres:
  BEGIN
    i := removectl(ctl1.base);
    CASE removectl(ctl1.base) OF
      1:
        FOR x := 1 TO 9 DO
          BEGIN
            cdb(.patno.).service1(.x.) :=
              lcbuffer(.i.);
            i := i + 1;
          END;
      2:
        FOR x := 1 To 9 Do
          BEGIN
            cdb(.patno.).service2(.x.) :=
              lcbuffer(.i.);
            i := i + 1;
          END;
    END;
  END;
upward:
  BEGIN
    i := removectl(ctl1.base);
    FOR x := 1 TO 10 DO
      BEGIN
```

```
                cdb(.patno.).wardinf(.x.) := lcbuffer(.i.);
                i := i + 1;
              END;
          END;
        queryreq:
          BEGIN
            i := removectl(ctl1.base);
            FOR x := 1 TO 10 DO
              BEGIN
                lcbuffer(.i.) := 'q';
                i := i + 1;
              END;
          END;
        uptransf:
          BEGIN
            i := removectl(ctl1.base);
            FOR x := 1 TO 10 DO
              BEGIN
                WITH cdb(.patno.) DO
                  BEGIN
                    patprofile(.x.) := lcbuffer(.i.);
                    pasthistory(.x.) := lcbuffer(.i.);
                    admdata(.x.) := lcbuffer(.i.);
                    wardinf(.x.) := lcbuffer(.i.);
                    service1(.x.) := lcbuffer(.i.);
                    service2(.x.) := lcbuffer(.i.);
                  END;
                i := i + 1;
              END;
            END;
      END
    END;

  PROCEDURE tdriver2;
    VAR
      i,x : integer;

    BEGIN
      i := removectl(ctl2.base);
      CASE ctl2.curtask OF
        inadmp:
          FOR x := 1 TO 10 DO
            BEGIN
              WITH cdb(.patno.) DO
                BEGIN
                  patprofile(.x.) := acbuffer(.i.);
                  pasthistory(.x.) := acbuffer(.i+10.);
                  admdata(.x.) := acbuffer(.i+20.);
                END;
              i := i + 1;
            END;
        deletereq:;
        inemerfp:
          FOR x := 1 TO 10 DO
```

```
            BEGIN
              WITH cdb(.patno.) DO
                BEGIN
                  patprofile(.x.) := acbuffer(.i.);
                  pasthistory(.x.) := acbuffer(.i+10.);
                END;
              i := i + 1;
            END;
      END;
    END;

  BEGIN
    IF cf = 0 THEN
      BEGIN
        patno := removectl(ctl1.base);
        tdriver1
      END
    ELSE
      BEGIN
        patno := removectl(ctl2.base);
        tdriver2
      END;
  END;

PROCEDURE arcdbctl;
```

This procedure represents the OADB controller as a passive resource. This procedure performs all the required accesses to the ADB data structure to process the OADB controller tasks called.

```
    VAR
      i,patno,loc,pbl : integer;
      a : PACKED ARRAY(.1..10.) OF char;

  PROCEDURE search (x,y : integer);

      BEGIN
        WITH adb(.x.) DO
          CASE y OF
            1: a := pbl1;
            2: a := pbl2;
            3: a := pbl3;
          END;
      END;

  BEGIN
    patno := removectl(ctl2.base);
    pbl := removectl(ctl2.base);
    loc := removectl(ctl2.base);
    CASE ctl2.arctask OF
      retrieve:
        BEGIN
          search(patno,pbl);
```

```
          FOR i := 1 TO 10 DO
            BEGIN
              acbuffer(.loc+10.) := a(.i.);
              acbuffer(.loc.) :=
                adb(.patno.).patprofile(.i.);
              loc := loc + 1;
            END;
        END;
      query:
        BEGIN
          search(patno,pbl);
          FOR i := 1 TO 10 DO
            BEGIN
              acbuffer(.loc.) := a(.i.);
              loc := loc + 1;
            END;
        END;
      update:
        BEGIN
          WITH ADB(.patno.) DO
            CASE pbl OF
              1: pbl1 := a;
              2: pbl2 := a;
              3: pbl3 := a;
          END;
          FOR i := 1 TO 10 DO
            BEGIN
              a(.i.) := acbuffer(.loc.);
              loc := loc + 1;
            END;
        END;
    END;
  END;

PROCEDURE admission;

    This procedure corresponds to the admission interface.
The admission interface is implemented as a passive re-
source. This procedure initializes the ac buffer for the ad-
mission interface tasks called.

  VAR
    i,loc : integer;

  BEGIN
    loc := removectl(ctl2.base);
    CASE ctl2.admtask OF
      arr: ;
      art: ;
      atradm:
        FOR i := loc TO loc+9 DO
          BEGIN
            acbuffer(.loc.) := 'k';
            loc := loc + 1;
```

```
        END;
      atrnr:
        FOR i := 1 TO  10  DO
          BEGIN
            acbuffer(.(loc-1)+i.) := '1';
            acbuffer(.(loc+9)+i.) := 'm';
            acbuffer(.(loc+19)+i.) := 'n';
          END;
    END;
  END;

PROCEDURE pdma1;

   This procedure transfers data to and from the COMMONM and
the LCBUFFER.

 VAR
  loc,i,size : integer;

  BEGIN
    size := removectl(ctl1.base);
    loc := removectl(ctl1.base);
    CASE ctl1.dmatask OF
      dmalcom:
        FOR i := 1 TO size Do
          commonm(.i + (loc-1).) := lcbuffer(.i.);
      dmalcb:
        FOR i := 1 TO size DO
          lcbuffer(.i+49.) := commonm(.i+(loc-1).);
    END;
  END;

PROCEDURE pdma2;

   This procedure transfers data to and from the COMMONM and
the ACBUFFER.

  VAR
    loc,i,size : integer;

  BEGIN
    size := removectl(ctl2.base);
    loc := removectl(ctl2.base);
    CASE ctl2.dmatask OF
      dma2com:
        FOR i := 1 TO size DO
          commonm(.i+(loc-1).) := acbuffer(.i+49.);
      dmaacb:
        FOR i := 1 TO size DO
          acbuffer(.i.) := commonm(.i+(loc-1).);
    END;
  END;

PROCEDURE taskmanager;
```

The task manager is implemented as a passive resource. This procedure controls all accesses to the current data base and the common memory through semaphores (flags).

```
BEGIN
  CASE ctltm.request OF
    cdbr :
      BEGIN
      IF cdbf = 0 THEN
        BEGIN
          cdbf := 1;
          ctltm.resp := 0;
        END
      ELSE
        ctltm.resp := 1;
      END;
    cdbd:
      cdbf := 0;
    comr:
      IF comff = 0 THEN
        BEGIN
          comff := 1;
          ctltm.resp := 0;
          ctltm.address := 1;
        END
      ELSE
        IF combf = 0 THEN
          BEGIN
            combf := 1;
            ctltm.resp := 0;
            ctltm.address := 50;
          END
        ELSE
          ctltm.resp := 1;
    comd:
      BEGIN
        IF ctltm.address = 50 THEN
          combf := 0
        ELSE
          comff := 0;
      END;
  END;
END;
```

PROCEDURE taskdrl;

This procedure simulates the operation of task driver 1. The main section of this procedure is divided into two parts. The first part calls a procedure (ACTIVATE) to check if any tasks in the ready list can be executed. The other part provides access to the look-up table. This table is accessed for the following purposes: (1) to call the passive resource procedure (e.g. BUSCTL,ADMISSION) to execute a task and (2) to determine the next task(s) ready to be executed.

```
BEGIN
  activate;
```

For each task in the ready list, this procedure checks if the resources that it requires are available. If the resources are available, then the ready task is inserted into the execution list (ACTlLST). Each entry of the execution list contains the following information: (1) request identifier, (2) name of the process, (3) name of the task, and (4) a count indicating the end of the execution of the task.

```
  l := actllst@.prochain;
  WHILE l <> nil DO
    BEGIN
      IF l@.timeout = counter THEN
        BEGIN
          remact(actllst,l);
          CASE l@.process OF
            serreq : prserreq;
            updreq : prupdreq;
            addinf : praddinf;
            adadm : praadadm;
            ademer : prademer;
            release : prrelease;
          END;
        END;
    l := l@.prochain;
  END; END;
```

For each task in the execution list, the above program section checks if the task is completed. This is achieved by verifying if the clock (COUNTER) is equal to the count field (TIMEOUT) which is stored in the ready list. If the above condition is satisfy, the look-up table is accessed to execute the task and to determine the next task(s) ready to be executed. In this program, the look-up table is subdivided into a number of process look-tables. Each process look-table is implemented as a separate procedure.

```
  PROCEDURE activate;
```

For each task in the ready list (READYQ1), this procedure checks if the task is ready to be executed.

```
    VAR
      a,i,flag,z : integer;
      x : processtype;
      y : tasktype;
      w : resstype;
      rs : dix;
      pt : sept;
      ta1,ta2 : dixhuit;
```

```
BEGIN
  z := backl;
  WHILE frontl <> z DO
    BEGIN
      flag := 0;
      a := readyql(.frontl.).ind;
      x := readyql(.frontl.).process;
      y := readyql(.frontl.).task;
      IF resslsem(.tdlptlst(.x,y.).ressource.) = 0 THEN
        BEGIN
          IF tdlptlst(.x,y.).buffer = wlcfs THEN
            IF lcff = 1 THEN
              flag := 1;
          IF tdlptlst(.x,y.).buffer = wlcbs THEN
            IF lcbf = 1 THEN
              flag := 1;
          IF tdlptlst(.x,y.).buffer = wcom THEN
            BEGIN
              ctltm.request := comr;
              taskmanager;
              flag := ctltm.resp;
              work(.x,a.) := ctltm.address;
            END;
          IF flag = 0 THEN
            BEGIN
              IF tdlptlst(.x,y.).ressource=rcurrent THEN
                BEGIN
                  ctltm.request := cdbr;
                  taskmanager;
                  flag := ctltm.resp;
                END;
            END
        ELSE
          BEGIN
            flag := 1;
          END;
        IF flag = 0 THEN
          BEGIN
            i := tdlptlst(.x,y.).timeout;
            w := tdlptlst(.x,y.).ressource;
            i :=  counter + i;
            insertact(actllst,a,x,y,i);
            resslsem(.tdlptlst(.x,y.).ressource.) := 1;
            frontl := (frontl + 1) MOD 10;
            IF tdlptlst(.x,y.).buffer = wlcbs THEN
              lcbf := 1;
            IF tdlptlst(.x,y.).buffer = wlcfs THEN
              lcff := 1;
          END
        ELSE
          BEGIN
            readyql(.backl.).ind:=readyql(.frontl.).ind;
```

```
            readyql(.backl.).process:=readyql(.frontl.).process;
            readyql(.backl.).task:=readyql(.frontl.).task;
            backl := (backl + 1) MOD 10;
            frontl := (frontl +1) MOD 10;
          END;
      END;
    END;
```

PROCEDURE prserreq;

This procedure represents the look-up table of the process SERVICE REQUEST.

```
    VAR
      i : integer;

    BEGIN
      CASE l@.task OF
        twasr:
          BEGIN
            ctll.bustask := twasr;
            insertctl(ctll.base,1);
            busctl;
            insertql(l@.ind,serreq,upserreq);
            resslsem(.rbusctl.) := 0;
          END;
        upserreq:
          BEGIN
            i := 0;
            ctll.curtask := upserreq;
            insertctl(ctll.base,2);
            insertctl(ctll.base,1);
            curdbctl(i);
            resslsem(.rcurrent.) := 0;
            ctltm.request := cdbd;
            taskmanager;
            lcff := 0;
            finish(.serreq,l@.ind.) := 1;
          END;
      END;
    END;
```

PROCEDURE prupdreq;

This procedure implements the look-table of the process UPDATING REQUEST.

```
    VAR
      i : integer;

    BEGIN
      CASE l@.task OF
```

```
        twaup:
          BEGIN
            ctl1.bustask := twaup;
            insertctl(ctl1.base,1);
            busctl;
            insertql(l@.ind,updreq,upserres);
            resslsem(.rbusctl.) := 0;
          END;
        upserres:
         BEGIN
           i := 0;
           ctl1.curtask := upserres;
           insertctl(ctl1.base,2);
           insertctl(ctl1.base,1);
           insertctl(ctl1.base,1);
           curdbctl(i);
           resslsem(.rcurrent.) := 0;
           ctltm.request := cdbd;
           lcff := 0;
           taskmanager;
           finish(.updreq,l@.ind.) := 1;
         END;
      upward:
        BEGIN
          i := 0;
          ctl1.curtask := upward;
          insertctl(ctl1.base,2);
          insertctl(ctl1.base,1);
          curdbctl(i);
          resslsem(.rcurrent.) := 0;
          ctltm.request := cdbd;
          taskmanager;
          lcff := 0;
          finish(.updreq,l@.ind.) := 1;
        END;
      END;
    END;

  PROCEDURE praddinf;
```

This procedure implements the look-table of the process ADDITIONAL INFORMATION REQUEST.

```
    VAR
      i : integer;

    BEGIN
      CASE l@.task OF
        queryreq:
          BEGIN
            i := 0;
            ctl1.curtask := queryreq;
            insertctl(ctl1.base,1);
            insertctl(ctl1.base,50);
```

```
            curdbctl(i);
            resslsem(.rcurrent.) := 0;
            ctltm.request := cdbd;
            taskmanager;
            insertql(l@.ind,addinf,taddinf);
          END;
        dmalcb:
          BEGIN
            ctll.dmatask := dmalcb;
            insertctl(ctll.base,10);
            insertctl(ctll.base,work(.addinf,l@.ind.));
            pdmal;
            resslsem(.dmal.) := 0;
            ctltm.request := comd;
            ctltm.address := work(.addinf,l@.ind.);
            taskmanager;
            insertql(l@.ind,addinf,taddinf);
          END;
        taddinf:
          BEGIN
            ctll.bustask := taddinf;
            insertctl(ctll.base,50);
            busctl;
            resslsem(.rbusctl.) := 0;
            finish(.addinf,l@.ind.) := 1;
          END;
      END;
    END;
  PROCEDURE pradadm;
```

This procedure represents the process ADMITTING A PATIENT FROM ADMISSION.

```
    BEGIN
      CASE l@.task OF
        dmalcb:
          BEGIN
            ctll.dmatask := dmalcb;
            insertctl(ctll.base,30);
            insertctl(ctll.base,work(.adadm,l@.ind.));
            pdmal;
            resslsem(.dmal.) := 0;
            ctltm.request := comd;
            ctltm.address := work(.adadm,l@.ind.);
            taskmanager;
            insertql(l@.ind,adadm,bnwr);
          END;
        bnwr:
          BEGIN
            ctll.bustask := bnwr;
            insertctl(ctll.base,50);
            busctl;
            resslsem(.rbusctl.) := 0;
```

```
            lcbf := 0;
            finish(.adadm,1@.ind.) := 1;
            num(.adadm,1@.ind.).ck := counter;
          END;
      END;
    END;

PROCEDURE prademer;
```

This procedure represents the look-up table of the process ADMITTIMG A PATIENT FROM EMERGENCY.

```
    VAR
      i : integer;

    BEGIN
      CASE 1@.task OF
        dmalcb:
          BEGIN
            ctl1.dmatask := dmalcb;
            insertctl(ctl1.base,20);
            insertctl(ctl1.base,work(.ademer,1@.ind.));
            pdmal;
            resslsem(.dmal.) := 0;
            ctltm.request := comd;
            ctltm.address := work(.ademer,1@.ind.);
            taskmanager;
            insertql(1@.ind,ademer,bemr);
          END;
        bemr:
          BEGIN
            ctl1.bustask := bemr;
            insertctl(ctl1.base,50);
            busctl;
            resslsem(.rbusctl.) := 0;
            lcbf := 0;
            insertql(1@.ind,ademer,upemerfp);
          END;
        upemerfp:
          BEGIN
            i := 0;
            ctl1.curtask := upemerfp;
            insertctl(ctl1.base,2);
            insertctl(ctl1.base,1);
            curdbctl(i);
            resslsem(.rcurrent.) := 0;
            ctltm.request := cdbd;
            taskmanager;
            lcff := 0;
            finish(.ademer,1@.ind.) := 1;
          END;
      END;
    END;
```

```
PROCEDURE prrelease;

  This procedure implements the look-up table of the
RELEASING A PATIENT process.

    VAR
      i : integer;
    BEGIN
      CASE l@.task OF
        breour:
          BEGIN
            ctl1.bustask := breour;
            insertctl(ctl1.base,1);
            busctl;
            resslsem(.rbusctl.) := 0;
            IF indent(.release,l@.ind.) = 0 THEN
              indent(.release,l@.ind.) := 1
            ELSE
              BEGIN
                indent(.release,l@.ind.) := 0;
                insertql(l@.ind,release,bdec);
              END;
          END;
        checkoutreq:
          BEGIN
            i := 0;
            ctl1.curtask := checkoutreq;
            insertctl(ctl1.base,1);
            curdbctl(i);
            resslsem(.rcurrent.) := 0;
            ctltm.request := cdbd;
            taskmanager;
            i := removectl(ctl1.base);
            IF indent(.release,l@.ind.) = 0 THEN
              indent(.release,l@.ind.) := 1
            ELSE
              BEGIN
                indent(.release,l@.ind.) := 0;
                insertql(l@.ind,release,bdec);
              END;
          END;
        bdec:
          BEGIN
            ctl1.bustask := bdec;
            insertctl(ctl1.base,1);
            busctl;
            resslsem(.rbusctl.) := 0;
            insertql(l@.ind,release,dmalcom);
          END;
        dmalcom:
          BEGIN
            ctl1.dmatask := dmalcom;
            insertctl(ctl1.base,10);
            insertctl(ctl1.base,work(.release,l@.ind.));
```

```
            pdma1;
            resslsem(.dma1.) := 0;
            lcff := 0;
            insertq2(l@.ind,release,dmaacb);
          END;
      END;
    END;

PROCEDURE taskdr2;

   This procedure simulates the operation of task driver 2.
The task driver 2 is implemented in the same fashion as the
task driver 1.

BEGIN
  activate;
  WHILE l <> DO
    BEGIN
      IF l@.timeout = counter THEN
        BEGIN
          remact(act2lst,l);
          CASE l@.process OF
            adadm: pradadm;
            ademer: prademer;
            addinf: praddinf;
            release: prreleasev
          END;
        END;
      l := l@.prochain;
    END; END;

top

  PROCEDURE activate;

   For each task in the ready list (READY2), this procedure
checks if the task is ready to be executed.

    VAR
      a,i,flag,z : integer;
      x : processtype;
      y : tasktype;
      w : resstype;
      rs : dix;
      pt : sept;
      ta1,ta2 : dixhuit;

    BEGIN
      z := back2;
      WHILE front2 <> z DO
        BEGIN
          flag := 0;
```

```
a := readyq2(.front2.).ind;
x := readyq2(.front2.).process;
y := readyq2(.front2.).task;
IF ress2sem(.td2ptlst(.x,y.).ressource.) = 0 THEN
  BEGIN
    IF td2ptlst(.x,y.).buffer = wacfs THEN
      IF acff = 1 THEN
        flag := 1;
    IF td2ptlst(.x,y.).buffer = wacbs THEN
      IF acbf = 1 THEN
        flag := 1;
    IF x = adadm THEN
      IF (y = retrieve) or (y = atradm) THEN
        IF admf = 1 THEN
          BEGIN
            flag := 0;
            admf := 0;
          END
        ELSE
          IF acbf = 0 THEN
            admf := 1;
    IF td2ptlst(.x,y.).buffer = wcom THEN
      BEGIN
        ctltm.request := comr;
        taskmanager;
        flag := ctltm.resp;
        work(.x,a.) := ctltm.address;
      END;
    IF flag = 0 THEN
      BEGIN
        IF td2ptlst(.x,y.).ressource=rcurrent THEN
          BEGIN
            ctltm.request := cdbr;
            taskmanager;
            flag := ctltm.resp;
          END;
      END;
  END
ELSE
  BEGIN
    flag := 1;
  END;
IF flag = 0 THEN
  BEGIN
    i := td2ptlst(.x,y.).timeout;
    w := td2ptlst(.x,y.).ressource;
    i := counter + i;
    insertact(act2lst,a,x,y,i);
    ress2sem(.td2ptlst(.x,y.).ressource.) := 1;
    front2 := (front2 + 1) MOD 10;
    IF td2ptlst(.x,y.).buffer = wacbs THEN
      acbf := 1;
    IF td2ptlst(.x,y.).buffer = wacfs THEN
      acff := 1;
```

```
          END
        ELSE
          BEGIN
            readyq2(.back2.).ind := readyq2(.front2.).ind;
            readyq2(.back2.).process:=readyq2(.front2.).process;
            readyq2(.back2.).task:=readyq2(.front2.).task;
            back2 := (back2 + 1) MOD 10;
            front2 := (front2 + 1) MOD 10;
          END;
    END;
  END;

PROCEDURE pradadm;
```

This procedure represents the look-up table of the process ADMITTING A PATIENT FROM ADMISSION.

```
  VAR
    i : integer;

  BEGIN
    CASE 1@.task OF
      retrieve:
        BEGIN
          ctl2.arctask := retrieve;
          insertctl(ctl2.base,1);
          insertctl(ctl2.base,3);
          insertctl(ctl2.base,50);
          arcdbctl;
          IF indent(.adadm,1@.ind.) = 0 THEN
            indent(.adadm,1@.ind.) := 1
          ELSE
            BEGIN
              indent(.adadm,1@.ind.) := 0;
              insertq2(1@.ind,adadm,inadmp);
              insertq2(1@.ind,adadm,dma2com);
            END;
          ress2sem(.rarchival.) := 0;
        END;
      atradm:
        BEGIN
          ctl2.admtask := atradm;
          insertctl(ctl2.base,70);
          admission;
          ress2sem(.radmission.) := 0;
          IF indent(.adadm,1@.ind.) = 0 THEN
            indent(.adadm,1@.ind.) := 1
          ELSE
            BEGIN
              indent(.adadm,1@.ind.) := 0;
              insertq2(1@.ind,adadm,inadmp);
              insertq2(1@.ind,adadm,dma2com);
            END;
        END;
```

```
        atrnr:
          BEGIN
            ctl2.admtask := atrnr;
            insertctl(ctl2.base,50);
            admission;
            ress2sem(.radmission.) := 0;
            insertq2(1@.ind,adadm,inadmp);
            insertq2(1@.ind,adadm,dma2com);
          END;
        inadmp:
          BEGIN
            i := 1;
            ctl2.curtask := inadmp;
            insertctl(ctl2.base,1);
            insertctl(ctl2.base,50);
            curdbctl(i);
            ress2sem(.rcurrent.) := 0;
            ctltm.request := cdbd;
            taskmanager;
            IF flag1 = 0 THEN
              flag1 := 1
            ELSE
              BEGIN
                acbf := 0;
                flag1 := 0;
              END;
          END;
        dma2com:
          BEGIN
            ctl2.dmatask := dma2com;
            insertctl(ctl2.base,30);
            insertctl(ctl2.base,work(.adadm,1@.ind.));
            pdma2;
            ress2sem(.dma2.) := 0;
            insertql(1@.ind,adadm,dmalcb);
            IF flag1 = 0 THEN
              flag1 := 1
            ELSE
              BEGIN
                acbf := 0;
                flag1 := 0;
              END;
          END;
        END;
    END;

  PROCEDURE prademer;
```

This procedure represents the look-table of the process ADMITTING A PATIENT FROM EMERGENCY.

```
  VAR
    i : integer;
```

```
BEGIN
  CASE l@.task OF
    retrieve:
      BEGIN
        ctl2.arctask := retrieve;
        insertctl(ctl2.base,1);
        insertctl(ctl2.base,1);
        insertctl(ctl2.base,50);
        arcdbctl;
        insertq2(l@.ind,ademer,inemerfp);
        insertq2(l@.ind,ademer,dma2com);
        ress2sem(.rarchival.) := 0;
      END;
    inemerfp:
      BEGIN
        i := 1;
        ctl2.curtask := inemerfp;
        insertctl(ctl2.base,2);
        insertctl(ctl2.base,50);
        curdbctl(i);
        ress2sem(.rcurrent.) := 0;
        ctltm.request := cdbd;
        taskmanager;
        IF flag2 = 0 THEN
          flag2 := 1
        ELSE
          BEGIN
            acbf := 0;
            flag2 := 0;
          END;
      END;
    dma2com:
      BEGIN
        ctl2.dmatask := dma2com;
        insertctl(ctl2.base,20);
        insertctl(ctl2.base,work(.ademer,l@.ind.));
        pdma2;
        ress2sem(.dma2.) := 0;
        insertql(l@.ind,ademer,dmalcb);
        IF flag2 = 0 THEN
          flag2 := 1
        ELSE
          BEGIN
            acbf := 0;
            flag2 := 0;
          END;
      END;
  END;
END;

PROCEDURE praddinf;
```

This procedure represents the look-up table of the process ADDITIONAL INFORMATION REQUEST.

```
BEGIN
  CASE 1@.task OF
    query:
      BEGIN
        ctl2.arctask := query;
        insertctl(ctl2.base,1);
        insertctl(ctl2.base,3);
        insertctl(ctl2.base,50);
        arcdbctl;
        insertq2(1@.ind,addinf,dma2com);
      END;
    dma2com:
      BEGIN
        ctl2.dmatask := dma2com;
        insertctl(ctl2.base,10);
        insertctl(ctl2.base,work(.addinf,1@.ind.));
        pdma2;
        ress2sem(.dma2.) := 0;
        insertq1(1@.ind,addinf,dma1cb);
        acbf := 0;
      END;
  END;
END;
```

```
PROCEDURE prrelease;
```

This procedure represent the look-up table of the process RELEASE A PATIENT.

```
VAR
  i : integer;

BEGIN
  CASE 1@.task OF
    dmaacb:
      BEGIN
        ctl2.dmatask := dmaacb;
        insertctl(ctl2.base,10);
        insertctl(ctl2.base,work(.release,1@.ind.));
        pdma2;
        ress2sem(.dma2.) := 0;
        ctltm.request := comd;
        ctltm.address := work(.release,1@.ind.);
        insertq2(1@.ind,release,update);
      END;
    update:
      BEGIN
        ctl2.arctask := update;
        insertctl(ctl2.base,1);
        insertctl(ctl2.base,3);
```

```
            insertctl(ctl2.base,1);
            arcdbctl;
            insertq2(1@.ind,release,deletereq);
            ress2sem(.rarchival.) := 0;
          END;
        deletereq:
          BEGIN
            i := 1;
            ctl2.curtask := deletereq;
            insertctl(ctl2.base,1);
            insertctl(ctl2.base,1);
            curdbctl(i);
            ress2sem(.rcurrent.) := 0;
            ctltm.request := cdbd;
            taskmanager;
            finish(.release,1@.ind.) := 1;
            num(.release,1@,ind.).ck := counter;
          END;
      END;
  END;
```

REFERENCES

BARN 82 Barnett, G. O. et al, "COSTAR a Comprehensive Medical Information System for Ambulatory Care," Sixth Annual Symposium on Computer in Medical Care, Nov. 1982, pp. 8-18.

BEAM 79 Beaman, P. D., Justice, N. S., and G. O. Barnett, "A Medical Information System and Data Language for Ambulatory Practices," IEEE Computer, Nov. 1979, pp. 9-17.

BEND 82 Bender, M., "Distributed Databases: Needs and Solutions," Mini-Micro Systems, Oct. 82, pp. 229-238.

BENJ 80 Benjamin, B., "Medical Records." William Heinemann Medical Books LTD, London, 1980.

BERN 81 Bernstein P. A. and N. Goodman, "Concurrency Control in Distributed Data Base Systems," Computing Surveys, Vol. 13, No. 2, June 81.

BERR 80 Berra, P. B., "A Summary of the State of the Art in Data Base Machines," INRIA, France, 1980, pp. 11-38.

BLUM 82 Blum, B. I., et al, "Information Systems and Patient Care," Sixth Annual Symposium on Computer in Medical Care, Nov. 1982, pp. 3-7.

BUHR 84 Buhr, R. "System Design With Ada," Prenctice Hall, to be published.

CARR 75 Carren, D. M., "Multiple Minis for Information Management," Datamation, Sept. 1975, pp. 55-58.

CHAM 76 Chamberlin, D. D., "Relational Data-Base Management Systems," Computing Surveys, Vol. 8, No. 1, March 1976, pp. 44-66.

CHAM 77 Champine, G. A., "Six Approaches to Distributed Data Bases," Datamation, May 1977, pp. 69-72.

CHAM 79 Champine, G. A., "Current Trends in Data Base Systems," IEEE Computer, May 1979, pp. 27-41.

CHEN 77 Chen, P. P. and S. B. Yao, "Design and Performance Tools for Data Base Systems," Proceeding of 3rd Int. Conf. Very Large Data Bases, 1977, pp. 3-15.

CHI 82 Chi, S. C., "Advances in Computer Mass Storage Technology," IEEE Computer, May 1982, pp. 60-74.

COPE 82 Copeland, G., "What If Mass Storage Were Free?," IEEE Computer, July 1982, pp. 27-35.

DEEN 82 Deen, S. M., "Distributed databases - An Introduction," Distributed Data Bases, edited by H.-J. Schneider, North-Holland, 1982, pp. 239-246.

ECKH 78 Eckhouse, R. H., J. A. Stankowic and A. V. Dam, "Issue in Distributed Processing - An Overview of Two Workshops," IEEE Computer, Jan. 78, pp. 22-26.

ENSL 78 Enslow, P. H., "What is a "Distributed" Data Processing System?," IEEE Computer, Jan. 78, pp. 13-21.

ESTE 82 Esterhay, R. J., "Hospital Information System: Approaches to Screen Definition: Comparative Anatomy of the PROMIS, HIH, and DUKE Systems," Sixth Annual Symposium on Computer in Medical Care, Nov. 82, pp. 903-911.

ESTR 79 Estrin, T., and R. C. Uzgalis, "Information Systems for Patient Care," IEEE Computer, Nov. 1979, pp. 4-7.

FRIE 72 Fries, J. F., "Time-Oriented Patients Records and a Computer Databank," J. Am. Med. Assoc., Vol. 222, No. 12, 1972, pp. 1536-1542.

GIEB 75 Giebink, G. A. et al, "Current States of Ambulatory Health Care Computer Apllications," IEEE Computer, Jan. 1975, pp. 28-32.

GILM 81 Gilmore, G. H., and A. Ellis, "An Automated System for Medical Records Management," Fifth Annual Symposium on Computers in Medical Care, Nov. 1981, pp. 837-840.

HILL 81 Hill, C. L., and P. Balch, "On the Particular Applicability and Usefulness of Relational Data Bases Systems for the Management and Analysis of Medical Data," Fifth Annual Symposium on Computers in Medical Care, Nov. 1981, pp. 841-846.

HSIA 79 Hsiao, D. K. "Guest Editor's Introduction: Data Base Machines Are Coming, Data Base Machines Are Coming!," IEEE Computer, March 1979, pp. 7-9.

KERR 79 Kerr, D. S., "Data Base Machines with Large Content-Addressable Blocks and Structural Information Processors," IEEE Computer, March 1979, pp. 64-79.

KRIE 81 Krieger, M. and E. T. Fathi, "Design Aspects of a Simple Distributed Microprocessor System," International Conference on Communications, Circuits and Systems, Judavpor University, Calcutta, Dec. 1981.

KRIE 82 Krieger, M. and N. Santoro, "Oligarchical Control of Distributed Processing Systems," Proc. 20th Allerton Conf. on Communication, Control, and Computing, 1982, pp. 793-802.

LINC 83 Lincoln, T. L., "Medical Information Science," J. Am. Med. Assoc., Vol. 249, No. 5, 1983, pp. 610-612.

LITW 82 Litwin, W., "SIRIUS Systems for Distributed Data Management," Distributed Data Bases, edited by H.-J. Scheider, North-Holland, 1982, pp. 311-361.

LOWE 82 Lowenthal, E. "Multiuser Microprocessor Systems get a Data-Base Manager," Electronics, June 30, 1982, pp. 113-117.

MACD 79 Macdonald, C. et al, "Minicomputer Improves Clinical Health Care," Mini-Micro Systems, October 1979, pp. 86-92.

MALC 79 Malcom, L. S., "Surveying Data Base Management Systems," Mini-Micro Systems, Nov. 1979, pp. 94-104.

MART 77 Martin, J., "Computer Data-Base Organization," Second Edition, Prentice-Hall, Inc., Englewood Cliff, N.J., 1977.

MARY 78 Maryanski, F. J. "A Survey of Development in Distributed Data Base Management Systems," IEEE Computer, Feb. 1978, pp. 28-36.

MCKE 82 Mckendrick, M. D. "Dedicated Machine Offers Host-Independent Data-Base Management," Computer Technology Review, Vol. 3, No. 1, Winter 1982, pp. 11-17.

MISH 81 Mishelevich, J. D., "Hospital Information System Tutorial: a Guide for Computers Scientists and Practitioners," National Computer Conference, 1981, pp. 63-638.

MORT 82 Morton, D., "Computers in the Clinical Laboratory," Apllications of Computers in Medecine, IEEE, 1982, pp. 254-267.

MYER 82 Myers, J., "Advances in Computer Architecture," John Wiley and Sons, inc., New York, 1982.

NEUH 82 Néuh, E. J., and B. Walter, "An overview of the Architecture of the Distributed Data Base System "Porel"," Distributed Data Bases, edited by H.-J. Schneiner, North-Holland, 1982, pp. 247-290.

PARK 83 Parker, D. S. and et al., "Detection of Mutual Inconsistency in Distributed Systems," IEEE Trans. Software Engineering, Vol. SE-1, No. 3, May 1983, pp. 240-246.

PEEB 78 Peebles, R. and E. Manning, "Systems Architecture for Distributed Data Management," IEEE Computer, Jan. 78, pp. 40-48.

POLK 81 Polk, W. J. and K. Byrd, "Managing the Very Large Database," Datamation, 1981, pp. 115-124.

RUMB 75 Rumbaugh, J., "Data Flow Languages," Sagamore Computer Conference on Parallel Processing, 1975, pp.217-219.

SIBL 76 Sibly, E. H., "The development of Data-Base Technology," Computer Surveys, Vol. 8, No. 1, March 76, pp.1-5.

SCHK 78 Schkolnick, M., "Physical Database Design Techniques," NYU Symp. Database Design, may 1978, pp. 99-110.

SCHU 79 Schultz, J. R. and L. Davis, "The Technology of PROMIS," Proceedings of the IEEE, Vol. 67, No. 9, April 1979, pp. 1237-1244.

SCHT 82 Schteingart, R. et al, "Development of an Integrated Medical Record System at a Large Hospital Using COSTAR as a Designing Tool," Sixth Annual Symposium on Computer in Medical Care, Nov. 1979, pp. 166-171.

SKEE 83 Skeen, D. and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," IEEE Trans. Software Engineering, Vol. SE-1, No. 3, May 1983, pp. 219-228.

SREE 80 Sreekaouth, S. I. and T. A. Marsland "The Deadlock Problem: An Overview," IEEE Computer, Sept. 1980, pp. 57-70.

SU 79 Su, S. Y. W. "Cellular-Logic Devices: Concepts and Applications," IEEE Computer, March 1979, pp. 11-25.

TEOR 82 Teorey, T. J. and J. P. Fry, "Design of Database Structures," Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

TOLC 81 Tolchin, S. G. et al, "Implementation of a Prototype Generalized Network Technology for Hospitals," Fifth Annual Symposium on Computers in Medical Care, Nov. 1981, pp. 942-948.

ULLM 80 Ullman, J. D. "Principles of Database Systems," Computer Science Press, In., Potomac, Maryland, 1980.

WEIT 80 Weitzman, C. "Distributed Micro/Minicomputer Systems Structure, Implementation, and Application," Prentice-Hall, Inc., Englewood, N.J., 1980.

WHIT 81 Whithing-O'keefe, Q. E. et al, "The Argument for Modular Distributed Hospital Information Systems," Fifth Annual Symposium on Computers in Medical Care, Nov. 1981, pp. 912-916.

WIED 75 Wiederhold, G., Fries J. F., and S. Weyl, "Structures Organization of Clinical Data Bases," National Computer Conference, 1975, pp. 479-484.

WIED 83 Wiederhold, G., "Database Technology in Health Care," Journal of Medical Systems, Vol. 5, No. 3, 1983, pp. 175-193.

WIST 82 Wist, A. O., R. E. Horowitz, and R. Megargle, "Computers in the Clinical Laboratory," Application of Computers in Medecine, 1982, pp.244-267.

YAO 78 Yao, S. B., et al., "Data-Base Systems," IEEE Computer, Sept. 1978, pp. 46-60.