National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform 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.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme 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.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

A Database for an Intensive Care Unit

Emile Saab

Department of Electrical Engineering
McGill University
Montréal
October, 1995

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Engineering

ISBN 0-612-12136-4

Abstract

The rapid growth of medical sciences and technologies created the need to manage data generated by sophisticated medical equipment (e.g. lab results, vital signs, etc.). This class of equipment, especially in the modern Intensive Care Unit (ICU), emits large quantities of patient data which medical staff usually records on log sheets.

This thesis presents a database design that allows abstract definition of data types, and offers a unified view of data during the development phase, distinct levels of data management and a higher degree of system flexibility. This database model is an implementation of a database for a Patient Data Management System (PDMS) developed for use in the ICU of the Montreal Children's Hospital. The PDMS has a variety of application modules that handle and process various types of data according to functionality requirements.

Sommaire

Le développement de la technologie appliquée dans les unités de soins intensifs a permit une surveillance médicale constante et l'affichage graphique des données physiologiques des patients. Par contre, l'équipement utilisé dans cette surveillance génère une grande et vaste quantité d'information qui est typiquement enregistrée manuellement sur papier.

Cette dissertation présente un modèle de base de données qui permet l'abstraction des données et qui en l'appliquant offre une vue uniforme des données durant la phase de conception du système, de différents niveaux de gérance d'information et un plus grand degré de flexibilité du système.

Cette base de données est appliquée dans la réalisation d'un Système de Gestion de Données Médicales (SGDM) pour les unités de soins intensifs de l'Hôpital pour Enfants de Montréal. L'intégration des données des différents modules du SGDM permettra de gérer les données médicales telles que les paramètres physiologiques, les équilibres liquidiens et les données administratives des patients.

Acknowledgments

I thank my thesis supervisor Dr. Alfred Malowany for his encouragement, guidance during my research and for the confidence he had in me in carrying out my research. I also wish to thank him for the opportunity he provided me to discover and develop many interests.

Thanks also go to my PDMS colleagues who started this interesting and rewarding project and for all the advice and help they provided me.

I wish to thank Gilles for encouraging me throughout the pursuit of my Master's degree and Wassim for convincing me to pursue a Master's degree in the first place.

My deepest thanks to Elaine for her support, encouragement, help and patience, and to her family for their gratefully appreciated support.

Finally, my thoughts go out to my parents, for their confidence in me, sacrifices, patience, and "long distance" encouragement.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 Introduction

In the past two decades computer technology has impacted the practice of medicine in dramatic ways. The application of this technology may range from the molecular level to the individual patient or clinical level and ultimately to a complete health care delivery system. It is now being recognised that computer systems constitute a powerful tool for the management of the patient record and the surveillance of patient data.

The advance in technology allowed the introduction of sophisticated monitoring equipment in the ICU, which led to more detailed and larger amounts of patient data. The equipment encountered in a typical ICU is capable of monitoring the vital signs of all the patients in the ward around the clock. The bedside monitoring equipment usually displays the data on small screens and does not provide a history of the patient's status. Thus, it lacks data management capabilities. In a typical non-computerised ICU the medical staff samples the data from the bedside monitors every half hour and records it manually on log sheets. Later, the log sheets are used to manually chart or plot the data into graphs. The log sheets and the charts are used as analysis tools for the patient's status and are eventually archived in file cabinets.

Managing such large amounts of data manually can pose a heavy burden on the health care staff, and requires long work hours. This system can become inadequate when urgent and precise action is required. Furthermore, the manual system is prone to human error and data omission.

A computerised, automated data management system can employ all the benefits of the sophisticated medical equipment. What is needed is an integral data management system that is capable of processing data and information flow in ICUs such as data acquisition, data storage and archiving, data analysis and monitoring, and communication of critical data. Moreover, such a system should assist the medical staff in the administrative and scheduling tasks, for example patient registration, patient medication timetable and dosage, nursing task planning, scheduling and the like.

There are many factors that contribute to the development of a successful patient data management system. The system must satisfy the functional requirements identified at its inception, and be developed according to some formal system specifications. Any application system should have a set of application programs that interact with a database, in order to be defined as *complete* [Brown, 1991]. Moreover, the success of database implementations relies on a formal definition of the precise semantics of the database commands, the design methodology, and the data model [Lynngbaek and Kent, 1991] [Korth and Silberschatz, 1991].

Intensive care unit management is dependent on the hospital and the "culture" of its administrators. Every hospital operates its intensive care ward in a specific, if not unique, manner. Hence, the context and the data representation in a patient data management system vary from one ward to another. It sometimes varies within the same ward, for instance a physician could ask for the vital signs data of a patient to be plotted (in a certain colour) over a period of time, while another medical staff reviewing the patient's history could ask for a listing of such information.

Thus, the design and development of a database for an intensive care unit data management system will be confronted with the following issues: Which database design method and database model should be employed in the design of the system in order to accommodate the diverse and complex data types, satisfy the functional requirements, reduce the cost of the database maintainability and promote the system's flexibility, all while producing a portable and reusable database design?

The following sections will present a historical overview of ICUs and will delineate the various types of data that constitute the information administered in an ICU. Finally, the last section of this chapter will give an overview of this thesis, which presents the design and implementation of a database for an ICU patient data management system.

1.1 Historical aspects of ICUs

Intensive care units which provide a higher degree of patient care than the rest of the departments in a regular hospital have witnessed evolutionary changes associated with the medical and technological progress. The intensive care unit represents a development of the concept of progressive care where the facilities, equipment, and staff are grouped according to the intensity and nature of care that should be provided to a patient, rather than according to any other classification, such as one that relates to a disease entity.

As early precursors of the intensive care unit, some would credit Florence Nightingale for the recovery areas ("recesses") in an earlier military conflict more than 100 years ago [Nightingale, 1963]. Although the origin of intensive care units is somehow indistinct, it is clear that they have been conceived as an extension of existing post-anaesthesia recovery rooms in order to provide an adequate 24 hour surveillance [Hilberman, 1975].

In 1923 the earliest post-operational neurosurgical recovery unit was developed at the John Hopkins Hospital; soon afterwards in the late 1920s Kirschner designed a post-operational area for recovery and care of surgical patients [Kirschner, 1930]. During World War II the battle field hospitals of the U. S. Armed Forces introduced pre-operation and post-operation units, as well as post anaesthesia recovery areas. After World War II community hospitals were left with an inadequate supply of expert nurses to keep up with the demands of the individual care of critically ill patients. To alleviate this problem and to deploy resources efficiently, seriously ill patients were grouped together in rooms specially organised for the purpose. The advantages observed were rapidly emulated in larger institutions [Hilberman, 1975].

It was not until the early 1950s that respiratory care units became prominent. They emerged as a result of spontaneous efforts, some to cope with problems of tetanus and others to manage the large number of polio patients. At that time respiratory units began to emerge in different parts of the world, one by Reuben Cherniak in Winnipeg, one in Oxford at the Churchill Hospital by Campton-Smith and Spandeling, another in Minneapolis by Frederick Van Bergen. Due to their efforts and those of many others, like

Eric Nilsson, the older Drinker-type tank respirators were replaced by positive pressure mechanical ventilating devices [Morris, 1977].

A considerable contribution to the evolution of intensive care areas came as the result of the special needs of post-operative cardiac surgery. The growth of coronary care units is also believed to be a result of experiences gained in post-operative cardiac surgery care. One of the first coronary care units in a teaching centre was at the University of Toronto. Toronto also received international acclaim for its acute respiratory care unit developed in 1958 as a group effort between anaesthesia, medicine, and otolaryngology, which is still an important asset at the Toronto General Hospital [Fairley, 1961].

Indeed, the need for efficiency has always been the motivation for the concentration of patients and equipment in one area. These areas are conceived in accordance with the need to centralise equipment, services, talented people, and to conserve the energies of a limited number of personnel dedicated to a particular type of care. Consequently, we now have units oriented towards explicit fields of care such as paediatric, neonatal, surgical, trauma or shock, etc. [Morris, 1977].

1.2 Information in ICUs

Blum [Blum, 1984] identifies three processing elements in medical computing;

1. *Data*. These are individual elements that can be quantified.
2. *Information*. This consists of elements derived from data through analysis.
3. *Knowledge*. This is the set of rules which defines the relationships among information (and, by extension, data).

In the ICU, medical personnel collect a large amount of data through continuous monitoring, data sampling, testing, observing and recording the information. In addition, high demands are placed on the medical staff in the intensive care environment. Unique skills are required, along with accurate and prompt treatment decisions. In his Textbook of Critical Care, Weil [Weil *et al.*, 1989] defines critically-ill patients as patients who require complex and time-critical therapies.

Thus, in order to present all the important facts about a patient's condition and to facilitate the decision making of the physician, this data should be conveyed in a compact, organised form, where important events or trends could not be missed. Patients requiring treatment decisions to be made within minutes are the norm rather than the exception. Under such conditions physicians need rapid access to all relevant information concerning the patient. In practice, unfortunately, the medical record is often not available, not up to date, or in the case of newly admitted patients, non-existent.

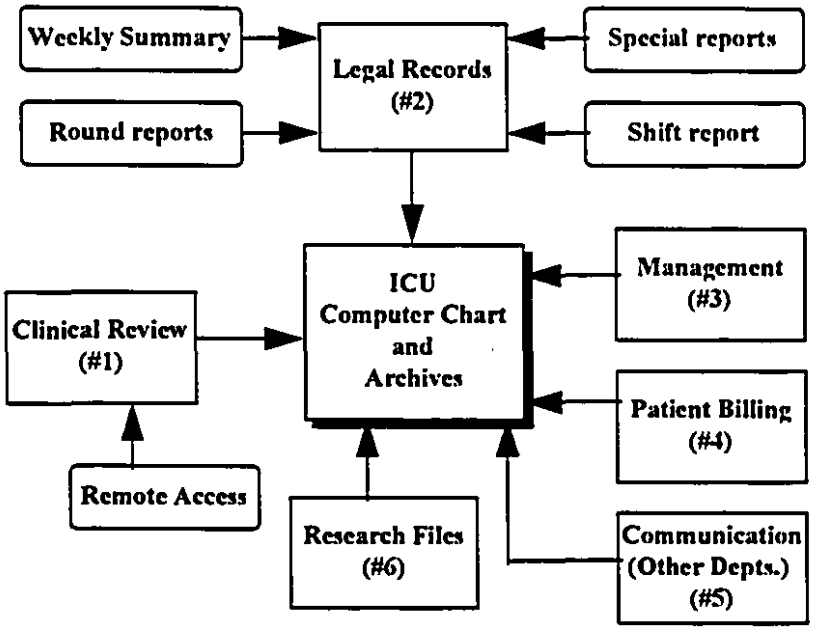


Figure 1.1: ICU Chart Complexity.

A patient chart, also called the patient's record, has regularly constituted the main repository for a patient's medical data. The Textbook of Critical Care [Gardner *et al.*, 1989b] discusses the major issues involved in ICU charting such as data display and presentation, data acquisition, and data storage. Figure 1.1 illustrates the complexity involved with each of these issues. The patient record must contain all relevant patient data as well as the action undertaken by the medical personnel. After having been acquired, this data should be organised for medical and legal requirements (Figure 1.1, #1 and #2), for use in the administration (Figure 1.1, #3) and billing purposes (Figure 1.1, #4). Interdepartmental communications within a hospital are mandatory (Figure 1.1, #5).

The chart data is then properly archived to fulfil the legal requirements, the archived data is also essential for use in research (Figure 1.1, #6).

Thus, the patient's medical record is the main instrument for patient care. A study by Whitinh-O'Keefe shows that the conventional medical chart does not offer the adequate support for proper patient management [Whitinh-O'Keefe *et al.*, 1985]. Bradshaw [Bradshaw *et al.*, 1984] suggests that this lack of adequate support is especially evident in the ICU where advanced techniques of patient monitoring produce large volumes of data for evaluation and where decisions have to be quickly undertaken, usually at a time when data are incompletely understood.

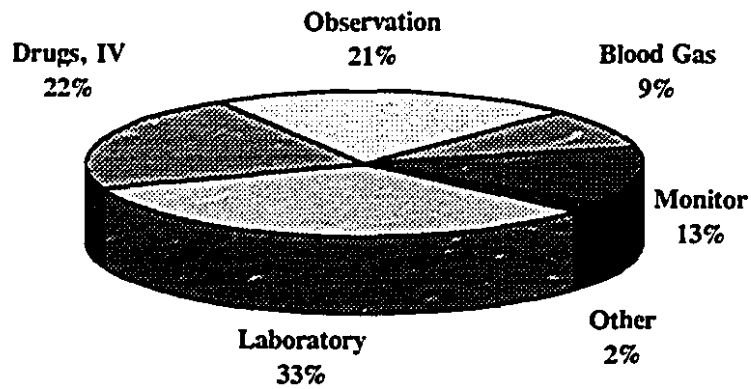


Figure 1.2: ICU Chart Use.

Figure 1.2 illustrates the different types of data employed in an ICU, as well as their percentages of use in an ICU Chart, according to a study conducted by Bradshaw [Bradshaw *et al.*, 1989]. This study showed that 33% of the physician's treatment decision weight relies on laboratory data, 22% on fluids balance data, 21% on clinical observations and finally 13% on bedside physiological monitor readings.

In the following subsections we will examine some of the data types that constitute the ICU information and the patient medical record, a comparison will be also drawn between handling each data type manually or via computerisation.

1.2.1 Physiological Monitoring Systems

In many cases the patient's parameters monitored by the bedside physiological monitors are not automatically recorded in the patient's chart. The nurse takes readings at one hour or half hour intervals and enters them into the paper chart [Collet *et al.*, 1989]. Transcription errors occur or scraps of paper that are used to temporarily hold information are misplaced or forgotten [Hendrickson *et al.*, 1991], [Nolan-Avila and Shabot, 1987], [Soontit, 1987], [Staggers, 1988]. Hammond *et al.* [Hammond *et al.*, 1991a] report that nearly one third of all errors in an ICU involve mistakes in charting or relaying information between shifts. Furthermore, relevant data of events that could occur in an interval between two readings is lost. Therefore the automatic acquisition of physiological data is seen as a primary goal. When McDonald *et al.* [McDonald *et al.*, 1988] developed their Regenstreif Medical Records System they started with laboratory results and vital signs, in their opinion the easiest type of data to capture. Milholland [Milholland, 1988] references many of these computer-based monitoring systems.

Dasta [Dasta, 1990] mentions the importance of computerised data acquisition in ICU wards, where a computer-based system can interface directly with the instruments and the data can be collected and stored in its native electronic form. This is bound to eliminate the burden and errors in data transcription.

Collet *et al.* [Collet *et al.*, 1990] describes a Trend Analysis module in a PDMS which analyses cardio-vascular data from physiological monitors in order to generate an early warning alarm. Trend patterns are recognised and analysed by a multilevel expert system. This Trend Analysis System recognises and considers relationships among the different parameters. Andreoli and Musser [Andreoli and Musser, 1985] criticise many monitoring systems for their inability to comprehend such relationships.

Subramanian [Subramanian, 1989] describes a microcomputer-based obstetrics information management system that monitors an eight-bed unit continuously. Analogue information from a fetal monitor is digitised and stored in an IBM personal computer. Waveform information can be displayed on a central monitor for all eight beds simultaneously. The University of Louisville Medical School [Strickland Jr., 1991] has developed an information management system to assist in the monitoring of patients with severe head injuries. It supplements the monitoring done by the ICU nurses and concentrates on the recording of the brain functions. Several other examples of computer-based medical systems can be found in the literature, Nagel and Smith [EMBS, 1991] in the IEEE Engineering in Medicine and Biology Society Annual Conference, Bankman and Tsitlik in the IEEE Symposium on Computer-Based Medical Systems, and other conferences have sessions dedicated to automatic medical data acquisition. Alesch [Alesch *et al.*, 1991] describes a system based on a PC/AT which controls 64 parallel data acquisition channels for recording data readings from neurological monitors. Aukburg *et al.* implemented a system for automating alarm data acquisition in a post anaesthesia care unit [Aukburg *et al.*, 1989].

Tachakra *et al.* [Tachakra *et al.*, 1990] and Kari *et al.* [Kari *et al.*, 1990] presented, in two of their articles, an evaluation and comparison of computerised systems versus manual data management systems. In these articles, studies demonstrated that the computerised procedure of data acquisition and storage generates significant savings in time and personnel workload, and also enhances the data accuracy.

1.2.2 Drugs, IV, Blood Gas, and Observations

In an ICU the fluid balance of the patients must be constantly monitored. Fluid balance accounts for the patient's fluid intake (ingesta) and fluid output (excreta). The nurse takes periodic measurements from infusion pumps, urine bags, blood gas, injected or lost blood, as well as other gastric or fluid measurements and enters these numbers into the fluid balance chart. Medications are also entered here. Running totals must be calculated and abnormalities noted. Fluid balance sheets are often difficult to read; in many cases they

contain cumbersome erased and re-recorded values and calculations. The calculation of running totals is prone to human error, and constitutes further tedious staff workload. Observations and tasks required to care for the patient are scheduled and planned in a "nursing workload measurement scoring sheet". These sheets contain tasks such as the initial assessment of the patient and reassessment, meeting the patient's care needs, and the planning and carrying out of interventions to meet those needs. The aforementioned tasks are divided into eight categories: respiration, nutrition and hydration, elimination, personal care, ambulation, combination, treatments and diagnostic procedures [Roger *et al.*, 1992]. Creating a nursing care plan is one of the activities of a nurse in the ICU, an activity that requires a considerable amount of time for preparation and follow-up.

The fluid balance measurements and the generation of the care plans as well as the measurement of nursing workload are all suited for computerisation. Computer-assisted care plans have the potential to decrease the time spent creating care plans and measuring nursing workloads. They improve the quality of the care plan, institutionalise standards of intensive care and perform the role of a reminder process for critical tasks [Allen, 1991].

In a study of data management systems conducted by Hammond *et al.* [Hammond *et al.*, 1991a, Hammond *et al.*, 1991b], results showed that the use of computerised systems in care units not only reduced non-nursing related work, but also improved the quality of the clinical information and the information recall by the medical or administrative staff. Indeed, this leads to a significant improvement in documentation flowsheets both in terms of quantity and accuracy which in turn have quality assurance impacts, as well as enhancing patient care.

1.3 Medical Informatics

This section presents a definition of a Database Management System and then extends it to a Medical-Based Data Management System. It will also describe medical-based systems in the context of intensive care units.

1.3.1 Medical Information Systems

Many different definitions of a Data Management System (DMS) could be found in the literature; nonetheless, they all agree that data management consists of everything that occurs to data from the process of defining what is to be collected up to its analysis. A Data Base Management System (DBMS) is defined by King [King *et al.*, 1984] to generally include the following functions;

1. Define and restrict the data objects to be stored and organised according to a predefined database structure.
2. Check the consistency of entered values with the object definition and the contents of the database.
3. Edit previously-recorded values.
4. Create subsets of the data objects characterised by a query.
5. Control access to the data.
6. Allow data retrieval in different types of forms.

Kriewall and Long [Kriewall and Long, 1991] identify three categories of computer-based medical systems; *controllers* such as drug infusion systems or artificial hearts, *diagnostic tools* such as blood pressure monitors or ICU monitors, and *information managers* that handle networks or interfaces. Andreoli and Musser [Andreoli and Musser, 1985] distinguish three types of computer applications for patient care: patient monitoring computer systems, medical information systems and computer assisted diagnostic systems.

Over the past sixteen years, a large number of Patient Data Management Systems have been developed and installed in hospitals. Many of these installations have been in the Intensive Care Unit (ICU). Gardner [Gardner *et al.*, 1989a] identifies four functions of computers in an ICU setting:

1. Physiological monitoring
2. Computers that facilitate the timely and accurate communication of data among multiple hospital locations and departments.

3. Management of medical records.
4. Expert computer systems to aid in patient care decision making.

The following section will present an overview of patient data management systems in ICUs, the challenges that such systems encounter and the research orientation of this specific field.

1.4 Data Management Systems in ICUs

In an intensive care unit integral and comprehensive documentation of diagnostic and therapeutic patient data is a crucial prerequisite for assessing the patient's physiological condition as well as deciding on treatment regimes.

Furthermore, in an ICU more variables such as vital signs and events must be observed, documented and evaluated. A study conducted by [Thull *et al.*, 1993] shows that a nurse spends up to 30 percent of his time keeping documentation up-to-date and that during a visit a physician typically spends up to 50 percent of his time evaluating and updating documentation.

A computerized data management system for such an environment is an adequate solution for reducing the time spent on managing ICU documentation and for helping the ICU medical staff focus on other important patient care tasks. However, the work in computerised ICU information management systems has just begun, model-based monitoring and management systems are in their infancy and no clinical systems yet exist [Mora *et al.*, 1993]. "Off-the-shelf" clinical or hospital management software packages ignore the data management aspect of an ICU because the information management requirements of such an environment differ from one hospital to another. Thus, the development of a common software package that aids in managing ICU information seems almost impossible and very expensive, from an industry point of view.

Therefore, most patient information systems similar to the PDMS resorted to design research and custom development for implementing computerised data management

systems in ICUs. An important aspect in any research undertaking is to identify the fundamental questions to be answered.

Considerable efforts are still expected in the acquisition of on-line data, not only in the form of signals, but also the registration of actions, change of settings in the monitoring devices, etc. [Mora *et al*, 1993]. Work is being focused on the feature extraction and trend analysis blocks in order to deal with noise and unreliable patient information. Most intelligent patient monitoring systems assume that this block is somehow solved, but practical implementations show a different story [Collet *et al.*, 1990, Moret-Bonillo *et al.*, 1993, Dawant *et al.*, 1993, Watt *et al.*, 1993].

Another direction is the design of generic prototypes, such that a monitoring and management system can be customised for specific applications such as anaesthesia or neonatal surveillance [*Mora et al.*, 1993]. This research deals with the identification of domain-independent concepts for databases that can be restructured in an adaptable architecture [Sukuvaara et al., 1993, Dawant *et al.*, 1993].

The next section will describe projects that are similar to the PDMS and discuss some of the challenges they have encountered. It will also delineate the functional differences between these projects and the PDMS.

1.4.1 Similar Work

The PATRICIA project at the La Coruna university, in Spain, used a semantic-based methodology to implement an intelligent monitoring system in the ICU [Moret-Bonillo *et al.*, 1993]. The PATRICIA system has been designed to provide the following functions: (1) monitoring and data acquisition, (2) interpretation of that information, (3) establishment of diagnosis according to the patient's condition and (4) prescription of therapeutic guidelines.

[Moret-Bonillo *et al.*, 1993] describes the following main challenges encountered during the design and development of the PATRICIA system:

1. diversity of information sources, as clinical information and data usually come from different sources.

2. handling of raw electronic data, as parameters are obtained from electronic equipment connected to the patient.

The SIMON project [Dawant *et al.*, 1993] employed a Model-Based design in order to acquire, analyse and interpret monitoring signals data as part of an intelligent patient monitoring system. On top of the aforementioned issues encountered during the development of the PATRICIA system, developers of the SIMON system identified the following challenges:

1. the dynamic nature of the design and implementation processes.

2. the limitation in computing resources.

The Patient Data Management System (PDMS) project, under way at McGill university, offers functionality that is similar to the PATRICIA and SIMON projects in the domain of patient data monitoring. However, the PDMS offers database management of signal data, such as look-up, patient history review and patient data archive. Moreover, the PDMS offers a more integral ICU information management system that handles nursing care plans, patient fluids balances and patient admittance records.

The database aspect of the PDMS renders the system useful not only in the ICU environment but also outside of that context. The query management facility of the PDMS can help detect any common patterns in patient diagnoses such as epidemics; it can also produce *ad hoc* statistics reports. Furthermore, the data stored or archived in the PDMS database can be used in case-study simulations.

1.4.2 The PDMS approach

The direction of the PDMS project is to design, as much as possible, a generic integral prototype for patient data management systems in ICUs. The PDMS also addresses the issue of on-line electronic data acquisition.

The PDMS database is only one aspect of the PDMS project, nonetheless an important one. This thesis addresses the problem of designing and developing a database for the PDMS as well as the methodology used in that process.

Using a clear and concise method in the design of a database for a patient data management system we can avoid many of the software development problems, such as the expensive cost of coping with changes in the system, high cost of maintenance and the difficulty in expressing the design. Employing a software engineering approach in the design of the database has the following advantages:

- Using a model-based design, the problem is tackled at an abstract level that renders the solution domain-independent and adaptable to more than one environment.
- With such a database design model, we can test the database for faults and accommodate changes in the system before actually building the database.
- The design model can be used as a communication tools that effectively conveys the database design between designers and programmers.
- Such an approach is highly recommended in large or complex systems, database design models reduce complexity by separating out a small number of important things to deal with at a time.
- Such a model serves as the blue print of the database, it reduces significantly the time that a database administrator or maintainer spends in understanding the cause of the problem and amending it.

As we can conclude from this chapter, managing information in the Intensive Care Unit (ICU) is a complex and difficult process, yet also delicate and essential to the survival of patients. The information generated or handled in the ICU is characterised by its diversity and large quantity. In a typical intensive care environment, information varies among the administrative, the pharmaceutical, medical records, patient parameter data, scheduling, and many other types.

The success of building a database for a PDMS in an ICU relies particularly on the database design. This thesis presents the design and implementation of such a database. Chapter Two describes the patient Data Management System (PDMS) for which the database is built. Chapter Three attempts to answer the following question: Which database design methodology is best suitable for the PDMS? Furthermore, which data modelling technique better ensures the domain abstraction and portability of that database design? It also presents an overview of two schools of thought for data modelling, as well as a survey of current database modelling techniques. Chapter Four applies the chosen technique in the design of the PDMS database. Finally, chapter Five discusses the results of the database design implementation.

Chapter 2 The PDMS

This chapter describes the Patient Data Management System (PDMS), a research project being developed at the McGill Research Centre for Intelligent Machines (CIM) in conjunction with the Paediatric Intensive Care Unit of the Montreal Children's Hospital. The first section of this chapter will outline the hardware and software platforms of the PDMS. The following section will discuss the functionality of the PDMS by presenting an overview of the different modules of the system.

2.1 PDMS Hardware and Software Platforms

In the paediatric intensive care units of the Montreal Children's Hospital, physiological bedside monitoring is based on a Hewlett-Packard CareNet system. The CareNet system is composed of fourteen HP 78534A physiological bedside monitors. The monitors are capable of smoothing measured parameters, real-time display of measured data and alarm generation. These monitors are linked in a star configuration local area network to a HP78581A Network System Communications Controller. One of the monitors Network Controller branches is connected to an HP78588A Careport. This unit provides a programmable interface between the network controller and a host computer system. The function of the Careport Network Interface is to translate the proprietary network messages and signal formats to the standard RS-232C messages which can be understood by the host computer (Figure 2.1).

The host computer system at the hospital presently consists of a 486 IBM PC clone with 16 Megabytes of RAM memory, and a 300 Megabyte hard disk. The display consists of a high resolution 8514/A adapter which has a resolution of 1024×768 pixels. A colour printer, the EPSON LQ-2550, provides the colour printout of the PDMS screens in addition to printing out the required forms and reports for the nurses. Soon, it is envisaged to extend the number of computers by linking them in a Token-Ring Local Area Network.

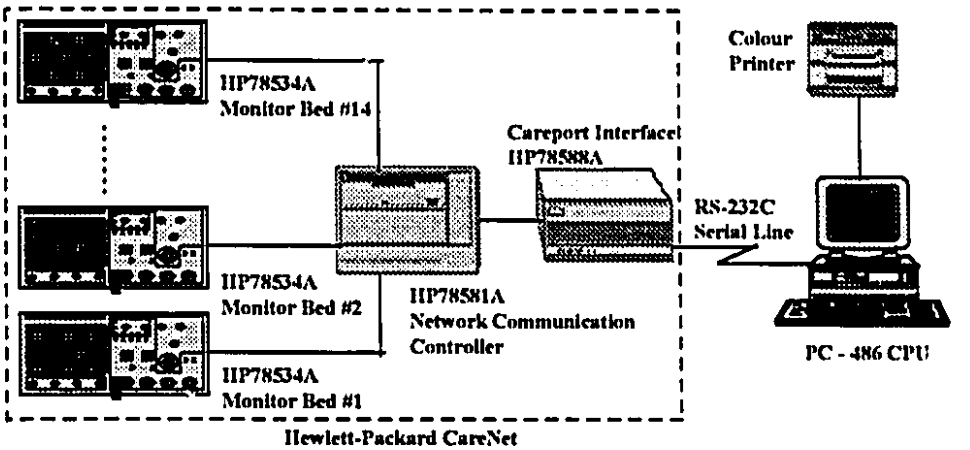


Figure 2.1: PDMS Network Configuration.

The PDMS is being developed for an IBM PS/2 network of computers running the OS/2 multitasking operating system version 1.3 and 2.0. OS/2's Presentation Manager is used to develop a window-based consistent user-friendly interface. The window interface together with a high resolution screen (1024×768×256 colours) enhance the multitasking feature of OS/2™, where multiple modules of the PDMS can be invoked concurrently. For example, several trends can be displayed on one workstation for analysis while another workstation or window could be reviewing fluid balance data sheets. The database is designed to be spread over different sites on the computer network. Files are distributed according to the location of their most frequent use and the type of data contained in them. The IBM Communication Manager provides remote file access, ensuring the database distributivity. Due to the multitasking facilities in OS/2, the PDMS is developed in a modular fashion.

2.2 The PDMS Functions and Modules

There are eight general categories of functions carried out in an ICU: patient admission, cardiovascular monitoring, fluid balance monitoring, the preparation of nursing care plans, the measurement of nursing workload, the request of laboratory tests and entry of

laboratory results, and the request of pharmaceuticals. Each functional category will be described along with the corresponding PDMS module.

The PDMS is build in a modular software architecture. The modules which are currently implemented under the present version of the PDMS are:

- *The Data Link Controller (DLC)* acquires and stores the vital signs data transmitted by the network of bedside monitors in real-time mode. Data acquisition is done through a serial RS-232C link to the Careport [Fumai *et al.*, 1991].
- *The Trend Display Module* interactively displays patient vital signs data and alarm data in graphical trends on a CRT, using combinations of colours and symbols [Fumai *et al.*, 1991].
- *The Fluid Balance Module* saves and computes all substances taken in (ingesta) or put out (excreta) by a patient, in a spread-sheet format. A user speech interface for the fluid balance module has been developed [Petroni *et al.*, 1991].
- *The Patient Registration Module* manages the administrative information such as the address of a patient, previous diagnostics, hospital identification number, and the like.
- *The Trend Analysis Module* is a multilevel expert system. This module performs appropriate analysis on the real-time data, scanning the trends for critical combinations and transmitting messages in case of triggering [Collet *et al.*, 1990].
- *The Nursing Workload Scheduler Module* focuses on improving the process of nursing care plan generation, workload measurement and workload scheduling [Roger *et al.*, 1992].

2.2.1 Patient Admission

The Registration Module manages the administrative information of the patients upon their reception to the ICU. This information consists of the admission date and the discharge date. The patient registration module handles also general information about a patient such as: name, age, sex, address, telephone number, hospital identification number,

ICU bed number, treating physician, diagnosis, etc. This module gives the user control over network activities such as collection of data of a bed from the network.

2.2.2 Cardiovascular Monitoring

The Data Link Controller (DLC) module is responsible for interfacing the PDMS with the CareNet bedside monitors through the Careport interface. The DLC manages the gathered data for easy access by other modules, and it stores the information in the PDMS database for future consultation and archiving purposes.

The main tasks of the DLC are to acquire the physiological parameter values from the bedside monitors every two seconds and to store them into circular queues. The *seconds* data is averaged every minute, and these values are inserted in minute queues. The *minute* values are in turn averaged every half hour and stored into the half hour queues. All the queues are located in shared memory. These values can then be accessed by the Trends module for graphical trend display.

The sampling rate of the DLC data acquisition from the Careport interface (every two seconds) is chosen because it appears to be adequate for present needs and lies within the operational constraints of the equipment available at this time. Moreover, alarm messages may be transmitted by the Careport to the DLC as they occur (asynchronously). The different functions of the DLC such as parameter values averaging, acquisition and transmission of data are implemented as distinct processes or threads.

The vital signs of the patient are monitored in the ICU through both, the Hewlett-Packard CareNet System (for example, blood pressure) and the nursing observation (for example, pupil size). The Trend Display module displays the vital signs monitored by the CareNet System on the 8514/A colour monitor of the personal computer. The user can interactively select which types of parameter to display at the same time on the screen, and thus be able to observe the correlation among these parameters. Vital signs data can be displayed in real-time for a remote site, or can be retrieved from the database. Since only

a limited quantity of data can be viewed on the screen, the screen appears as a sliding window which can be used to display any part of the trend data.

A Vital Sign Monitoring System is developed to complement the Trend Display Module. This system performs appropriate real-time analysis by scanning the trends for critical combinations. It generates warnings about slowly varying trends or short interval disturbances [Collet *et al.*, 1990].

2.2.3 Fluid Balance Monitoring

Patients must be monitored for their fluid balance. Fluid balance accounts for the patient's fluid intake (ingesta) and fluid output (excreta). This has to be monitored because of the effect the balance has on blood pressure, dehydration, pooling, drowning or thrombosis. The nurse periodically reads measurements from infusion pumps or urine bags, and record these figures onto the fluid balance sheet. Running totals must be calculated and abnormalities noted.

The Fluid Balance module enables the entry, calculation and correction of the volumes of all the fluid intake and output of a given patient. The user interface of the Fluid Balance module replicates the paper fluid balance chart used in the ICU. The computerised chart is set up as a spreadsheet and key combinations are used to navigate through the chart. The interface was developed using OS/2's Presentation Manager.

In order to avoid the cumbersome manual navigation and data entry in the Fluid Balance module, a speech interface has been developed. The interface uses both speech recognition, for data entry and module operation, and speech generation, for feedback, verification of spoken commands, and audio prompting that provides an "eyes-free" and "hands-free" means of directly entering fluid balance data at the bedside and at a distance from the PDMS console [Petroni *et al.*, 1991].

2.2.4 Measurements of nursing workload

Regularly, the nurse prepares a nursing care plan for a patient upon admission to the ICU, and updates it as the patient's state requires. Once the care plan has been filled out the nurse must assign a score to the plan.

The Nursing Workload Manager also called the *Nursing Care Plan* module focuses on improving the process of nursing care plan generation, workload measurement and workload scheduling. The Nursing Workload Manager will generate nursing care plans, automate PRN workload measurement scoring, schedule nursing activities and set up fluid balance charts by integrating with the Fluid Balance module described Section 2.2.3 [Roger *et al.*, 1992].

Chapter 3 Design Approach

The conventional wisdom in software engineering holds it as self-evident that a system design must be described in three dimensions: those of *process*, *control* (or dynamics) and *data*. The *process* models are usually described by dataflow diagrams (DFDs), such a diagram depicts all the processes found in the system and the data types or elements that are transferred among them. The *control* models are described by an entity life notation or state diagrams, such a diagram depicts the life cycle or the transformation phases that the system's entities undergo during runtime. Finally, the *data* domain corresponds to logical data models which depicts the data elements managed at different levels or parts of the designed system [DeMarco, 1978][Yourdon and Constantine, 1979][Gane and Sarson, 1979].

Data-centered approaches to software engineering begin with the data model. Data is more stable than process or control functions and so data-centered approaches are to be preferred in most cases, since they intend to produce coherent system designs and highly integrated systems [Rumbaugh, 1991].

A data model is a set of concepts that can be used to describe the structure of a database. The structure of a database can include definition of data types, relationships and constraints that should hold for the data.

Why data modelling as a tool to database design? Determining correct, consistent and complete information is a difficult and challenging task [Kim and March, 1995]. A database model serves two main purposes during the design process:

1. *Build a conceptual (representation) of the enterprise reality.* This model serves as a communication vehicle between the database designers and the users.

2. *Design validation before implementation.* Before concluding that the model is correct, consistent and complete, it must be validated. Validation of a data model has two aspects: comprehension and discrepancy checking. Users must understand the

meaning of the model and they should also be able to identify any discrepancies between the model and their comprehension of reality.

Hence, the data model has a large impact on the design, implementation, and the effort required to develop the PDMS database. This chapter will present an overview of the approaches available for constructing the data model of the PDMS database. The second section will discuss the current data modelling practices. Finally, this chapter will compare the most currently applied data modelling techniques and choose a method for developing the PDMS database.

3.1 Data Modelling Approaches

Data modelling approaches can be categorised into two main schools. The first school, called the *rule-based* school, considers that the problem domain cannot be solved independently from the user and therefore promotes subjectivity in data modelling. The rule-based approach to data modelling analyses and models the organisational rules that govern the real world rather than the *facts* about the real world. The second school, called the *fact-based* school, promotes objectivity in data modelling and solves the problem domain by concentrating on *facts* only.

In the following sections we will present a brief overview of both schools and explain how a *rule* is perceived by the rule-based school and how a *fact* is perceived by the fact-based school of data modelling.

3.1.1 Rule-Based School

The rule-based approach to data modelling is derived from a combination of foundation concepts found in disciplines such as the philosophy of natural speaking, the hermeumitical philosophy and pragmatics [Goldkhul and Lyytinen, 1982].

The basic view of the rule-based school is that the information system interpretation of the data must be consistent with the meaning that this data conveys when interpreted according to certain human rules which apply in real world situations. This view regards

information modelling in terms of social actions or behaviour where language is the mediating force. Since social actions are perceived in terms of communication [Goldkhul and Lyytinen, 1982], this view revolves around the need to model the social interaction or activity that occurs in an organization. In particular, the information model should model the rules that govern this social interaction.

Therefore, language is studied as an important category of human action and becomes a social artifact whose primary function is to support human interaction. In this case, data modelling is concerned with interpreting meanings from the user's business language into a formal language [Hirschheim, 1985]. An information system is, therefore, considered to be a "formal linguistic system for communication between people which support their actions" [Goldkhul and Lyytinen, 1982, p. 14]. Hence, the *linguistic* view or *language action* view of the rule-based school sees an information system as a technical implementation of a *social organization* or a *social system* where a formal language is maintained, handled and transmitted using information technology [Goldkuhl and Lyytinen, 1984]. Since information has limited capabilities in modelling human behaviour, according to this belief, data modelling is regarded as a study of message meaning expressed in speech acts [Lyytinen and Lehtinen, 1984].

Another view in the rule-based approach to data modelling is the *social organization* view. This view models information systems after a social or business institution. It identifies classifications and types of acts and actors which compose the system. The issue, here, becomes decision-taking; what decisions are made, by whom and for what reason [Huber, 1983]. This belief extends the language action view to the social institution where information modelling forms an activity by which a communication organization is created and maintained [Zmud, 1979].

3.1.2 Fact-Based School

The fact-based approach to data modelling is widely adopted in the database design industry. It has been extensively researched over the past two decades. This implies that

the database design and modelling community has a better understanding of the strength and limitations of this school.

Data modelling in the fact-based school employs methods and tools derived from analytical philosophy and logic concepts. The fact-based approach is based on concepts which are related to a given *fact*. These concepts mainly assign a specific attribute or relationship to an entity that exists in the real world [Kent, 1983].

In the fact-based school, a *fact* is considered to be a state of affairs of an entity. Hence, it is possible to assert or deny that fact. Facts can concern an entity as well as a group of entities and entity-types such as *the employees of a company* and *car-manufacturers*. In current data modelling techniques, the entity groups and entity-types are referred to as *entity class* or *object class*. The fact-based school denotes prevailing facts as information. General facts that describe what is acceptable and what is necessary in the problem domain are denoted as *constraints*.

The fact-based school attempts to model objects found in the real world as inter-related data entities. These data entities along with their relationships constitute the information system. This is the main reason why most current fact-based data modelling methods refer to these entities as *objects* [Shlaer and Mellor, 1992].

There are two types of entities used in the fact-based approach, *concrete* and *abstract*. Concrete entities are physical objects mapped from the real world such as *a car* or *this table*. On the other hand, abstract entities are not tangible objects, yet they could be concepts or ideas that exist in the real world such as a *car-ownership*.

In this case, a data model is a conceptual schema that represents the solution of the domain problem or the information system. The model defines the entities of that domain and which facts hold for each entity. Therefore, the data model represents these entities and facts in constructs of formal language or linguistic objects known as sentences or assertions [Bacman, 1960] [Chen, 1976]. An example of such assertions would be *a car is owned by a person* or *a person must have a first and last name*.

3.1.3 Rule-Based vs. Fact-Based Approaches

In the rule-based approach, the focus is on reconstructing or modelling a set of rules that promotes understanding of the *social world*. Data modelling in the rule-based approach is regarded as a means to understand communication in the information system and reconstruct rules that govern it. These aspects make rule-based data modelling a complex and difficult approach [Lyytinen and Lehtinen, 1984].

However, data modelling in the rule-based approach is an important tool for organizations. It helps to understand and learn about the organization's communication practices, and to discover and resolve any discrepancies in the organization communication system. This approach also emphasizes the study of the mechanism by which the use of language initiates, empowers and controls organizational behaviour [Goldkhul and Lyytinen, 1984].

On the other hand, the main focus of the fact-based school is designing a data modelling method which would clearly and most important objectively describe the states (denial or assertion) of a reality or a real fact.

In the design of the PDMS database we adopted a fact-based data modelling technique because of the following reasons:

- The rule-based approach to data modelling is complex and difficult to apply fact-based information. Therefore, the time and effort required to design and implement the database would be colossal.
- The rule-based approach is not widely used by the database design community. This means that there is more to learn about the limitations and the technical capabilities of the rule-based school, whereas in the fact-based school, the opposite is true.
- Literature about rule-based modelling methods, tools and techniques is scarce. The major emphasis in the information systems research community has been on fact-based approaches. This explains why the majority of the data modelling methods and tools used in the industry belong to the fact-based school.

- Another point to consider is the design or model portability. A rule-based model is dependent on the rules, the human social interaction and the "business culture" adopted in the modelled organization. This renders the model very specific and almost non-reusable by similar organisations. On the other hand, a fact-based model models only the business facts managed by an organization. Hence, a fact-based model could be adopted by organizations that manage similar business facts.

- Finally, the modelling and storage of concrete fact-based data as is generated in an ICU environment is clearly, even at the intuitive level, more suited to implementation using fact-based methods.

In order to render the database design model of the PDMS reusable and more flexible, we need to carefully consider the aforementioned issues. Using a fact-based database modelling technique ensures that the designed PDMS database model could be used for another ICU, or extended by another development group. Although a rule-based model is can also be extended, the resources capable of extending a rule-based model are scarce due to the industry's lack of expertise in this domain.

Moreover, the aim of the PDMS database is not to model the social behaviour of an ICU but mainly to manage its daily data.

3.1.4 Fact-Based Database Models

Fact-based data models can be classified into four categories [Brodie, 1984]:

1. Primitive data models, which are essentially sequential files.
2. Classical data models, such as the network, the hierarchical and the relational data models.
3. Semantic data models, which are designed to provide more expressive concepts in order to capture more *facts* than the classical data models; for example, the entity-relationship (ER) data model.

4. Special purpose data models, which are more application-oriented. They present an evaluation of the semantic data models applied to particular software applications such as VLSI and CAD/CAM; for instance, object-oriented methods.

The classical and semantic data models apply to databases, thus they are also referred to as *database models* [Ozkarra and Zodnik, 1987]. In the following sections we will first overview each of these categories. Then, we will explain the selection of the database modelling technique best suited for the design and implementation of the PDMS database.

3.2 Historical Aspect of Database Technology

If we can skip the plug board programming period, the beginning of data storage was the *sequential file*. A search program had to read through a file from the beginning until the required record was matched and then the tape had to be rewound in preparation for the next file access. As more sophisticated data storage devices such as drums and disks emerged, programming languages were extended to include statements enabling direct access - called *random access*, in contrast to the sequential access required of magnetic tapes. This soon led to the realisation that access speed could be enhanced by hashing or storing index files.

The existence of indexed files made it possible to conceptualise a structural relationship among these files that would capture some of the application or the real world structures. Since most of the real world's organisational models are structured on the basis of the class membership notion, the world's first database products imposed a hierarchical structure among their files. *Hierarchical* databases such as IBM's Information Management System (IMS) were both popular and efficient. However, many business relationships did not fit into neat hierarchies and more general networks often emerged from the investigations of systems analysts. Few early documents which describe IMS exist; however, Date [Date, 1986] presents IMS as an example of hierarchical systems. *Network* systems were nearly as popular and nearly as efficient as their hierarchical ancestors. Both types of databases depended on fixed pointers and were very difficult to change and extend in response to business reorganisation. All the aforementioned

database products were developed in response to practical needs without the benefit of formal theories such as the *hierarchical data model* or the *network data model*. It was not until 1969 that the Bachman diagram [Bachman, 1969] was introduced by Charles Bachman, one of the originators of the network model of information. In 1971, the network model structures and language constructs were defined by the Conference on Data Systems Languages committee (CODASYL), hence it is often referred to as the CODASYL network model. The CODASYL network data model was revised in 1978 and 1981 to incorporate more recent concepts. In this respect, object-oriented programming seems to be repeating a part of the history of databases as an *ad hoc* technique lacking a formal theory.

The next significant development was the introduction of the first formal data model, Ted Codd's relational model of data [Codd, 1970]. It was based on the first-order predicate calculus (FOPC) and, equipped with this theoretical basis, it supported a relationally complete, non-procedural enquiry language. The *de facto* standard now established in the industry for relational data definition and manipulation, SQL, is loosely based on the relational calculus.

It was soon realised that that the relational model was not the only possible formal model, and that other models might retain certain advantages.

3.3 The Relational Data Model

Since the implementation platform provided for the PDMS employs a relational database manager and since relational databases form a large part of the current database practice, we will briefly review the main ideas of the relational model.

As we have already mentioned, the relational model was motivated by many aims, among them the desire to use formal methods in database design, enquiry and update, and the desire to be able to prove the correctness of programs based on non-procedural descriptions.

The relational model consists of two intrinsic and two extrinsic parts. The first intrinsic part is a structural part which uses the notions of *domains*, *n-ary relations*, *attributes*, *tuples* and *primary* and *foreign keys*. The second intrinsic part is a manipulative component whose main tools are relational algebra and/or calculus and relational assignments. The first extrinsic part deals with integrity with respect to both entities and references. The second extrinsic part is the design component consisting of the theory of Normal Forms [Date, 1981][Ullman, 1981][Elasmri and Navathe, 1989]. These following sections will briefly overview the aforementioned relational model concepts.

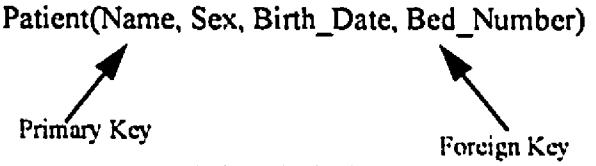
3.3.1 Intrinsic Structural Part

Mathematically defined, a *relation* is any subset of the Cartesian product of all lists of sets. Given a list of sets $A_1, \ldots, A_n$, their Cartesian product is the set of all lists or bags[1] of n elements of the A_i where there can be only one element in the bag from each A_i. Such a bag is called an ordered n-tuple, or just a *tuple*. The relation is sometimes called *n-ary* if there are *n* attributes (Figure 3.1 (a)). Each A_i is called a *domain* when viewed as a set of elements from which an attribute may take its values and an *attribute* when viewed as a label for that set (Figure 3.1 (c)). An equivalent notion is that of a table, and it is the most commonly used in the context of computers because of its strong physical analogy. A table is composed of rows and columns, where the columns represent the *attributes* of the table, and a row, called a *tuple*, represents the *relationship* among the attributes (Figure 3.1 (b)).

The next level of structure in a database deals with the relationships between *relations*. Chen [Chen, 1976] uses the terminology *entity-relations* and *relationship-relations* to distinguish the two types of relations, both of which must conform to certain integrity constraints. Furthermore, the relation must be in first Normal Form; that is, the attribute values must not be complex structures such as repeating groups or lists, but must be atomic data types such as integers and strings. The relationships-relations have two properties, multiplicity and modality. This level of structure is added to the relational

[1] A *bag* is a list wherein elements may be repeated, as opposed to a *set* where repetition is prohibited

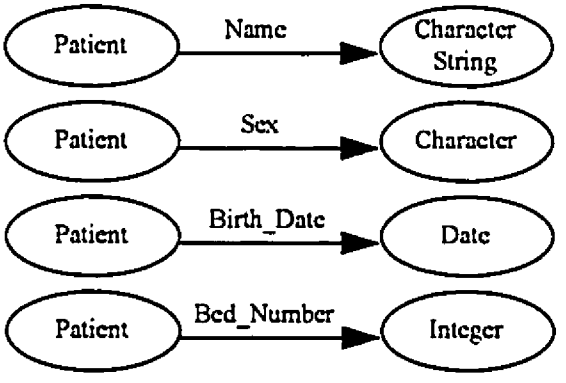
model and is not strictly a part of it. Integrity, multiplicity and modality constraints are normally coded in the application and usually in an exogenous procedural language. This part of the theory is often referred to as the *extended* relational analysis (ERA).



(a) A 4-ray relation with its four attributes, one primary key and one foreign key.

Name	Sex	Birth_Date	Bed_Number
John Doe	M	22/10/94	5
S. Bloe	F	27/06/93	9

(b) Some tuples from the Patient relation.



(c) The domains of the four attributes.

Figure 3.1: Relational Model.

3.3.2 Intrinsic Manipulative Part

The relational model has two essential manipulative methods. These two methods are known as the relational algebra and the relational calculus. A manipulation language is said to be *relationally complete* if any possible operation over the database may be prescribed within a single statement of the language. A language permitting all feasible

operations over a database, but in more than one statement, hence a procedural language, is operationally defined as a relational algebra.

The first method to emerge with Codd's original paper was the relational calculus, which is a retrieval and update language based on a subset of the first-order predicate calculus. The alternative approach to a single statement of predicate calculus is to regard enquiries and updates as a sequence of algebraic operations. Relational algebra is based on four primitive operations: selection, projection, join and union. The *selection* operation yields those tuples (rows) that satisfy the predicate or the selection condition. For example, we might wish to select all patients that are born before a certain date (Figure 3.1 (b)). On the other hand, the *projection* operation selects columns from that table and discards the rest. The *join* of two relations A and B over a relational operator "p" is obtained by building all the tuples that are the concatenation of a tuple from relation A and a tuple from relation B such that "p" holds for the attribute specified without duplication. If two tables have the same attribute, their *union* may be formed by appending them together and removing any duplicates in the primary key. This account of relational algebra has been simplified for brevity; for greater detail one may consult [Elmasri and Navathe, 1989] and [Date, 1981].

Several hybrid languages based partly on relational algebra exist, the most notable being those based on the IBM System-R language, Structured Query Language (SQL). SQL was initially derived from the motivation to produce a "structured" language. Later, it was found convenient to add into it most of the power of algebra and calculus. SQL is highly redundant and an inelegant language. However it has become the *de facto* industry standard, mainly because of its simple and convenient structure.

3.3.3 Integrity & Normal Forms

Before we turn to the design component of the relational mode, we need to define the concept of keys. A *primary key* is a set of one more attributes of a relation, whose value uniquely identifies tuple instances of that relation. A *foreign key* is an attribute which is the primary key of some other relation. The integrity rule specifies what happens to

relations related through foreign keys when a table is subject to delete or update operations.

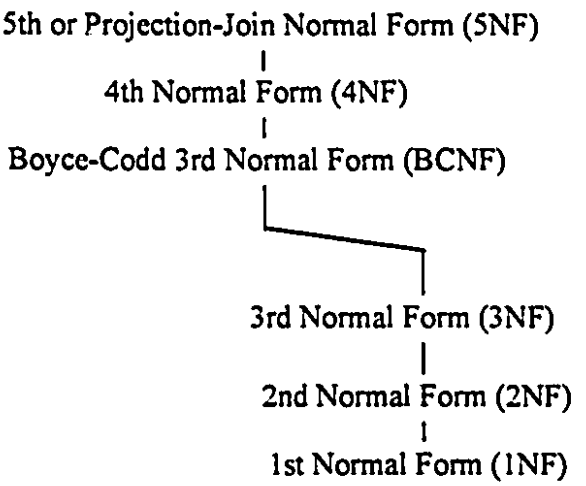


Figure 3.2: The hierarchy of Normal Forms.

Normal Forms are rules developed to avoid logical inconsistencies in table update operations. Each Normal Form prohibits a form of redundancy in table organisation that could yield inconsistent results if one table is updated independently of others. There are multiple levels of Normal Forms, with each higher level Normal Form adding a constraint to the Normal Form directly below it(Figure 3.2).

We mentioned earlier that relations must be in first Normal Form. In fact, this condition lies at the bottom of a the hierarchy of Normal Forms as shown in Figure 3.2, of which the most important are the Third or Boyce-Codd Normal Forms. The theory of Normal Forms is merely a way of formalising the common-sense notion of "good design". The notion of a good normalised database design will be explained in Section 4.3 (Testing for 3NF and BCNF).

To facilitate the definition of the various Normal Forms we will first define what is meant by one attribute being functionally dependent on another. An attribute is *functionally dependent* on another attribute if and only if each value of the second one uniquely determines the value of the first. Functional dependencies express concrete

relationships in the real world and require an understanding of the application. Functional dependencies generally cannot be discovered by an automatic process, only by a skilled systems analyst.

The First Normal Form, or 1NF, insists that the attribute value entries are "atomic"; that is, there are no "repeating groups" or lists, and values must be primitive data types rather than pointers. The Second Normal Form, or 2NF, says that tuples are uniquely defined by a primary key. The most useful form of 3NF, the Boyce-Codd NF, says that every determinant is a candidate key. In other words, every string of attributes that uniquely identifies a tuple could be the key. 4NF helps to avoid redundancy and 5NF prevents what are called lossy joins; that is, if two (or more) relations are joined and decomposed to the original form no data are lost [Ullman, 1981].

The theory of Normal Forms is an aspect of "bottom-up" design and is complementary to top-down design methods. It is not, necessarily, a part of the relational model. The whole theory of integrity is also not part of the model. This theory, as well as the relational Normal Forms will be used and discussed in more detail in Chapter 4.

3.4 Semantic Data Models

Semantic data models started with Abrail's [Abrail, 1974] binary relational model. The schemata of the binary model are essentially semantic networks restricted to classes. Chen's notation [Chen, 1976] was introduced originally as a notation for design. It had quite different goals than the relational model, which attempts to separate the physical and logical description of data and develop powerful non-procedural enquiry languages. Semantic data models were intended to allow the modelling of relationships and integrity. Modern semantic models intend to introduce very sophisticated means of handling inheritance, instantiation and subtyping. The general idea is to be able to model data at high and low levels of abstraction and to capture as much meaning as possible.

3.4.1 The ER Model

The Entity-Relationship (ER) model was first described by Peter Chen in 1976. The Chen basic ER model uses rectangles to specify entities, which are somehow analogous to records. It also uses diamond-shaped objects to represent the various types of relationships, which are differentiated by numbers of letters placed on the lines connecting the diamonds to the rectangle (Figure 3.3). Figure 3.3 models the following facts:

Assertion 1: Every patient has one or more admittance records into the ICU.

Assertion 2: A patient is characterised by a name and a gender.

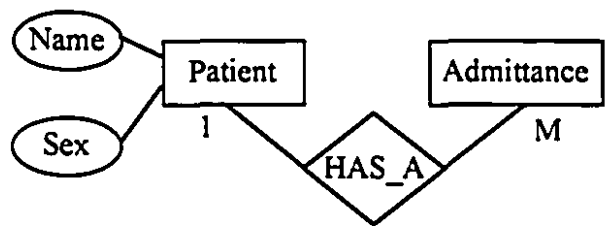


Figure 3.3:A patient (entity) has (relationship) many admittance records (entities) in an ICU.

ER models have two goals, to enforce business rules like referential integrity and to classify relations into types. The ER model has proven very useful in the field of database design. It represents the structure of the database in a simplistic and intuitively appealing fashion. The ER model plays a significant role in the construction of abstract and logical database designs, it also provides a suitable basis for enabling database designers to get an intuitive grasp of the real world. Since the original definition of entities and relationships, the ER model has undergone a variety of changes and extensions [*Teorey et al*, 1986].

The basic ER model consists of three basic constructs or classes of objects: *entities*, *attributes* and *relationships*.

Entities

Entities are the principal data object about which information is to be collected. An entity usually denotes a concrete or abstract object found in the domain we are modelling (or the real world), such as *a patient* (a person) or *an admittance sheet*. A particular occurrence of an entity is called an *entity instance* or simply an *instance*.

Attributes

Attributes are characteristics of entities or relationships; they provide details about them. A particular occurrence of an attribute within an entity or relationship is called an *attribute value*. The attribute construct for the ER model is an ellipse with the attribute name inside, as shown in Figure 3.3.

There are two types of attributes: *identifiers* or *descriptors*. An identifier (or key) is used to uniquely determine an instance of an entity. This often referred to as the *identification concept*. For instance, an entity name uniquely identifies the class or entity group. This means that the database system must have a mechanism that distinguishes entity instances by their identifiers. Therefore, every entity, relationship or any other type of construct in an ER model must have a unique and distinct name.

A descriptor (or nonkey attribute) is used to specify an nonunique characteristic of a particular entity instance.

Relationships

Relationships represent real-world associations among one or more entities. A particular occurrence of a relationship is called a *relationship instance*. Relationships are described in terms of *degree*, *connectivity*, *cardinality*, *attributes* and *existence*.

Degree of a Relationship

The degree of a relationship is the number of entities associated in the relationship. For instance, a *binary* relationship is a special case where the degree of the relationship is 2. A *ternary* relationship is of degree 3. Consequently, an *n-ary* relationship is the general form for any degree *n* relationship.

Connectivity and Cardinality of a Relationship

The *connectivity* of a relationship describes the mapping of the associated entity occurrences in the relationship. The values for connectivity are either *one* or *many*. The actual number associated with the connectivity is called the *cardinality* of the relationship connectivity. Cardinality describes the constraints on the number of entity instances that are related by a relationship. The most common cardinalities are *one-to-one*, *one-to-many* and *many-to-many*. Figure 3.3 shows a one-to-many binary relationship, the number 1 indicates the *one* side of the relationship and the letter M indicates the *many* side. M is sometimes replaced by N or P or it could simply indicate a number such as 2 or 3; in this case the cardinality of the relationship would a 1-to-3.

Attributes of a Relationship

Attributes can be assigned to entities as well as relationships, as shown in Figure 3.3. However, unlike what is shown in Figure 3.3 where at least one side of the relationship is a single entity, attributes are seldom assigned to one-to-one or one-to-many binary and ternary relationships because. Therefore, attributes are mainly assigned to many-to-many relationships.

Existence of a Relationship

In some cases, the existence of an entity may depend on the existence of another. This is called *existence dependency*, or simply *existence*. Existence of an entity as part of a relationship can be either *optional* or *mandatory*. If an instance of the *one* or *many* side entity must always exist in order for the entity to be included in the relationship, then it is a *mandatory* existence. When the instance of entity does not need to exist, the existences is considered *optional*. The representation of optional and mandatory existence vary from one notation to another. For example, in Chen's notation an optional existence is represented by a 0 on the relationship line.

3.4.2 Extended ER Models

As computers, programming and database implementation techniques advanced, the first ER model introduced by Chen failed to capture the intent of database designers during complex applications development. It lacked substructure presentation for entities and relationships. Smith and Smith [Smith and Smith, 1977] showed that the relational model was insufficiently expressive to capture the comprehensiveness of an Entity-Relationship (ER) model. This claim is evident from research that supports the extension and the replacement of the ER model. This research provided several semantically richer extensions to the basic ER model, some of them with similar names. The extended entity-relationship model, EER, introduced by Teorey *et al.* [Teorey *et al.*, 1986] includes new constructs to the ER model or notions which differentiate *generalisation* and subset *hierarchies (supertype/subtype)*. The enhanced entity-relationship model, also EER, was introduced by Elmasri and Navathe [Elmasri and Navathe, 1989] with the aim of being a superset of most other ER proposals. It introduced the concepts of *subclass* and *superclass*, and the related concepts of *specialisation* and *generalisation (aggregation/association)*, as well as the mechanism of attribute *inheritance*.

In order to achieve their goals, Teorey et al. and Elmasri and Navathe especially, were forced to introduce classes or *abstract classes* into their EER models much in the manner of object-orientation. It is instructive that these authors seem driven inexorably towards this introduction of classes. Possibly this indicates that the notion of abstract classes with inheritance is canonical; i.e. a "best" way of handling higher-order notions about entities and concepts in a data modelling context [Graham, 1994].

Hence, the most important concepts or constructs that were introduced as enhancements or extensions to the ER model are: classification, instantiation, specialisation, generalisation, aggregation and association. These are discussed below.

Classification and Instantiation

The process of *classification* involves grouping similar objects into object classes. In many cases, groups of objects share the same types of attributes and constraints,

classifying objects helps in defining their properties. *Instantiation* is the inverse of classification and refers to the generation of an object of a certain class.

In the EER model, entities are classified into entity types according to their structure and basic attributes. Moreover, entities are classified into subclasses and categories based on additional similarities or differences among them. Objects that are relationship instances are classified into relationship types. Hence, entities, subclasses, categories and relationships are the different types of classes in the EER model [Elmasri and Navathe, 1989].

Generalisation and Specification

Specialisation refers to the concept of classifying a class of objects into more specialised subclasses. Generalisation is the opposite; it refers to generalising several object classes into a higher level of abstraction which includes the objects in all these classes.

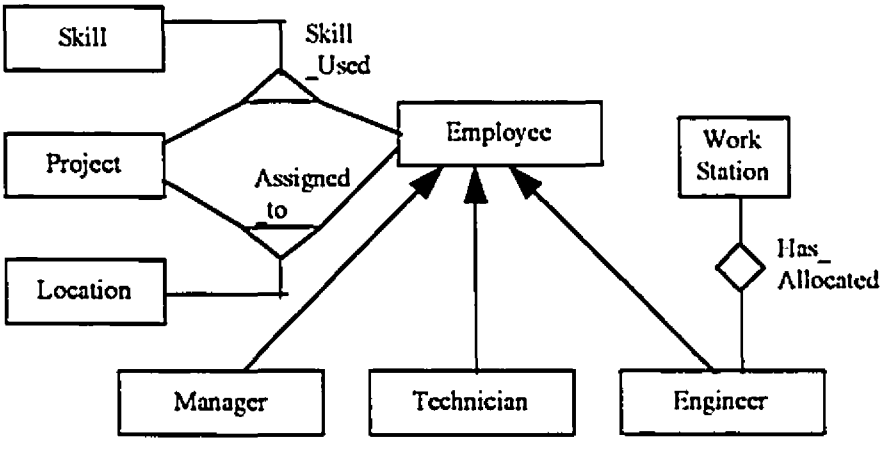


Figure 3.4: An EER data model showing binary, ternary relationships and inheritance (the thick arrows).

The EER models use the concepts of subclasses and object classes categorisation in order to model specialisation and generalisation. For example, Elmasri and Navathe [Elmasri and Navathe, 1989] use a relationship called *IS-A-SUBCLASS-OF* between a subclass and its super class in order to model generalisation and specialisation. Moreover, some EER models employ the subclass and superclass concepts in order to model attribute inheritance between object classes. For example, the typical EER model shown in Figure

3.4 uses thick arrows to indicate that *Technician*, *Manager* and *Engineer* are subclasses of the superclass *Employee*.

Aggregation and Association

Aggregation is an abstraction concept used for building objects from their component objects. It is the *part-whole* or *part-of* relationship in which objects representing the components of something are associated with an object representing the entire assembly. This type of relationship between the primitive object and its *aggregate* objects is often referred to, in an EER model, as the *IS-A-PART-OF* or *IS-A-COMPONENT-OF*. One common example is the bill-of-materials or parts explosion tree.

On the other hand, association is used to associate several independent classes. In an EER model this relationship is often referred as *IS-ASSOCIATED-WITH*. For example, a *company* is not an aggregation of its *employees*, since *company* and *person* are independent object classes of equal structure and therefore *a person* is associated with *a company*. Arguably, an aggregation is a special from of an association, not an independent class.

3.5 Object Oriented methods

The earliest work in computing goes back the late 1940s and it was exclusively concerned with what we now think of as programming. Only later did a conscious need for design and analysis tools arise. Similarly, it is object-oriented programming that first attracted attention and only recently have object-oriented analysis and design become major areas of interest.

Dyke and Kunz [Dyke and Kunz, 1989] claim that the designers of the Minutem missile used rudimentary object-oriented techniques as early as 1957. The history of object-oriented programming seems to start with the development of the discrete event simulation language Simula in Norway in 1967 and continues with the development of a language that almost makes a fetish of the notion of an object, called Smalltalk, in the 1970s. Samlltak was largely developed at the Xerox Research Centre in Palo Alto.

The 1980s showed an explosion of interest in the user interface (UI), which became the next phase of object-oriented programming [Larson, 1992]. Object-oriented programming supported the development of such user interfaces. Also, from the mid-1970s onwards, there was considerable cross-fertilisation between object-oriented programming and artificial intelligence (AI) research and development, leading to several useful extensions of AI languages, especially LISP [Winston, 1984].

As object-oriented programming began to mature, interest shifted to object-oriented design and analysis methods where researchers discovered that reusability and extensibility can be applied to designs and specifications as well as code. Prieto-Diaz and Freeman [Prieto-Diaz and Freeman, 1987] and Sommerville [Sommerville, 1989, 1992] have argued, in more general software engineering terms, that the higher level of reuse, the greater the benefit.

3.5.1 Object-Oriented Modelling

Object-oriented programming and database design incorporated many new ideas as well as well-established concepts such as information hiding into a coherent set of rules for data structure and data operations, including data abstraction, object communication through messaging, encapsulation of data structure and behaviour into the same object and sharing of data structure and code.

Object-oriented modelling and design methods inherited the aforementioned concepts from object-oriented databases and programming [Won, 1990, 1995]. Moreover, the object-oriented approach is characterized by the following four concepts: identity, classification, polymorphism and inheritance [Rumbaugh *et al.*, 1991]. The following is a presentation of these ideas and concepts:

- *Objects*. The basic unit of construction, be it conceptualisation, design or programming, is an abstraction of the real-world entities, much in the manner of ER modelling. Objects that exhibit common attributes, instances or behaviour are organised into *classes*.

- *Object Identity*. The object is uniquely identified in the system by an *object identifier* which is independent from its attribute values. Hence, any attribute of the object can be updated without destroying its identity.
- *Encapsulation*. The data structures and methods that manipulate the data of the object are hidden from the outside world and do not have to be known in order to access the object's data values or to invoke its methods. The *methods* associated with an object are a predefined set of procedures that manipulate the data of the object, they are also referred to as the *state* or the *behaviour* of the object. The only way to access the object's state is to send a message that causes one of the methods to execute.
- *Messages*. Objects, classes and their instances communicate by message passing. This eliminates data duplication and ensures that changes to data structures encapsulated within an object do not propagate their effect to other parts of the system. Messages are often implemented as function calls.
- *Inheritance*. Instances are capable of inheriting the features of the classes they belong to. This concept is also extended to classes, and known as *class hierarchy*, where classes are arranged in a hierarchy in which each class inherits all of the attributes and methods of its ancestors. Moreover, this concept allows object-oriented methods to construct *complex objects*, which is the ability to define new composite objects from previously defined objects in a nested or hierarchical manner.
- *Polymorphism*. The ability to use the same expression to denote different operations is referred to as polymorphism. For example, when the message "add 1" is sent both to a bank account and to a list of reminder notes; the same message should produce different results. Inheritance is a special kind of polymorphism that characterises object-oriented systems.

3.5.2 The OMG Abstract Object Model

The Object Management Group [OMG, 1993], a consortium of object technology vendors and interested parties, have defined a high-level reference model for object-oriented analysis and design. The OMG abstract data model is not a method, rather a framework

within which object-oriented software engineering methods can be evaluated. Figure 3.5 illustrates that reference model.

The OMG special interest group (SIG) who developed the reference model considered over 30 methods and none of them were excluded by the result. The need for the object model arose because of large variation in meaning given to terms like "object" and "method" in the literature. Does "object" mean an instance of an object or a class of objects? The object model now declares that *object type* shall be the correct term for class. *Methods* are defined as the implementation of *operations*. Appendix A contains a table which summarises the terminology of the object model and the various specialisation relationships between terms. However, designers will continue to say "object" for short, just as data modellers say "entity" for entity type.

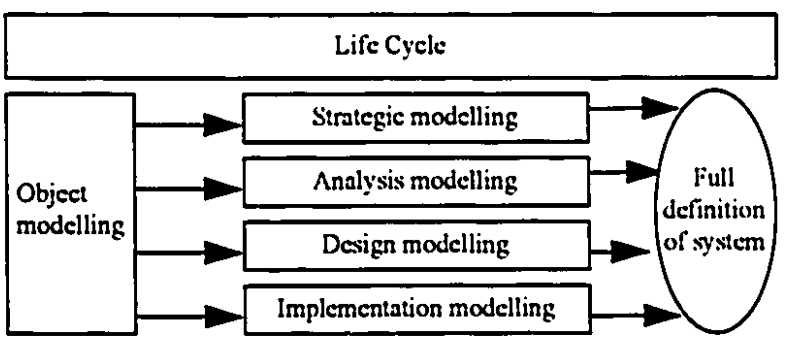


Figure 3.5: The OMG reference model.

Object modelling provides a set of terms and concepts for respecting everything within the scope of analysis and design as an object. It defines standard terms. *Strategic modelling* covers enterprise and business modelling, requirements capture and development planning. *Analysis modelling* covers the process of obtaining a description of the problem domain. *Design modelling* consists of adding non-public information to class specification and producing a solution to some particular problem including system objects. *Implementation modelling* consists of the physical design and involves designing modules, the distribution strategy and consideration of the software and hardware to be used.

3.6 Object Modelling vs. ER Modelling

After comparing 11 semantic data models, Biller and Neuhold [Biller and Neuhold, 1977] conclude that there are essentially two types of data modelling formalism, *entity-attribute-relationship* (EAR) and *object-relationship* (OR) models. Proponents of each claim that their model generates "better" design representations than the other [Everest, 1988].

Kim and March [Kim and March, 1995] presented an empirical study that compares two popular semantic fact-based data models: the EER model and *the Nijseen information analysis methodology* (NIAM) an object model. The NIAM model [Nijssen, 1977] is based on the early binary modelling work by Abrail [Abrail, 1974] and Senko [Senko, 1976]. It is widely used in Europe and Australia and is considered, along with the ER approach to be among the major approaches used internationally [Kim and March, 1995].

The study conducted by Kim and March was more of a context-sensitive empirical research in the data modelling area, with strong emphasis on external validity. In other words, the study examined the effects of EER and OR modelling on designer's performance in developing data models and, more importantly, on user's performance in validating data models.

The conclusion of that study showed that the EER and the OR models developed by analysts were significantly different in terms of their semantic quality but were not significantly different in terms of their syntactic quality. It also showed that the performance of designers and of users was subjective, depending on whether designers and users were familiar with a structured or object-oriented approach. However, this study also implied that in the case of modelling a database schema an object model has more to offer in terms of efficiency and design semantics than an EER model, and users can learn and comprehend an object model just as well as an EER model.

3.6.1 Current EER and OR

The object model, currently looming, combines many of the features and benefits of the relational and the EER models. Arguably, it is also the location of a convergence of ideas

from semantic data modelling, artificial intelligence and object-oriented programming, design and analysis.

An *attribute* of an object, in an object-oriented design method, is a descriptive property of an object in the same way an attribute describes an entity in an ER model. An *object instance* is a single occurrence of an object class, much like an *entity instance*. An *association* between two object classes, in the object-oriented modelling approach, is an abstraction of a link that exists in the real world between these two objects. Similar to the EER models, object-oriented models are expressive enough to capture the semantics of classification, aggregation, association and class identification.

Associations in the object model are abstracted in the same manner as in the EER, however the constructs used in depicting the model diagram are different. Associations in the object model are usually shown by a straight line between the two object classes it connects (Figure 3.9).

Object models use the concept of *multiplicity*, which is similar to the cardinality and connectivity of entities in an ER model. Multiplicity allows the designer to specify an association between an explicit number of any object instances with another object instance, such as *one-to-one*, *zero-to-many* or *2-to-3* (Figure 3.9). Object models also allow the naming of associations as well as the naming of roles of each object class in the association (e.g. Figure 3.10).

However, the main difference between these two models lies in modelling the behaviour of an object class. The ER or EER models are weak in capturing the behavioural aspect of objects. The object-oriented approach views classes (or types in EER) as a collection of methods. The EER model, on the other hand, is limited to regarding classes (supertypes/subtypes) as relationships among objects (entities), and ignores the dynamic behaviour of the object.

Another difference between EER models and object-oriented models is the representation of complex objects. Since object-oriented models emphasize behavioural abstraction, the accent is on the inheritance of encapsulated methods. Conceptually, the

encapsulation concept allows an object, in an object-oriented model, to store both attributes which themselves can be complex objects and methods together.

3.6.2 The PDMS Database Modelling Approach

For this initial design of the PDMS database, there is no considerable advantage in using an EER database modelling over an object-oriented one. Both methodologies provide easy constructs that convey the database schema in a clear and concise manner. They both provide methods that detect inconsistencies and discrepancies in the design. Moreover, both techniques allow classification and inheritance of data, which reduces data redundancy, as well as mapping techniques from the semantic design model to relational table model, since the PDMS database is being implemented on a relational platform.

However, if we consider the extension (future versions) and the reuse of the designed database model, an object-oriented design methodology has more to offer than an EER data modelling approach. The modelling and encapsulation of the object's dynamic behaviour, in an object-oriented method, do not only reduce data redundancy but also reduce the amount of code that interacts with the database by promoting code reuse. Moreover, encapsulation in the database design model yields code modules that are easy to implement and less expensive to maintain.

Another aspect to consider is that object-oriented analysis and design methods offer a more integral approach to system design. Most of these approaches contain three separate notions for modelling data, dynamics and process [Coad and Yourdon, 1991, 1991a] [Rumbaugh, 1991]. Such a method can be used not only as a modelling technique for the PDMS database, but as an integral method that can be used in the design of the entire system, which results in a more coherent system.

Recently, several famous designers in software development methods have produced object-oriented extensions to conventional systems analysis and design methods, the most noticeably of these being; object-oriented analysis (OOA) and object-oriented design (OOD) by [Shlaer and Mellor, 1988], by [Coad and Yourdon, 1991, 1991a] and by [Wiess

and Page-Jones, 1991], Object Modelling Technique (OMT) by Rumbaugh *et al.* [Rumbaugh *et al.*, 1991], OOD by Booch [Booch, 1991], object-oriented structured design (OOSD) by [Wasserman *et al.*, 1989]. Many others have also contributed. It is widely agreed that that these methods are all more or less incomplete and it should be noted that they are not so much methods as suggestions for methods. A comparison of object-oriented methods can be found in [Fichman and Kemerer, 1992] and [Graham, 1994].

In order to employ the most suitable object-oriented methods in the design of the PDMS database, the next section examines some of these suggestions or methods, all while focusing on the database design aspect of these methods. The methodology required should be able to satisfy, among others, the following principal three requirements:

1. To provide easy constructs for grouping, building and encapsulating object classes,
2. To provide mapping technique from an object model to a relational (table) model,
3. To, as closely as possible, result in a 3NF or higher relational model.

3.7 Object-Oriented Analysis Methods

In fact, there is a general problem in distinguishing object-oriented design methods from object-oriented analysis methods which is not the case for conventional methods. Therefore, we will present the database design aspect of these suggested methods rather than discuss the entire system design approach and methodology.

3.7.1 Shlaer/Mellor OOSA

One of the early object-oriented analysis examples is due to Shlaer and Mellor [Shlaer and Mellor, 1988], but this method could not really be considered as object-oriented for several reasons, mainly due to the absence of any inheritance notation. However, in a later book Shlaer and Mellor [Shlaer and Mellor, 1991] included inheritance in their data

modelling technique through entity subtyping as well as the idea that methods could be discovered by modelling the life cycles of entities with state transition diagrams.

The first step in Shlaer and Mellor's method is the definition of objects and their attributes. The entity modelling notation descended from the Ward/Mellor notation. The next step stresses the definition of object life histories by using Moore style state transition diagrams.

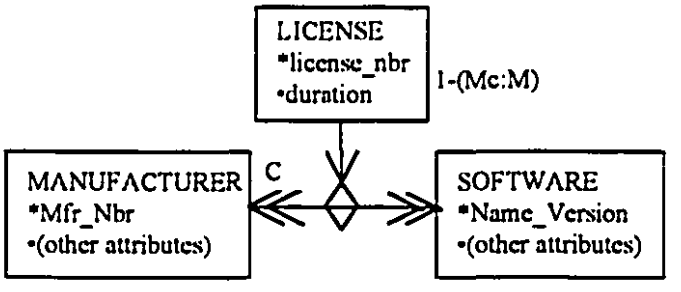


Figure 3.6: Shlaer/Mellor notation - ternary conditional relationship.

Shlaer and Mellor employ the same *n-ary* entity-relationships used in any extension to the ER model. However, they present another extension to the multiplicity and cardinality of the ER model to include the notion of conditional relationship. In an unconditional relationship every instance of each object participates; in a conditional relationship there may be instances of the object which do not participate in the established entity-relationship (Figure 3.6).

Shlaer and Mellor's methods is simple and easy to implement, it is strong in capturing the life cycle of an object. It is attractive to some designers because it introduced the idea of defining reusable domains. However, this method does not encapsulate the behaviour of the object in its data model; instead, the state of the object is defined in a life cycle diagram which depicts the methods that access the object, such as "get *object*" or "put *object*". The method is strongly influenced by relational design: objects are in the first Normal Form and object identity is not a natural feature of OOA. Shlaer and Mellor's data modelling technique can be viewed as an extension of the ER model and does not fit the profile of a complete object-oriented data modelling method.

3.7.2 Coad/Yourdon

Coad and Yourdon [Coad and Yourdon, 1990, 1991, 1991a] introduced a less clumsy notation than that found in Shlaer/Mellor or most of the object modelling techniques. They shifted the focus very much to analysis as opposed to design. In the second edition of their book, Coad and Yourdon responded to the criticism that they had failed to distinguish classes and instances by drawing a grey outline around the class notation to indicate its physical instance, if it has any instances (Figure 3.6) [Graham, 1994].

Defining object classes and attributes, in Coad/Yourdon's method, has the same consideration as any data modelling exercise. The database designer can take advantage of inheritance, and multiple inheritance of attributes and methods is notationally allowed in the second book edition. The second edition also made a distinction among three types of aggregation: part-whole, container-contents and collection-members (Figure 3.7). Such a specific relationship among classes may be confused with composite structures, there also may be other important structures required in a specific application. For example, some medical database might be concerned with a *kinship* relationship for inherited disease.

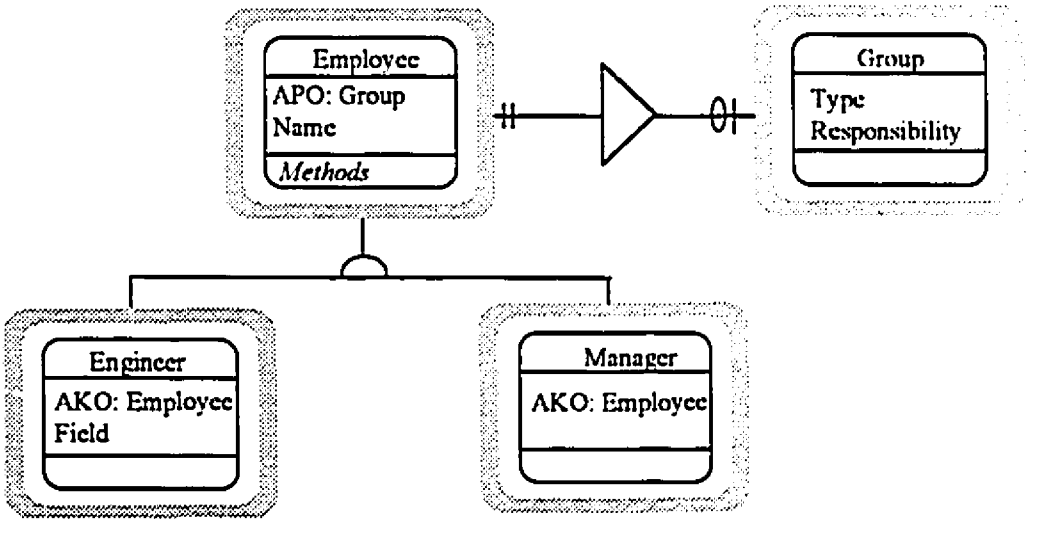


Figure 3.7: Inheritance and composition hierarchy in Coad/Yourdon notation. AKO= A Kind Of, APO = A Part Of.

Influenced by the relational model, the Coad/Yourdon approach insists that attributes should be *atomic*, which is contradictory to a main object-oriented concept, modelling

complex objects. They also discuss normalisation and the use of keys rather that object identity.

3.7.3 OMT

The Object Modelling Technique (OMT) is widely regarded as one of the most complete object-oriented systems analysis method published to date [Graham, 1994]. OMT was introduced by Rumbaugh and his colleagues at General Electric [Rumbaugh *et al.*, 1991]. It breaks down into three main phases or activities: analysis, system design and object design. OMT has strong roots in traditional structured methods and offers an extremely rich and detailed but complicated notation.

As mentioned at the beginning of this section we will concentrate on the object modelling aspect of the studied object-oreinted methods. The first step in OMT is to build the object model (OM), which consists of diagrams similar to those of Coad/Yourdon but contains a much richer notation. The notation is fundamentally that of an EER modelling with methods (operations) and other annotations added into the entity icon (Figure 3.8).

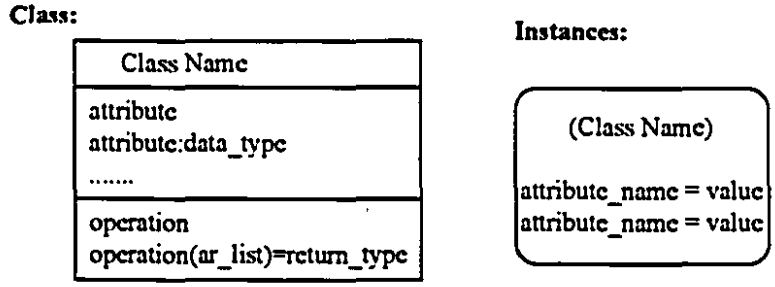


Figure 3.8: Classes and instances in OMT.

The basic data modelling notation of OMT is shown in Figures 3.8 to 3.12. Note in Figure 3.8 that attributes are typed and operations given argument lists and return types, and that instances have round corners and bracket names.

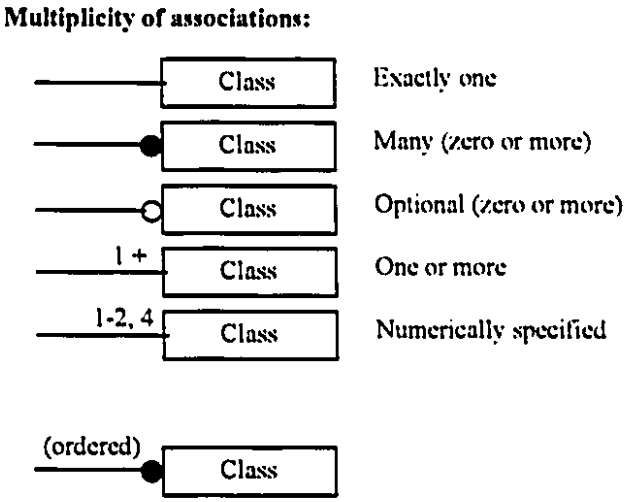


Figure 3.9: Associations in OMT.

Figure 3.10 shows that trenay associations are allowed but associations with attributes are often expanded into first-class objects. Associations are annotated with roles and may have a qualifier. The latter is occasionally useful.

Classification structures are shown in Figure 3.11, from which it can be deduced that the notation is richer than that of Coad/Yourdon in expressing exclusivity and optionality.

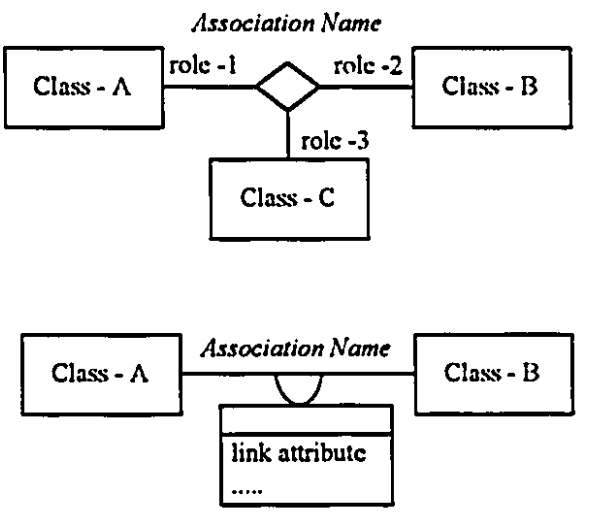


Figure 3.10: Ternary associations and associations with attributes in OMT.

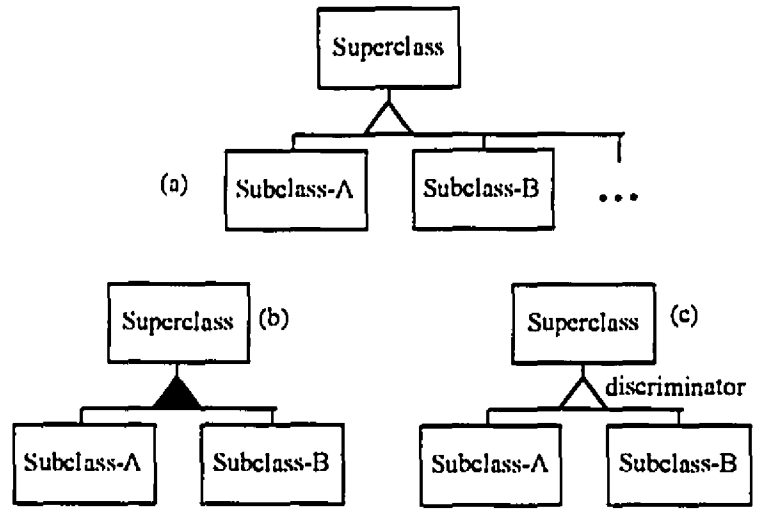


Figure 3.11: Classification in OMT.
(a) More subclasses exist. (b) Subclasses have overlapping (non-disjoint) membership.
(c) The discriminator is an attribute whose value discriminates between classes.

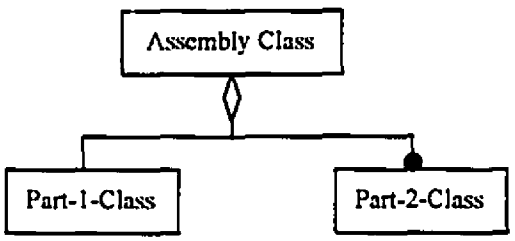


Figure 3.12: Aggregation (composition) in OMT.

Composition structures are shown in Figure 3.12. A particularly strong feature of OMT is the ability to represent recursive composite structures.

Rumbaugh *et al.* [Rumbaugh *et al.*, 1991] presents some heuristics for implementing object-oriented designs in relational databases and in conventional languages. The guidelines given for relational database implementation are as follows:

- Objects are identified using primary keys, preferably database engine-generated surrogates.
- Classes are tables and instance tuples.
- Associations are tables.
- Inheritance links are shared with a secondary index.

Generally, third-generation relational databases such as Ingres and Sybase give better facilities for object-oriented modelling; however, the above guidelines work for most database products.

OMT covers more issues than most other methods but it remains incomplete in some areas and it is very complex to learn and use the notations. The emphasis on state transition diagrams in OMT reflects its real-time modelling background. On the other hand, unlike some other methods that employ state transition diagrams such as Shlaer/Mellor, OMT allows to model the methods (operations) within its basic class notation according to the encapsulation concept.

3.7.4 OMT in the PDMS Database Design

Among the reviewed object-oriented analysis and design methods, OMT seems to satisfy the requirements presented in Section 3.6.1. The Rumbaugh *et al.* approach (OMT) uses two other types of models in addition to the object model in order to describe the design of an information management system. The first one is the dynamic model, which illustrates the dynamic and control aspect of the system such as the changing states of an object, using a state transition diagram (STD). The second type of model is the functional or process model which describes the transformation of values that occur during system execution, using data flow diagrams (DFD).

Such types of models are not found in the extensions of the ER model. They are not even found in some object-oriented system design methods. Triggering, or the propagation of operations from one object to another, is also defined in this methodology in the form of *object messages.*

Moreover, OMT was considered for the design of the PDMS database because it offered object-oriented modelling techniques for relational database implementations. This aspect renders the object model of the PDMS more flexible and extendible, while providing 3NF or higher realizable relational data tables.

Figure 4.1 illustrates the architecture proposed by the ANSI/SPARC committee on DBMS for a family of related database applications. The basic idea is that database design should comprise three layers: the external, conceptual and internal schemas. The external schema is an abstraction of the global conceptual model, it isolates applications from most changes in the conceptual model. The conceptual schema is a database design that integrates related applications and hides the peculiarities of the underlying DBMS. The conceptual schema contains what is known as *meta-data*. Meta-data is data that describes other data. For example, the definition of a class or a database table is meta-data. The internal schema deals with the limitations and features of a specific DBMS, it consists of actual code required to implement the conceptual schema or model.

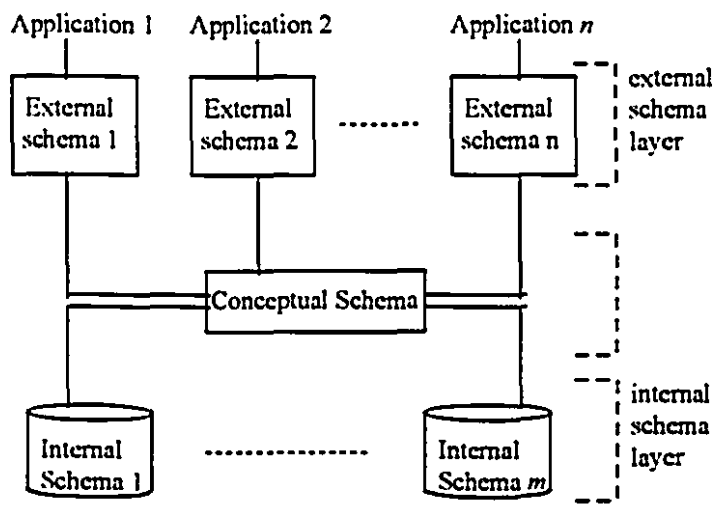


Figure 4.1: ANSI/SPARC three schema architecture.

The PDMS database design uses Rumbaugh's OMT for designing both the external and conceptual schemas, and OS/2's Database Manager specific code to create the internal schema. When used for a relational database implementation, OMT refers to these three layers of the database design as the *high*, *middle* and *low* levels. The initial high-level object model is successively converted into relational tables and then into the low level DBMS Data Definition Language (Figure 4.2).

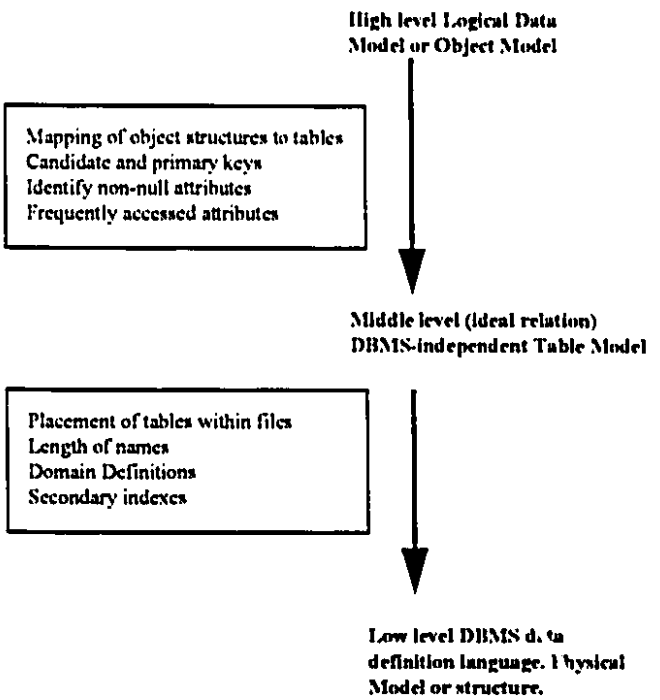


Figure 4.2: Design Levels.

Employing the OMT to design the PDMS database guarantees that the tables of the relational data model will be in the Third or Boyce-Codd Normal Form.

4.1 The PDMS Database Object Model

The first step in the design of the PDMS database is describing its high level object model. This step is also used to analyse the information or data which requires processing in the considered ICU. As suggested by the OMT, this phase will consist of grouping the similar data elements types, managed in the ICU, under respective object classes and then establishing the necessary or appropriate relationships types between these classes. For this prototype version of the PDMS, the object model is mainly composed of binary associations and aggregations between six different superclasses as illustrated in Figure 4.3, *Patient*, *Parameters*, *Admittance*, *Ingesta*, *Excreta* and *NCP*. The OMT graphical notations used in Figure 4.3 to depict the PDMS object model have already been explained in Section 3.7.3.

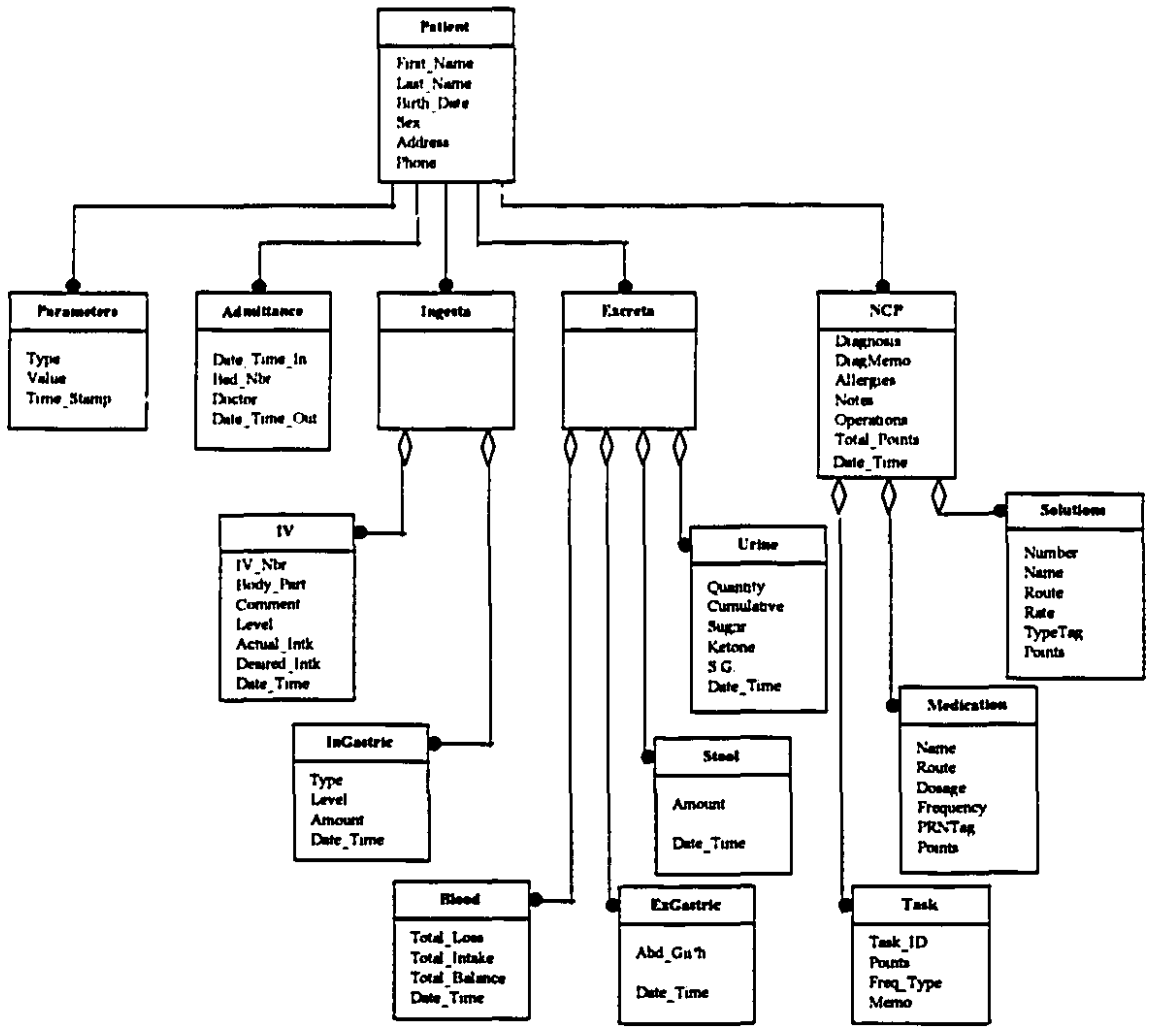


Figure 4.3: The PDMS Object Model.

Upon admission into the ICU of the Montreal Children's Hospital, the patient is registered into the system. The registration procedure consists of recording the name, sex, birth date, address and phone number of the *Patient*. Then, an *Admittance* record is created for that patient, which consists of the assigned bed number, the principal treating physician, the time and date of admission and eventually the time and date of discharge. Since a patient may be admitted more than once into the ICU an one-to-many association exists between a single patient and the corresponding admittance records, depending on the existence of the *Patient*.

During his or her stay in the ICU, the patient will be monitored by electronic equipment. This equipment generates electronic data such as the patient's heart beat rate

and respiration rate. It also measures and generates data for about fifty other vital sign parameters. The set of electronic data attributes is identified in the system by the *Parameters* object class. The PDMS has a requirement for storing all this electronic data in a real-time fashion, since this data needs to be plotted in real-time graphical charts. It also needs to be analysed and monitored by an expert system module that detects and warns about any alarming data value [Collet, 1990]. Since these may be up to one parameter recording per second related to the patient, there exists an one-to-many association between the *Patient* and the *Parameters* object classes, depending on the existence of the *Patient*.

Another set of data that is gathered manually in the ICU is the volume of all fluids extracted or injected from and into the patient's body. This data set is identified in the system as the *Fluid Balance* object class. Since the fluid balance is the recording of all injected and extracted fluids, we have divided the fluid balance object class into two object classes, *Ingesta* and *Excreta*. Since the patient may be associated with more than one Fluid Balance sheet, the relationship between the *Patient* and the *Ingesta* and *Excreta* object classes is an one-to-many association, depending on the existence of the *Patient*.

The *Ingesta* class defines the volume data of all injected fluids and is composed of many instances of two subclasses: *IV* (Intravenous) and *InGastric* object classes. The *IV* object class groups the data attributes of all the fluids injected into the patient's body in the form of an IV. The *InGastric* object class groups the data attributes of all the medication injected into the patient's body in the form of a gastric fluid. Hence, there exists a one-to-many aggregation relationship between the *Ingesta* and the *IV* classes, as well as between the *Ingesta* and *InGastric* classes.

The *Excreta* class defines the volume data of all fluids extracted from the patient's body and is composed of many instances of four subclasses: *Blood*, *Urine*, *ExGastric* and *Stool* object classes. The *Blood* object class groups the data attributes of all the blood balance in the patient's body. The *Urine* object class groups the data attributes of the quantity of urine that was released from the patient's body at a certain time and date as well as the cumulative quantity to date and time. It also groups the attributes that define the

characteristics of the urine such as the amounts of sugar and S.G. that were measured. The *ExGastric* object class groups the data attributes of the measured abdominal girth of the patient. The *Stool* object class groups the data attributes of the patient's stool. Hence, there exists a one-to-many aggregation between the *Excreta* class and all its subclasses.

Each patient in the ICU is associated with a *Nursing Care Plan*. This plan may vary according to the patient's condition and the treatment requirements. The data attributes set that compose the *Nursing Care Plan* is identified in the system as the *Nursing Care Plan* (*NCP*) object class. Since the *Nursing Care Plan* varies over time, and since more than one plan could exist for a single patient, there exists an one-to-many association between the *Patient* and the *NCP* object class, depending on the existence of the *Patient*.

The *NCP* class is composed of many instances of the following sub-classes: *Task*, *Medication* and *Solutions*. The *Task* object class represents the information about the tasks that the nursing staff has to perform. Each task is identified by a *Task Number* or Identifier and weighs a certain number of points. These points are used to measure the global load or effort associated with a certain care plan, it is also used to measure the individual nursing staff load. The *Medication* class represents the information about the type of medication that should be administered with this care plan, as well as its frequency, volume and means of administration. The *Solutions* class represents the information about the solutions that should be administered with this care plan as well as their name, type, frequency, volume and means of administration.

4.2 The PDMS Database Table Model

The middle level table model contains generic, DBMS-independent tables. The motivation for the middle level is to decouple the general problem of mapping objects to tables from the specifics of each DBMS. The middle level or the table model is wordier and less effective at conveying the overall structure of the database model than the high level. However, using the mapping notations suggested by Rumbaugh *et al.*, the middle level conveys more details, and facilitates the generation of the Data Definition Language in

order to physically implement the database. This improves documentation and eases design portability.

In order to map the object model to the table model, we can choose among several mapping alternatives. For example, there are two ways to map an association to tables and four ways to map a generalisation. However, since the relationships in the PDMS object model are all associations or aggregations, Section 4.2.2 will concentrate on the procedure which maps associations from the object model to the table model.

The middle level is one step closer to the physical implementation of the database than the high level object model, it conveys more specific detail about the attributes of each class and the domains to which each attribute belongs. Such details aid the programmer to develop code that creates the database tables, even without any understanding of the global system model.

The low level is the data definition of the target DBMS. This level contains the actual DBMS commands that create the tables, attributes, and indexes. The low-level considers DBMS specifics such as location of the tables within the database, and choice of performance tuning mechanisms. It deals with the arbitrary restrictions such as size limitations.

The following two sections will present a general procedure or algorithm which is employed in mapping the object model of the PDMS into the table model. The mapping procedure will be detailed in a step-wise fashion. Moreover, these sections will also explain the graphical representation of the middle level table model.

4.2.1 Mapping Object Classes to Tables

STEP 1: *For each class C in the object model, we create a relation R which includes all the attributes of C.*

STEP 2: *If the designer chose to identify the object by an ID, then the ID must become a non-null attribute of R and its primary key (for example, "Patient.Patient_ID" in Figure 4.3). Otherwise, the key attributes of C become the primary key for R.*

There are benefits for using IDs. They are immutable and completely independent of changes in the data value and physical location. The stability of object IDs is particularly important for associations since they refer to the object. This can be contrasted with the referring of objects by name, where changing a name requires the update of many associations. IDs provide a uniform mechanism for referencing objects.

On the other hand, IDs have disadvantages. Generating IDs is a nuisance, for which RDBMSs provide no inherent support. Therefore, in the design of the PDMS object model, we were careful as to which class should benefit of an object identifier. This decision was based on the type of association between two classes, as will be described in Section 4.2.2.

Graphically, a class is usually portrayed by a table (Figure 4.4), whose title is the name of the class which eventually will become the name of the database table. The table contains five columns that list the following:

- The first column defines the name of each attribute defined in this class.
- The second column defines whether this attribute can contain a null value in the database; i.e., is it mandatory for this attribute to contain a value when inserted in the database?
- The third column indicates whether this value should be *unique* in the database. This is used in order to preserve the *referential integrity* (see Section 4.4.1) in the database.
- The fourth column defines the domain for which this attribute belongs; for example, an attribute can be an integer, a character, etc.
- The last column indicates the size in bytes of that attribute. If the columns contains the value "dflt", this indicates that the programmer may use the default database system type size for the considered attribute.

The table model also indicates, next to each table, the candidate keys and the selected primary key of that table (see Section 4.4.1), as well as its frequently accessed attributes. It is also suggested that table indexes should be defined at this design level. Indexes are typically used to speed up the access to a table. If an index is used by the Database

Manager to process a query, specific rows can be located faster with an index scan than a table scan. Index files generally are small and require less time to read than an entire table.

However, the use of indexes has disadvantages. The various access paths that the Database Manager can choose suggests that indexes reduce access time significantly. Each index takes up a certain amount of storage depending on the size of the table and the number of attributes included in the index. Hence, in the case of the PDMS database model each insert operation performed on a table requires additional updating of each index on that table. And since the PDMS database model requires intensive insert operations, especially in the DLC module, the system performance can deteriorate.

A better solution would be to define primary keys wherever they apply. Since each primary key has a unique index it reduces the overhead of maintaining and managing extra index tables or files. This solution proves to be ideal in the case of the PDMS database. Since the candidate index on all the data model tables is the *Patient_ID* attribute, which is also part of all the primary table keys. Therefore, we relied on primary keys to perform the table indexing.

4.2.2 Mapping Binary Associations to Tables

In general, an association may, or may not, be mapped to a table. It depends on the type and multiplicity of the association and the database designer's preferences in terms of extensibility, number of tables, and performance trade-offs.

A many-to-many association always maps to a distinct table. This schema satisfies Third Normal Form. On the other hand, and since we are mainly concerned with one-to-many associations, there are two options of mapping one-to-many associations to tables. We may create a distinct table for each association or bury a foreign key in the table of the *many* class. The advantages of merging an association into a class are:

- Fewer tables.
- Faster database performance due to the fact of navigating less tables.

The disadvantages to that approach are:

- More complexity; an asymmetrical representation of the association complicates search and updates.
- Less design rigor; associations are between independent objects of equal syntactic weight. It seems inappropriate to contaminate objects with knowledge about other objects.

However, since in our case real-time response is a crucial design element and since the aforementioned disadvantages can be carefully resolved in the application module surrounding the database, in other words the behaviour of the object class, we opted for mapping one-to-many associations as foreign keys in the table of the *many* class. Another factor that played a role in that decision is the fact that all associations in the PDMS object model are one-to-many. Hence, if we translate them all in the same manner, we promote consistency throughout the database design and, in particular, in the modules that access each table, which reduces the complexity of search and updates. This was implemented by making sure that the buried key should become a non-null attribute of the primary key of the *many* class.

STEP 3: *For each one-to-many association with existence dependency between a one class C_1 and a many class C_m, the primary key K_1 of the relation R_1 (table of C_1) should be added as an attribute of R_m (table of C_m).*

STEP 3a: *K_1 should not be null and should be an attribute of the primary key of R_m.*

STEP 4: *For each one-to-many aggregation repeat STEP 3 and STEP 3a.*

STEP 4a: *Make sure that no attributes of R_1 are replicated in R_m except the primary key of R_1.*

STEP 4b: *Make sure that no attributes of R_m are replicated in R_1.*

As aforementioned an aggregation is a special type of association. But since in an aggregation relationship the subclasses compose the abstract notion of the assembly object class, and in order to ensure design integrity, the attributes between the superclass and the subclass should not replicated, except when mapping the relationship from the object

model to the table model (step 4a). In this context, an aggregation allows the inheritance of attributes between the "super" and the "sub" classes. Hence, steps 4a and 4b enforce these design guidelines.

4.3 Testing for 3NF and BCNF

Normalisation of data can be viewed as a process in which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties. Hence, Normal Forms can be considered as tools that provide the database designer with a series of tests that can be carried on individual schemas so that the relational database can be normalised to a desired degree. When a test fails, the relation violating that test must be decomposed into relations that meet the normalisation test [Elmasri and Navathe, 1995]. Date [Date, 1986, pp. 390-391] has formulated the successive decomposition process as a set of rules.

The higher levels of normalisation are important for database designers because they make the structure of the database easier to understand and they reduce data anomalies. Hence, a database schema design in 3NF and higher is considered to be a "good design". Normalisation beyond 3NF is rarely done in practice. Although BCNF and 4NF are less rare then 5NF, they are still highly theoretical in nature [Mittra, 1991]. Usually, a relation in First and Second Normal Form exhibits certain anomalies with respect to insertion, deletion or updates of tuples, whereas a relation in Third Normal Form does not have such problems. Consequently, a relation in 3NF or higher is preferred.

The goal of the design algorithm presented in this section is to test whether each individual relation in the PDMS table model is in 3NF or BCNF. Otherwise, if a table fails to be in 3NF or BCNF, we will discuss the alternative procedures available to amend this situation as well as their design implications. However, before discussing the algorithm, we should present some concepts or terminology that will be extensively used throughout this exercise.

4.3.1 Referential Integrity Constraint

In general, a relation or a relation schema may have more than one key, and each of these keys is referred to as a candidate key. It is common to designate one of the *candidate keys* as the primary key. The *entity integrity constraint* dictates that a non-primary key can be null, but a primary key value cannot. This is because the primary value is used to identify individual tuples in a relation.

An attribute is called a *prime attribute* of a relation if it is a member of any candidate key or the primary key of that relation. An attribute is called a *nonprime* attribute if is not a prime attribute [Elmasri and Navathe, 1995].

The *referential integrity constraint* is specified between two relations and is used to maintain the consistency among two tuples of two relations. For example, in Figure 4.4, the attribute *Patient_ID* in the *Admittance* relation is used as a foreign key part of the primary, in order to maintain a consistency between a tuple from the *Patient* relation and a corresponding tuple in the *Admittance* relation.

4.3.2 Functional Dependencies

Consider two sets of attributes X and Y that are subsets of R, R being the single *universal* relation which contains all the attributes in the database, $R=\{A_1, \ldots, A_n\}$. A functional dependency between X and Y, often denoted $X \rightarrow Y$, specifies a constraint on the possible tuples that can form a relation instance *r* of R. That constraint states that for any two tuples t_1 and t_2 in *r* such that t_1 of X equals t_2 of X, t_1 of Y must also equal t_2 of Y, denoted $t_1[X]=t_2[X] \Rightarrow t_1[Y]=t_2[Y]$. This means that the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. The term Functional Dependency is often abbreviated by FD and the set X is referred to as *the left-hand side* of the FD, while Y is the *right-hand side* [Elmasri and Navathe, 1995].

In other words, X is a candidate key of R, if there exists a constraint on R which states that there cannot be more than one tuple with a given X value in any instance r of R, which implies that $X \rightarrow Y$ for any subset of attributes Y of R. Note that if $X \rightarrow Y$, this does not necessarily imply that $Y \rightarrow X$.

4.3.3 Normal Forms Based on Primary Keys

Since the table model of the PDMS is based on primary keys, as described in Section 4.2.1, the definitions of the Second and Third Normal Form presented here will be based on the FD and primary keys of a relation.

As aforementioned in Section 3.3 a relation is in 1NF if the domains of all its attributes include only atomic values. In the proposed database design, we begin testing for Normal Forms by observing the fact that all the relations of the PDMS table model are in 1NF.

Elmasri and Navathe [Elmasri and Navathe, 1995] provide a definition of 2NF based on the concept of full functional dependency. A FD $X \rightarrow Y$ is a *full functional dependency* if the removal of any attribute A from X means that the dependency does not hold any more. A FD $X \rightarrow Y$ is a partial dependency if for some attribute $A \in X$, $(X - \{A\}) \rightarrow Y$. A relation is 2NF if every nonprime attribute A is fully functionally dependent on the primary key of that relation.

If a relation schema R is not in 2NF, R can be further normalised by decomposing it into a number of 2NF relations where nonprime attributes are associated only with the part of the primary key on which they are fully dependent. In any case, 1NF and 2NF are not considered good relational designs. As we mentioned in chapter 3, it is best to have relation schemas in BCNF. However, if that is not possible, 3NF is acceptable.

[Elmasri and Navathe, 1995] also provides a definition of 3NF based on transitive dependency. A FD $X \rightarrow Y$ in a relation is called *transitive* if there is a set of attributes Z which is not a subset of the primary key or does not include the primary key of that relation, and both $X \rightarrow Z$ and $Z \rightarrow Y$ hold. According to Codd's original definition, a relation is in 3NF if it is in 2NF and none of its attributes are transitively dependent on the primary key.

Boyce-Codd normal from is stricter than 3NF, every relation that is in BCNF is already in 3NF. However, the opposite is not necessarily true. A relation R is in BCNF if it is already in 3NF and whenever a FD $X \rightarrow A$ holds in R, where A is an attribute of R, then X

is a candidate key of R. Accordingly, if a FD X→A holds in R where X is not a candidate key and A is a prime attribute, then R is 3NF rather than BCNF.

4.3.4 Normal Form Testing Algorithm

The testing algorithm for Normal Form presented in this section will be applied to the table model of the PDMS described in Sections 4.4 through 4.11. However, in order to reduce repetition, the details of applying the NF test algorithm will only be considered for three sample tables, namely the *Patient*, *IV* and *Blood* tables. For the remaining relations, we will simply state the NF test algorithm outcome.

Assuming that the designer has already verified that all attribute domains of the PDMS table model are atomic, the algorithm will not test for 1NF.

Testing for 2NF and 3NF:

STEP 1: *If the primary key of the considered relation R is composed of a single attribute, then R is in 2NF and jump to STEP 4, because every nonprime key is evidently fully functionally dependent on the primary key.*

STEP 2: *Otherwise, let* $B=\{A_1, \ldots, A_j\}$ *be the set of nonprime attributes of R and let* $X=\{A_1, \ldots, A_n\}$ *be the primary key of R. If there exits an* $A_i \in B$ *such that the FD* $(X-A_k) \rightarrow A_i$ *holds, where* $A_k \in X$*, then stop since R is not in 2NF. Otherwise continue (STEP 3).*

STEP 3: *Do STEP 2 for every attribute Ai* $\in B$*, if STEP 2 passes for every nonprime attribute then R is in 2NF. At the end of STEP 3, if R is in 2NF then continue to STEP 4 which tests for 3NF.*

STEP 4: *Let* $G(A_i)$ *be the set of all the left-hand sides of all FDs* $Z_i \rightarrow A_i$ *in R,* $G(A_i)=\{Z_1, \ldots, Z_n\}$*. If there exists a* Z_i *such that* $Z_i \not\subset X$ *and* $X \not\subset X_i$*, and the FDs* $X \rightarrow Z_i$ *and* $Z_i \rightarrow A_i$ *hold, then R is not in 3NF. Otherwise, continue testing (STEP 5).*

STEP 5: *Do STEP 4 for every nonprime attribute Ai ∈ B, if STEP 4 fails for any Ai ∈ B, then R is not in 3NF and STOP. Otherwise, R is in 3NF and continue testing for BCNF (STEP 6).*

Testing for BCNF:

STEP 6: *Let A_i be an attribute of the relation R, $A_i \in R=\{A_1, \ldots, A_n\}$. Let $G(A_i)$ be the set of all the left-hand sides of all FD $Z_i \rightarrow A_i$ in R, $G(A_i)=\{Z_1, \ldots, Z_n\}$. If every $Zi \in G(A_i)$ is a candidate key of R continue (STEP 7), otherwise stop since R is not in BCNF.*

STEP 7: *Do STEP 6 for every attribute $A_i \in R$, if STEP 6 fails for any $A_i \in R$, then R is not in BCNF and STOP. Otherwise, R is in BCNF.*

The following sections will describe the middle level table model of each of the PDMS modules, using the aforementioned methodology and test the resulting table model for BCNF and 3NF. The testing exercise is also an opportunity to verify that indeed employing the OMT for the design of the PDMS database will result in 3NF, or above, relation schemas. First, we will describe the tables managed in the *registration* module and test them for Normal Forms. Then, we will follow the same presentation approach for the *fluid balance* module, the *nursing care plan* module and the *data link controller* modules.

4.4 Registration Tables

The registration module manages two types of information; the first class is general information about the patient such as name, sex, date of birth, address, etc. The second class of information describes admittance information about the patient such as the date and time the patient was admitted into the ICU, the date and time the patient was released from the ICU, the patient's bed number, etc.

The data represented in the first class *Patient* (general patient information) is used by all the other modules of the PDMS. Therefore this class has been moved to the highest level of hierarchy in the object model, Figure 4.3. Hence, the data of this class which is specific for each patient can be inherited by all the modules rather than being replicated and copied throughout the tables of the PDMS modules.

Figure 4.4 shows the mapping between the high level information model and the middle level information model of the *Patient* class:

- *Patient.Patient_ID*: indicates the ID number given to the patient. Domain: Serial number. This object identifier should contain a *unique not null* value.
- *Patient.First_Name*: indicates the first name of the patient. Domain: Long name, String of characters.
- *Patient.Last_Name*: indicates the last name of the patient. Domain: Long name, String of characters.
- *Patient.Birth_Date*: indicates the birth date of the patient. Domain: Date, formatted as such "DD-MM-YYYY".
- *Patient.Sex*: indicates the sex of the patient. Domain: Set of two single characters (M, F).
- *Patient.Address*: indicates the address of the patient's residence and/or any other addresses where the patient could be contacted. Domain: String of characters.
- *Patient.Phone*: indicates a telephone number where the patient can be reached. Domain: Integer, Number, e.g. 5143989394.

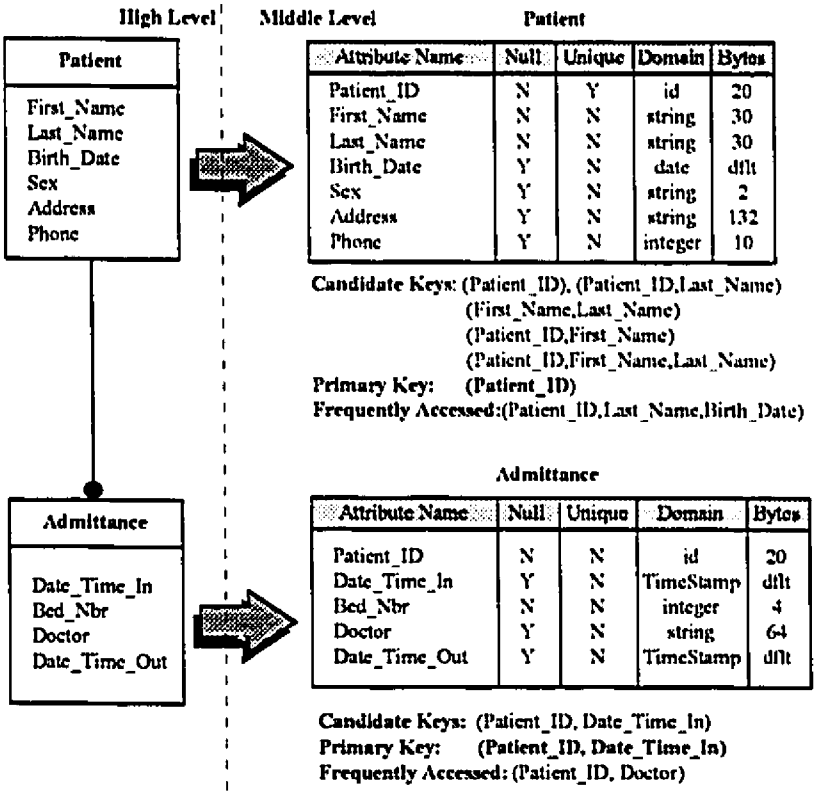


Figure 4.4: Patient & Admittance Data Classes.

The *Patient* class is accessed by application modules that can only perform insert, update and retrieve data operations on it, based on its primary table key. Therefore, in the case where the patient changes phone numbers the staff can update the patient's record in the *Patient* table.

The second class of data, *Admittance*, used in the Registration Module has administrative as well as historical purposes. The data represented in this class reveals information such as the date the patient was admitted into the ICU, the date the patient was released from the ICU, the diagnosis, and the name of the treating physician. This information serves as a patient history, and could also be used by the accounting department.

Figure 4.4 shows the mapping between the high level information model and the middle level information model of the *Admittance* class. Since there is a one-to-many association between the *Patient* class and the *Admittance*, the key attribute of the *Patient* class, *Patient_ID*, has been inserted as a foreign key and used as a part of the primary key of the

Admittance class when applying the mapping algorithm between the object model and the table model. This serves well in choosing the primary key of the *Admittance* relation. Since a patient may be admitted more than once into the ICU, the primary key should be able to identify which admittance is in question: the first, the second or the "Nth" one. Therefore, the primary key of the *Admittance* relation is a combination of the *Patient_ID* and time and date of admittance attributes (Figure 4.4):

- *Admittance.Patient_ID*: indicates the same value as *Patient.Patient_ID*. The purpose of duplicating this attribute in the *Admittance* class is to establish and maintain the relationship as described in the database object model. Domain: Serial number, not *unique* in this table in order to allow multiple entries or records of admittances for the same patient which satisfies the one-to-many association relationship between the *Patient* and the *Admittance* class.
- *Admittance.Date_Time_In*: indicates the time and date the patient was admitted into the ICU. Domain: TimeStamp.
- *Admittance.Bed_Nbr*: indicates the current bed number assigned to the patient in the ICU upon admittance; this value may change if the patient is re-assigned to another bed. Domain: Integer, Number.
- *Admittance.Doctor*: represents the name of the primary physician who is responsible for the patient's treatment. Domain: String.
- *Admittance.Date_Time_Out*: indicates the time and date the patient was discharged from the ICU. Domain: TimeStamp.

The application modules using the *Admittance* table are designed to perform record insertion, update and retrieval according to the primary key of the table. Thus, if the patient's physician changes, the staff can update that attribute in the patient admittance record.

4.4.1 Registration Tables NF Testing

Applying the NF testing algorithm of Section 4.3.4, we will begin with testing the *Patient* relation schema for BCNF, R_P={Patient_ID, First_Name, Last_Name, Birth_Date, Sex, Address, Phone}. The set B_P of nonprime attributes of R_P is {Birth_Date, Sex, Address, Phone}.

STEP 1

Since the primary key of R_P is composed of a single attribute, R_P is in 2NF. Next we will verify whether R_P is in 3NF.

STEPS 4 and 5

The attributes that belong to B_P are not functionally interdependent; that is, they are all functionally dependent on the patient. Moreover, we cannot say that any attribute in B_P is functionally dependent on *Last_Name*, *First_Name* or the combination of both for the following reasons:

- *G(Birth_Date)* = {Patient_ID, (Patient_ID, Last_Name), (Patient_ID, First_Name), (Patient_ID, First_Name, Last_Name)}.

  More than one patient can have the same name but different birth dates. Similarly two people born on the same date do not necessarily have the same name.

  In both cases, $t_1[X]=t_2[X]$ does not imply $t_1[Y]=t_2[Y]$ (see Section 4.3.2), hence we don't have functional dependencies between *Birth_date* and any other attribute than the primary Key.

- *G(Sex) = G(Birth_Date).*

  More than one patient can have the same name (first and last or both) but different sex, therefore $t_1[X]=t_2[X]$ does not imply $t_1[Y]=t_2[Y]$.

- *G(Address) = G(Phone) = G(Birth_Date).*

  It is not rare that two people with the same name live under the same civic address and use the same phone number. This case is common with children that

have similar names as their parents, such as *Joe Doe* and *Joe Doe Jr.* Hence, the *Address* and *Phone* attributes cannot functionally depend on {Last_Name, First_Name}. Similarly, we cannot be sure that {Address, Phone} → {Last_Name, First_Name}.

Hence, R_P is 2NF and has no transitive dependencies. Thus it is in 3NF. Now, we should continue with the algorithm and test R_P for BCNF.

STEPS 6 and 7

- *G(First_Name)* = {Patient_ID, (Patient_ID, Last_Name)}.

  The patient's first name is functionally dependent on the patient's ID or on the combination of the patient's ID and the patient's last name. The *First_Name* cannot be functionally dependent on the *Last_Name* because we may have more than one patient with the same last name.

  Hence, in this case, $t_1[X]=t_2[X]$ does not imply $t_1[Y]=t_2[Y]$, which contradicts the concept of functional dependency (see Section 4.3.2). Similarly, the *Last_Name* cannot be functionally dependent on the *First_Name*.

  Examining the table model of the Patient table, we notice that every left-hand side that belongs to G(First_Name) is a candidate key. Therefore we continue the algorithm by building G(Last_Name).

- *G(Last_Name)* = {Patient_ID, (Patient_ID, First_Name)}.

  Since every $X_i \in$ G(Last_Name) is a candidate key we will continue.

- *G(Birth_Date)* = {Patient_ID, (Patient_ID, Last_Name), (Patient_ID, First_Name), (Patient_ID, First_Name, Last_Name)}.

  The birth date is functionally dependent on the patient. Every $X_i \in$ G(Birth_Date) is a candidate key, hence we will continue.

- *G(Sex) = G(Address) = G(Phone) = G(Birth_Date).*

Obviously the sex, address and phone number are functionally dependent on the patient. Since every Xi ∈ G(Sex) ∈ G(Address) ∈ G(Phone) ∈ G(Birth_Date) is a candidate key, the algorithm finishes "successfully". Therefore, the *Patient* relation is in BCNF, and we will not test this relation schema for any other Normal Form.

Applying the same algorithm on the *Admittance* table, we can deduce that it is also in BCNF.

4.5 Fluid Balance Tables

The Fluid Balance Module manages information concerned with the volumes of all fluids injected into and extracted from the patient's body.

Data in the Fluid Balance Module is captured manually or through a voice activated user interface [Petroni, 1991], in a spread-sheet like format. The Fluid Balance Module spread-sheet management program calculates, balances and corrects all the entered volumes data before it commits them to the database.

Data in the Fluid Balance Module is divided into two major classes, *Ingesta* and *Excreta*. Theses classes are considered *abstract* classes. An *abstract* class is a class that has no direct instance but whose descendent classes have direct instances. On the contrary, a *concrete* class is a class that is instantiable or one that can have direct instances. Designs frequently use abstract classes in order to convey subordinate classes that participate in the same association or aggregation. Some abstract classes appear naturally in the application, such as the *Ingesta* and the *Excreta* classes.

Being abstract classes, the *Ingesta* and *Excreta* classes will not be mapped onto a database table in the middle level layer of the design. Moreover, the association relationship between the *Patient* classes and both the *Excreta* and *Ingesta* classes will be translated onto, or inherited by, the descendent classes of the laters. Therefore, in the middle level mapping, the aggregation between the *Ingesta* class and its descendent classes will be converted into association relationships between the *Patient* class and those

Ingesta descendent classes. Similarly, the relationships between the *Patient* class and the *Excreta* class will be translated into association relationships between the *Patient* class and the descendent classes of the later. Those relationship translations are illustrated in Figures 4.5, 4.6, and 4.7.

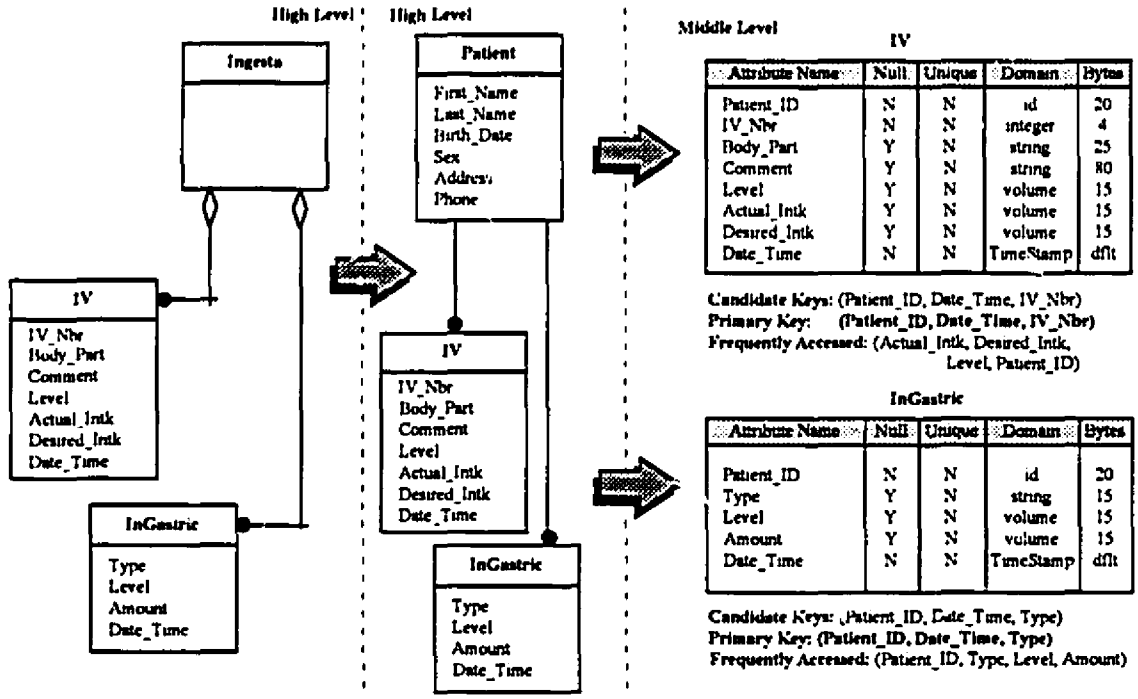


Figure 4.5: Ingesta - IV & InGastric Classes.

Figure 4.5 illustrates the application of the mapping algorithm between the object model and the table model of the *IV* class:

- *IV.Patient_ID*: indicates the same value as *Patient.Patient_ID*. The purpose of duplicating this attribute in the *IV* class is to establish and maintain the inheritance relationship as described in the database object model. Domain: Serial number, this attribute is not defined as *unique* in order to allow the insertion of multiple Ingesta-IV records for the same patient. This satisfies the *one-to-many* association relationship between the *Patient* and the *Fluid Balance* object classes, since a single patient can have many fluid balance sheets during his stay in intensive care.
- *IV.IV_Nbr*: indicates the number of the IV given to that patient on a certain day. Domain: Integer.

- *IV.Body_Part*: indicates the part of the patient's body where the IV is injected. For example, left arm or left leg. Domain: String of characters.
- *IV.Level*: represents the level of the IV in millilitres at a given time. Domain: Volume expressed as string of characters.
- *IV.Actual_Intk*: indicates the actual volume of IV injected in the patient's body. Domain: Volume expressed as string of characters.
- *IV.Desired_Intk*: indicates the desired volume of IV that is supposed to be injected in the patient's body. Domain: Volume expressed as string of characters.
- *IV.Date_Time*: indicates the time and date the injection and measurements operations occurred. Domain: Time Stamp.

Applying the mapping algorithm between the object model and the table model, Figure 4.5 illustrates the middle level table model of the *InGastric* class:

- *InGastric.Patient_ID*: indicates the same value as *Patient.Patient_ID*. The purpose of duplicating this attribute in the *InGastric* class is to establish and maintain the relationship as it was described in the database object model. Domain: Serial number, this attribute is not defined as *unique* in order to allow the insertion of multiple Ingesta - InGastric records for the same patient. This respects the one-to-many relationship between the *Patient* class and the *InGastric* class.
- *InGastric.Type*: indicates the type of the gastric fluid injected into the patient's body. Domain: String of characters.
- *InGastric.Level*: indicates the level of all gastric fluids injected into the patient's body at a given time. Domain: Volume expressed as string of characters.
- *InGastric.Amount*: indicates the amount of gastric fluid injected into the patient's body at a given time. Domain: Volume expressed as string of characters.
- *InGastric.Date_time*: indicates the date and time the injection and measurements operations occurred. Domain: Time Stamp.

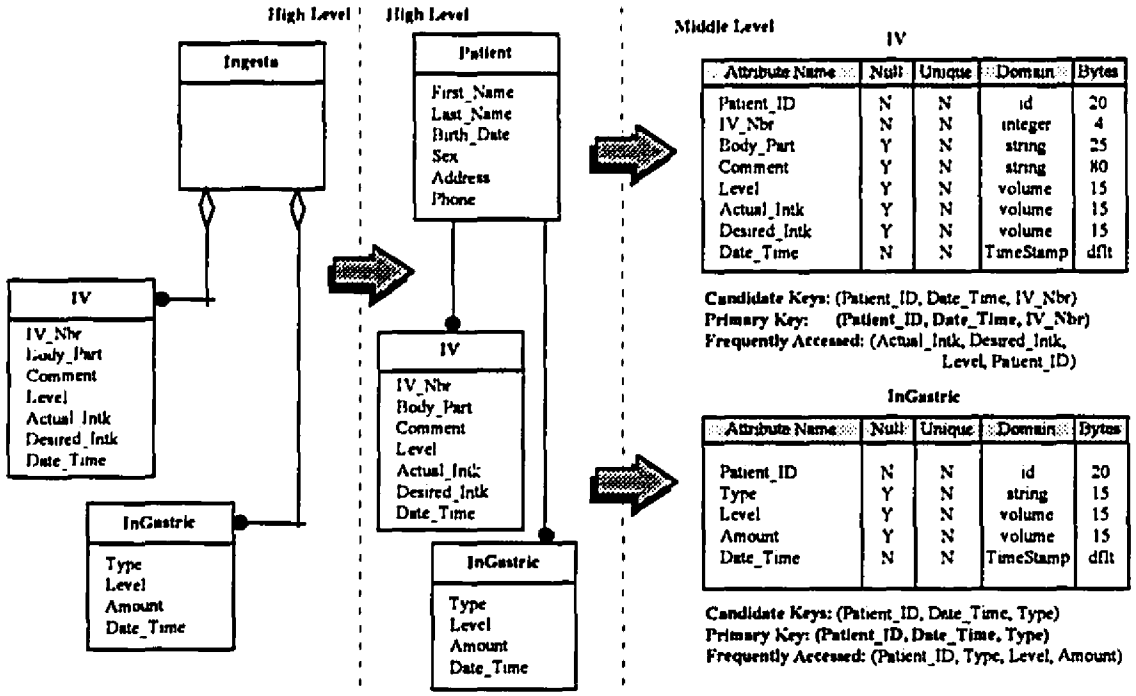


Figure 4.6: Blood & Urine Classes.

Figure 4.6 illustrates the mapping between the high level information model and the middle level information model of the *Blood* object class, again using the mapping algorithm between these two levels of the database design:

- *Blood.Patient_ID*: indicates the same value as *Patient.Patient_ID*. This attribute is also duplicated in the *Blood* object class in order to establish and maintain the relationship between the *Patient* and the *Blood* classes, as it is described in the database object model. Domain: Serial number, this attribute is not defined as *unique* in this table, in order to allow multiple insertions of *Excreta - Blood* records for the same patient.
- *Blood.Total_Loss*: indicates the total volume of the blood that has been lost from the patient's body at a certain time and date. Domain: Volume expressed as a string of characters.
- *Blood.Total_Intake*: indicates the total volume of blood that was injected into the patient's body at a certain time and date. Domain: Volume expressed as a string of characters.

- *Blood.Total_Balance*: mainly represents the difference between the Total_Intake and the Total_Loss attributes, and indicates the balance of blood in the patient's body at a certain time and date. Domain: Volume expressed as a string of characters.
- *Blood.Date_Time*: indicates the time and date the aforementioned blood volume measurements took place. Domain: Time Stamp.

Figure 4.6 also illustrates the mapping between the high level object model and the middle level table model of the *Urine* object class:

- *Urine.Patient_ID*: indicates the same value as *Patient.Patient_ID*. Similar to the other classes, this attribute establishes the relationship between the *Urine* class and the *Patient* class as it is described in the database object model. Domain: Serial number, this attribute is not defined as *unique* in this table in order to allow multiple insertions of records that pertain to the same patient.
- *Urine.Quantity*: indicates the quantity of the patient's urine at a certain time and date. Domain: Volume expressed as string.
- *Urine.Cumulative*: indicates the cumulative quantity of the patient's urine at the particular time and date. Domain: Volume expressed as string.
- *Urine.Sugar*: indicates the quantity sugar found in the patient's urine at a certain time and date. Domain: Volume expressed as string.
- *Urine.S_G*: indicates the quantity of S.G. that was found in the patient's urine at a certain time and date. Domain: Volume expressed as string.
- *Urine.Ketone*: indicates the volume of a ketone if found in the patient's urine at a certain time and date. Domain: Volume expressed as string.
- *Urine.Date_Time*: indicates the time and date when all the above measurements were recorded. Domain: Time Stamp.

The *Stool* object class groups the data attributes of the patient's stool at a certain time and date. The *ExGastric* object class groups the data attributes of the patient's abdominal girth at a certain time and date. Figure 4.7 illustrates the mapping between the high level data model and the middle level data model of these classes:

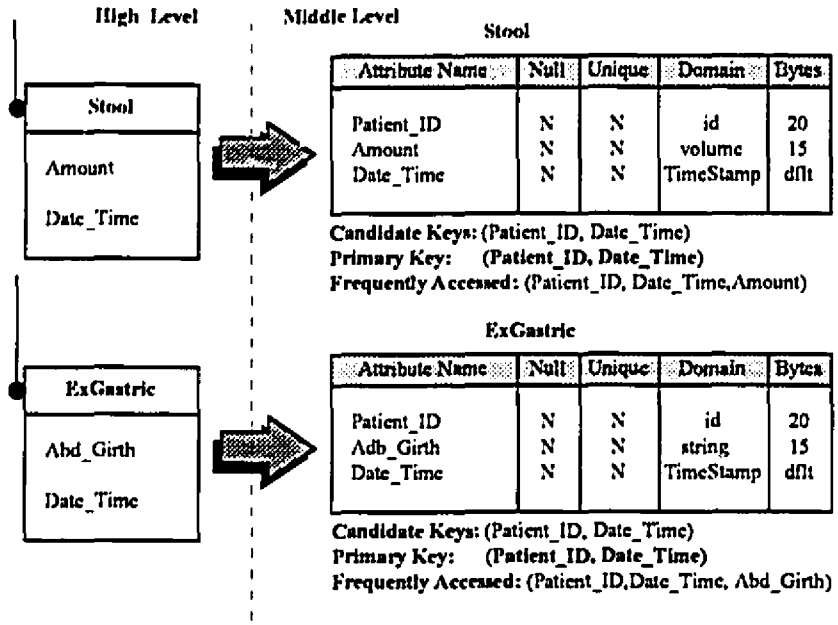


Figure 4.7: Stool & ExGastric Classes.

- *Stool.Patient_ID*: indicates the same value as *Patient.Patient_ID*. Similar to the other classes this attribute establishes the relationship between the *Stool* class and the *Patient* class as described in the database object model. Domain: Serial number, this attribute is not described to be *unique* in this table in order to allow multiple insertions of stool records that pertain to the same patient.
- *Stool.Amount*: indicates the amount of stool recorded at a certain time and date. Domain: Volume expressed as string.
- *Stool.Date_Time*: indicates the time and date the stool amount was recorded. Domain: Time Stamp.

- *ExGastric.Patient_ID*: indicates the same value as *Patient.Patient_ID*. This attribute establishes the relationship between the *Stool* class and the *Patient* class as as it is described in the database object model. Domain: Serial number, not *unique* in this table in order to allow multiple insertions of gastric abdominal girth records that pertain to the same patient.
- *ExGastric.Adb_Girth*: indicates the measured abdominal girth recorded at a certain time and date. Domain: length expressed as string.

- *ExGastric.Date_Time*: indicates the time and date the abdominal girth was measured. Domain: Time Stamp.

The application modules accessing the *Fluid Balance* tables are designed to perform only insert and retrieve record operations on the tables, based on their primary keys. If a fluid measurement needs to be corrected, then a new record with the current time stamp is inserted in the table. Therefore, measurements traceability is assured. Moreover, when first started or re-initialised, the application modules are designed to retrieve the measurements that occurred on the date specified by the user and store them in local application space variables.

Currently, the application user interfaces of the Fluid Balance module do not allow queries on measurements traceability, this has to be performed through the query manager of the database management system. However, the database stores all the data required to perform queries with various search criteria.

4.5.1 Fluid Balance Tables NF testing

This section tests the tables in the Fluid Balance module for Normal Form using the NF testing algorithm described in Section 4.3.4. First, we will consider testing the *IV* relation, (R_{IV}), for Normal Forms. R_{IV}=(Patient_ID, IV_Nbr, Body_Part, Comment, Level, Actual_Intk, Desired_Intk, Date_Time) with a primary key, X={Patient_ID, IV_Nbr, Date_Time}.

Since the primary key, X, of R_{IV} is composed of more than one attribute the algorithm starts testing for 2NF at steps 2 and 3.

STEPS 2 and 3

Let B_{IV} be the set of all nonprime keys of R_{IV}. B_{IV} = {Body_Part, Comment, Level, Actual_Intk, Desired_Intk}.

- *Body_Part* is fully functionally dependent on X.

In some cases, up to five IVs may be simultaneously injected into the patient's body. Hence, The *Body_Part* where the IV is injected depends not only on the *Patient_ID* but also on the date and time of that IV injectin and which IV number is it. In other words, if we extract any attribute from X, *Body_Part* will no longer functionally depend on X.

- *Comment* is fully functionally dependent on X.

  The comment made by the medical staff depends on the same functional parameters of *Body_Part*, hence the full FD X→{Comment}.

- *Level* is fully functionally dependent on X.

  The level of IV measured by the medical staff depends on the same functional parameters of *Body_Part*, hence the full FD X→{Level}.

- *Actual_Intk* is fully functionally dependent on X.

  The level of actual IV intake measured by the medical staff depends on the same functional parameters as *Body_Part* or *Level*, hence the full FD X→{Actual_Intk}.

- *Desired_Intk* is fully functionally dependent on X.

  The level of desired IV that should have been ideally injected into the patient depends on the same functional parameters a "Actual_Intk", hence the full FD X→{Desired_Intk}.

STEP 2 was "successfully" completed for every attribute in Biv. Therefore, Riv is in 2NF. Next, steps 4 and 5 will test Riv for 3NF.

STEPS 4 and 5

- *G(Body_Part)* = '(Patient_ID, IV_Nbr, Date_Time_In)}.

  There are no transitive dependencies in G(*Body_Part*), therefore we continue.

- *G(Comment)* = {(Patient_ID, IV_Nbr, Date_Time_In)}.

  There are no transitive dependencies in G(*Comment*), therefore we continue.

- *G(Level)* = {(Patient_ID, IV_Nbr, Date_Time_In)}.

  There are no transitive dependencies in G(Level), therefore we continue.

- *G(Actual_Intk)* = {(Patient_ID, IV_Nbr, Date_Time_In)}.

  There are no transitive dependencies in G(Actual_Intk), therefore we continue.

- *G(Desired_Intk)* = {(Patient_ID, IV_Nbr, Date_Time_In)}.

  There are no transitive dependencies in G(Desired_Intk), end.

We can notice that there exists no transitive dependencies in R_{IV}, therefore it is in 3NF. Since R_{IV} is in 3NF, in the following steps we will test for BCNF.

STEPS 6 and 7

- *G(Patient_ID)* = {Patient_ID}.

  Every $X_i \in$ G(Patient_ID) is a candidate key, hence we continue.

- *G(IV_Nbr)* = {Patient_ID, (Patient_ID, Date_Time)}.

  Every $X_i \in$ G(IV_Nbr) is a candidate key, hence we continue.

- *G(Date_Time)* = {Patient_ID, (Patient_ID, IV_Nbr)}.

  Every $X_i \in$ G(Date_Time) is a candidate key, hence we continue.

Considering that, every $X_i \in$ G(Body_Part), every $X_i \in$ G(Comment), every $X_i \in$ G(Level), every $X_i \in$ G(Actual_Intk) and every $X_i \in$ G(Desired_Intk) in steps 4 and 5, are candidate keys, then R_{IV} is in BCNF.

Next, we will consider testing the *Blood* relation, (R_{Blood}), for Normal Forms. R_{Blood}=(Patient_ID, Total_Loss, Total_Intake, Total_Balance, Date_Time) with a primary key, X={Patient_ID, Date_Time}.

Since the primary key, X, of R_{Blood} is composed of more than one attribute the algorithm starts testing for 2NF at steps 2 and 3.

STEPS 2 and 3

Let B_{Blood} be the set all nonprime attributes of R_{Blood}, B_{Blood}={Total_Loss, Total_Intake, Total_Balanace}.

- *Total_Loss* is fully functionally dependent on X.

  The blood loss value depends on two attributes the *Patient* and *Date_Time* the value was measured. Furthermore, if we extract any one attribute from the key X, *Total_Loss* will no longer be functionally dependent on X.

- *Total_Intake* is fully functionally dependent on X.

  Similar to *Total_Loss*, the blood intake value depends on two attributes the *Patient* and *Date_Time* the value was measured. Hence, if we extract any one attribute from the key X, *Total_Intake* will no longer be functionally dependent on X.

- *Total_Balance* is also fully functionally dependent on X.

  The blood loss value depends on both attributes of the key X, *Patient* and *Date_Time*. Hence, if we extract any one attribute from the key X, *Total_Balance* will no longer be functionally dependent on X.

STEP 2 was "successfully" completed for every attribute in B_{Blood}. Therefore, R_{Blood} is in 2NF. Next, steps 4 and 5 will test R_{Blood} for 3NF.

STEPS 4 and 5

- *G(Total_Loss)* = {(Patient_ID, Date_Time)}.

  There exists no transitive dependencies in G(*Total_Loss*), therefore we continue.

- *G(Total_Intake)* = {(Patient_ID, Date_Time)}.

  There exists no transitive dependencies in G(*Total_Intake*), therefore we continue.

- *G(Total_Balance)* = {(Patient_ID, Date_Time), (Total_Loss, Total_Intake)}.

  In this case, there exists a transitive dependency. *Total_Balance* is functionally dependent on the total blood loss and total blood intake since it represents their difference.

Therefore, we have the following transitive dependency: X→(Total_Loss), X→(Total_Loss) and (Total_Loss, Total_Intake)→(Total_Balance). Since STEP 4 failed for this specific attribute of B_{Blood}, the algorithm stops here and we deduce that R_{Blood} is not in 3NF.

The fact of having the *Blood* relation in 2NF only, requires us to analyse the options available for solving any consequent database design problem. We have two options to amend this situation. The first one is to further decompose the *Blood* relation schema until it meets the 3NF or the BCNF requirements. The second option is to ensure that the modules accessing or managing this table are aware of the aforementioned transitive dependency and are programmed to resolve any database anomality.

Since the modules that access the *Blood* table don't perform update and delete operations, database update anomalies are less likely to occur. Analysing the database design model, we can also note that the *Total_Balance* attribute is not required or used in any other module of the PDMS than the *Fluid Balance* module. Furthermore, the *Total_Balance* attribute does not have any kind of interdependency with other PDMS tables.

Decomposing the *Blood* relation into two relations adds another table to the database which affects the performance of the database manager. Therefore, we opted to resolve this potential database inconsistency or integrity problem at the application level. We ensured that the modules accessing the *Blood* table calculate the value of the *Total_Balance* attribute from the difference of the other two attributes (*Total_Loss* and *Total_Intake*) rather than reading and inserting it into the database table.

Therefore, the *Total_Balance* attribute will be dropped from the *Blood* relation schema and will be managed at the application level when and where required. The modification of the relation schema will be reflected in Chapter 5, upon the physical creation of the *Blood* relational table (Figure 5.7).

Finally, If we apply the NF testing algorithm presented in Section 4.3.4 to the remaining relation schemas of the Fluid Balance module, we will discover that they are also in BCNF.

4.6 Nursing Care Plan

The Nursing Care Plan module of the PDMS manages the care plan required for a patient based on the patient's age, diagnosis, medical procedures performed, allergies and possibly other individual circumstances [Roger *et al.*, 1992]. It also maintains and schedules the nursing actions and tasks associated with each plan.

The data in the NCP module is entered manually, the module incorporates a scheduler that will automatically schedule and assign the actions and tasks of the care plan to the nurses based on the input data of staff availability and level of expertise. Figure 4.8 illustrates the mapping between the high level model and the middle level model of the *NCP* class.

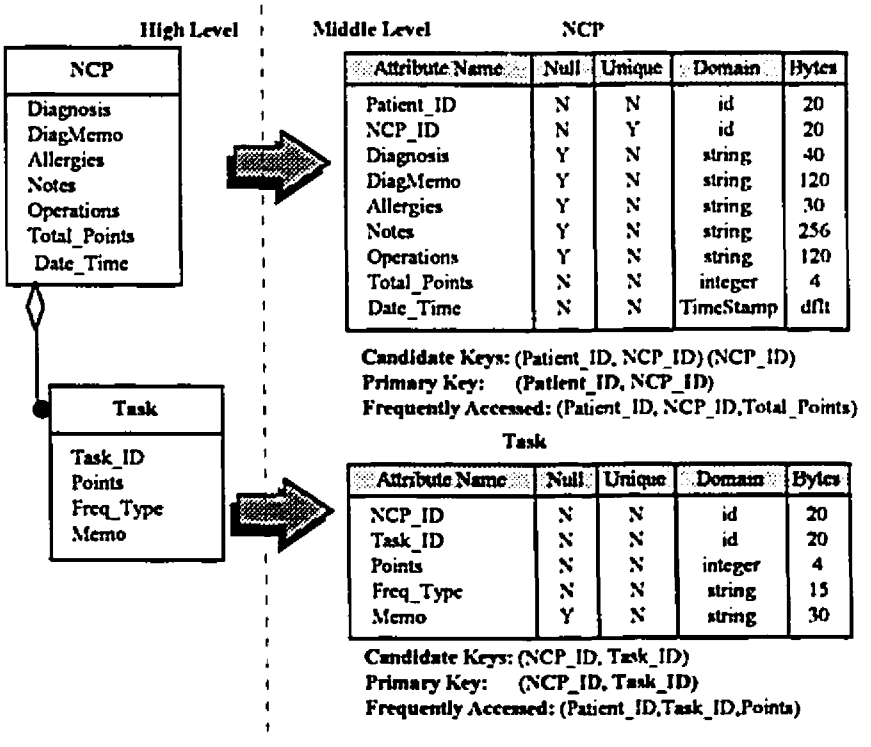


Figure 4.8: NCP & Task Classes.

- *NCP.Patient_ID*: indicates the hospital ID number given to the patient and is meant to establish the inheritance between the *NCP* class and the *Patient* class as described in the object model. Domain: Serial number, this attribute is not defined to be *unique* in order to allow multiple insertions of *NCP* records that pertain to the same patient.
- *NCP.NCP_ID*: is a sequentially-generated number which associates a unique object identifier with an individual *Nursing Care Plan*, and it used to maintain the aggregation relationship between the *NCP* class and its subordinate object classes. Domain: Serial number, must be a *unique* value in this table.
- *NCP.Diagnosis*: represents the diagnosis of the patient. Domain: String.
- *NCP.DiagMemo*: holds a memo, if available, that is associated with the diagnosis of the patient. Domain: String.
- *NCP.Allergies*: indicates whether the patient has any allergies; the NCP uses this attribute to check for any medications or solutions that should not be given to a patient with the indicated allergies. Domain: String.
- *NCP.Notes*: stores the physician and nursing staff notes, if any, about the patient's care plan. Domain: String.
- *NCP.Operations*: represents the medical procedures that the patient has undergone or is about to undergo. Domain: String.
- *NCP.Total_Points*: represents the sum of all the *Task* points found in this care plan. This is the method used by the Montreal Children's Hospital to measure the work load of a care plan. Domain: Integer.
- *NCP.Date_Time*: indicates the time and date the Nursing Care Plan was created. Domain: Time Stamp.

Figure 4.8 also illustrates the mapping between the high level model and the middle level model of the *Task* object class:

- *Task.NCP_ID*: should be the same as the *NCP.NCP_ID* and it is duplicated in the *Task* table in order to establish the aggregation relationship between the *NCP* class

and one of its constituent subordinate classes, namely the *Task* object class. Domain: Serial number, this number is not *unique* in this table.

- *Task.Task_ID*: uniquely identifies each of the tasks that are available in a Nursing Care Plan; there are about 283 different tasks defined as part of the Hospital's care ICU care system. A static code/decode table must be created in the database, this table translates each *Task_ID* into a name along with an explanation of the nature of that ICU care task. Domain: Integer.
- *Task.Points*: a number of points associated with each task, used in order to evaluate the effort required to execute a nursing care plan task, according to the PRN system. Domain: Integer.
- *Task.Freq_Type*: indicates the type of the *Task(s)* that is found in the *NCP*, the yype of the task is always associated with its frequency. For instance, "1HQT" means that the task of type "QT" should be performed hourly. Again, a static code/decode table that maps the task types to their meaning needs to be inserted in the database. Domain: String.
- *Task.Memo*: holds any memo that has been associated with the Task. Domain: String.

Figure 4.9 illustrates the mapping between the high level model and the middle level model of the *Medication* object class:

- *Medication.NCP_ID*: should be the same as the *NCP.NCP_ID*, and it is duplicated in the *Medication* table in order to establish the aggregation relationship between the *NCP* class and the *Medication* object class. Domain: Serial number, this attribute is not defined as *unique* in this table.
- *Medication.Name*: holds the name of the medication that should be administered as part of the NCP. Domain: String.
- *Medication.Route*: indicates the manner in which the medication should be administered, orally, by injection, etc. Domain: String.

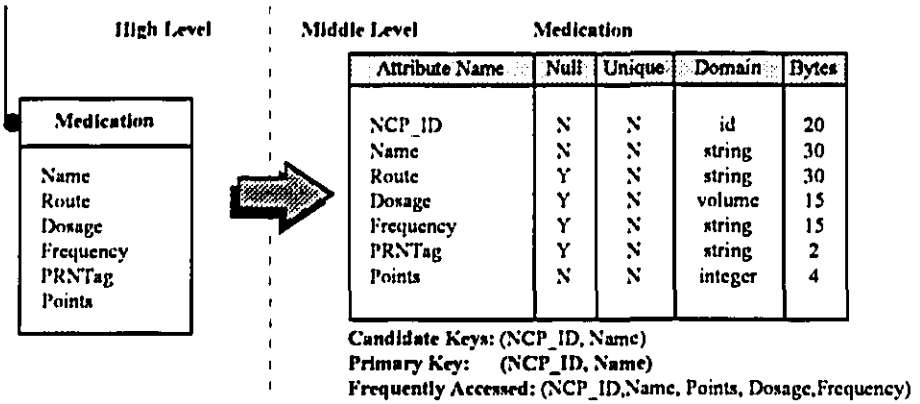


Figure 4.9: Medication Class.

- *Medication.Dosage*: indicates the dosage of medication that should be given to the patient. Domain: Volume expressed as string.
- *Medication.Frequency*: indicates the frequency of administration of that medication, hourly, every six hours, etc. Domain: String.
- *Medication.PRNTag*: indicates whether the Progressive Research Nursing system has been adopted by the hospital to evaluate the workload points of the medication section of the care plan [Roger et al., 1992], or not. Domain: String or flag.
- *Medication.Points*: indicates the number of points associated with administrating this medication. Domain: Integer.

Figure 4.10 illustrates the mapping between the high level model and the middle level model of the *Solutions* object class:

- *Solutions.NCP_ID*: should be the same as the *NCP.NCP_ID*, and it is duplicated in the *Solutions* table in order to establish the aggregation relationship between the *NCP* class and the Solution object class. Domain: Integer, not *unique* in this table.
- *Solutions.Number*: indicates the number of the solution that is being administered to the patient, i.e. the first, the second or the third. Domain: Integer, Numeric.
- *Solutions.Name*: holds the name of the solution that should be administered to the patient as part of the NCP. Domain: String.

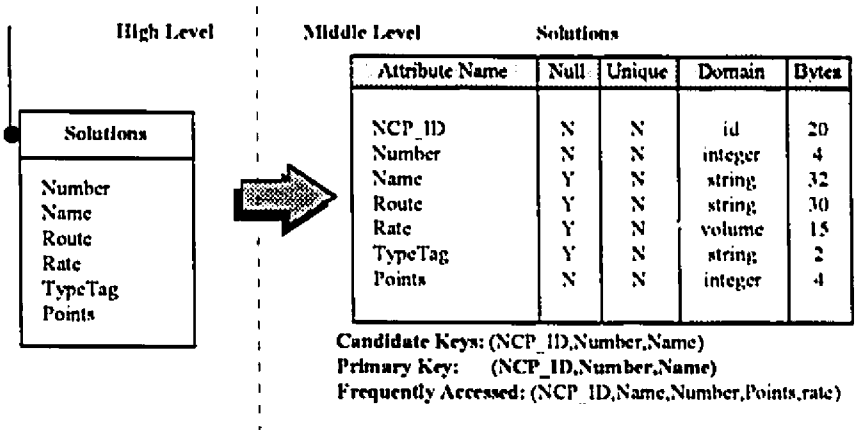


Figure 4.10: Solutions Class.

- *Solutions.Route*: indicates the manner by which the solution should be administered, orally, by injection or through an IV injection. Domain: String.
- *Solutions.Rate*: indicates the rate of administration of this solution. For example, two bags a day. Domain: String.
- *Solutions.TypeTag*: indicates which type of system has been used to evaluate the workload points of the solutions section in the care plan. Domain: String.
- *Solutions.Points*: indicates the number of points associated with administrating this solution. Domain: Integer.

Similar to the Fluid Balance application modules, the NCP application modules that handle the NCP tables are designed to perform only insert and retrieve record operations based on the table's primary keys. This ensures records correction traceability. Furthermore, when first started or re-initialised, the application modules are designed to retrieve the plans that occurred on the date specified by the user and store them in local application space variables.

4.6.1 Nursing Care Plan Table NF Testing

This section tests the relation schemas of the Nursing Care Plan module for Normal Form using the NF testing algorithm described in Section 4.3.4. If we apply the NF

testing algorithm presented in Section 4.3.4 to the relation schemas of the NCP module, we will discover that they are in BCNF.

4.7 Data Link Controller

The Data Link Controller (DLC) module acquires and stores the vital sign data transmitted by the network of bedside monitors in real-time mode. This information is automatically gathered without manual entry. The data acquired by the DLC is of two major types:

1. *Parameter Data* are monitored numerical data values which originate from the bedside monitors, such as heart beat rate, blood pressure or respiration rate. Parameter data is identified by a Medical Function Code (MFC), and its values are updated through the bedside monitors every 1024 ms.
2. *Wave Data* is digitised information, sampled representation of the wave form generated by the bedside monitors.

Figure 4.11 describes the mapping between the high level information model and the middle level information model of the *Parameters* class:

- *Parameters.Patient_ID*: indicates the same value as *Patient.Patient_ID*. The purpose of duplicating this attribute between the *Patient* and the *Parameters* class is to establish the relationship as described in the object data model. Domain: Serial number, this attribute is not defined to be *unique* in this table.
- *Parameters.Type*: indicates the type of recorded the vital sign parameter. Domain: List of forty types of parameters; i.e., list of 40 MFCs.
- *Parameters.Value*: indicates the vital sign parameter's value recorded at an instance in time. Domain: Bit data type.
- *Parameters.Time_Stamp*: indicates the value of the time instance when the parameter was measured. This stamp is generated by the bedside monitor, and designates the time that value was recorded. Domain: Time Stamp.

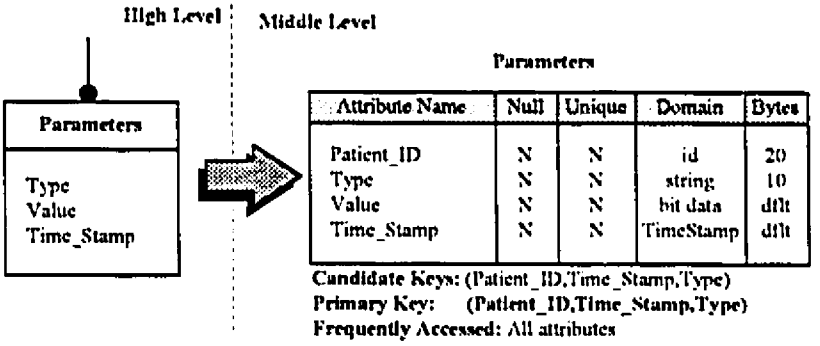


Figure 4.11: Parameters Class.

The application modules using the DLC *Parameters* table are designed to perform only insert and retrieve record operations on the table based on the primary keys. However, in this module, insertions are done only by a transfer process, which is described in the following section.

4.7.1 Data Link Controller Table NF Testing

Applying the NF testing algorithm to identify the Normal Form of the *Patient* relation seems excessive, three out of the four attributes that belong to that relation compose its primary key. Thus, it is clear that this relation is in BCNF or higher.

However, one may argue as to why is the attribute *Type* part of the key. The bedside monitors are capable of measuring a large number of vital signs, of which, only forty are required to be recorded in the PDMS database. Each of these parameters has a different type of measuring unit such as beat/second or flow/minute. Since the value of the measured parameter is inserted in bit data format, we need to know its type in order to determine its measuring unit. That is achieved through a code/decode table which holds the parameters types and their corresponding measurement unit.

This implies that the recorded "value" cannot functionally depend on the patient and the time when it has been measured, it must also depend on its type.

4.7.2 Levels of Parameter Data Management

In order to address the real-time quick response in data management required in the DLC module, a multilevel management of data storage has been adopted. In the current implementation of the DLC module, the acquired parameters data are stored in main memory resident queues [Fumai *et al.*, 1991]. Then, in order to avoid disk access every two seconds, data is transferred in blocks to disk in periodic intervals. Physical data block transfer is an appealing technique because of its simplicity. However, it may have a few inadequacies--for instance, it does not support an object representation change when moved between levels. On disk, objects are represented by unique identifiers, whereas in main memory they are referenced by pointers. Another shortcoming is that conventional relational DBMSs support a main memory cache of system catalogue objects (e.g. create table, open table, seek position, etc.) and objects must change representation when transferred from main memory to disk table format. Changing representation when moving objects between levels constitutes a threat to data security and must be properly managed using semaphores.

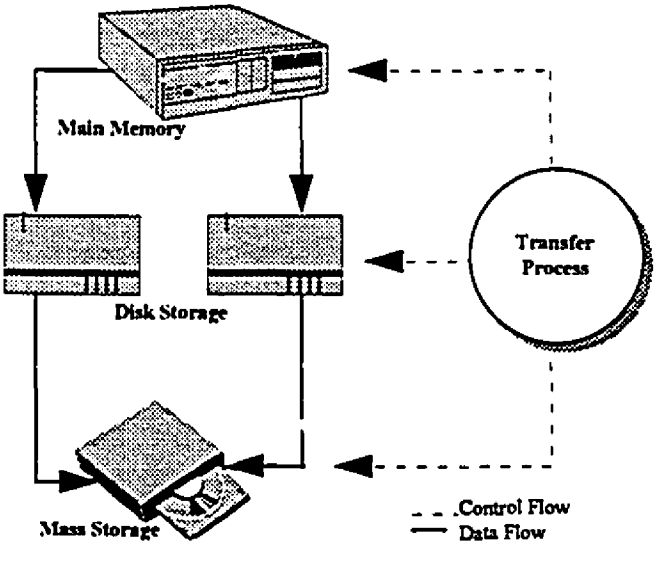


Figure 4.12: Multilevel Storage Scheme.

Assuming that storage systems consist of logical devices forming a rooted tree, where the unique root node is the main memory and the rest of the storage levels are descendants

[Stonebraker, 1991], a logical three-level storage management is considered in this design (Figure 4.11).

The storage of parameter data is managed at three levels, the first level being main memory. The second level of storage is the hard disk level, which at the same time could be distributed across the computer network. The last level is the massive tape storage. Therefore, an object may exist in any of the three levels of databases. In another sense, the three logical databases coexist. Furthermore, in the logical model of multilevel data management the same format of disk data is maintained in both main memory and archive. In order to determine the criteria for object transfer between levels, we perceive a background decision engine (Data Transfer Process) between each of the three levels, as illustrated in Figure 4.12. The transfer engine can determine how and when the transfer should take place according to the programmed conditions. A pseudo-code example for one such set of criteria for Real_Time Parameters data could be:

- **main memory representation**: transfer Parameter data from RAM to disk every *Time* seconds, where *Time* is a variable in seconds.
- **disk representation**: transfer Parameters data from disk to archive device where *Date in Time-Stamp is within a certain period of time*.
- **archive representation**: archive Parameters data where *Date in Time-Stamp* $\geq$ *a certain date*.

What the example above suggests is that when the *Real-Time* data is acquired, it is kept resident in main memory for thirty minutes (1800 seconds) and that within a certain period during the year the data can still be kept on disk. After a certain given date, however, the data should be moved to massive storage. The mentioned conditions are a sample case and can easily be changed without introducing any modifications to the database system. Moreover, a conditional statement could be added to the transfer engine where data is transferred to hard disk every five minutes for security and data recovery purposes in case of a system crash or any other unexpected malfunctioning event. The actual system keeps the data on hard disk as long as the patient is still in the ICU, then moves it to mass

storage upon the patient's discharge. Afterwards data is traditionally archived on off-line tape storage, for access only when needed.

The Main Memory level of data management of the DLC module has been implemented in a previous work [Fumai *et al.*, 1991]. The second and third levels of the *Parameters* data management has been implemented as described in Figure 4.12. The code illustrated in Figure 4.11 has been used to create the database *Parameters* table on disk and in massive tape storage.

The main memory level stores data only for those patients currently in the ICU. Main memory resident data is stored in three circular queues in the computer's RAM: a *second* data queue, a *minute* data queue and a *half-hour* data queue. For each active patient, *Parameters* data is received from the CarePort interface simulator every two seconds and stored in the *second* data queue. After sixty seconds, data items in the *second* queue are averaged over the minute and stored in the *minute* data queue. Data items in the *minute* queue are then averaged over thirty minutes and placed in the *half-hour* queue.

Even though we were only required to permanently store the half-hour DLC data and archive it, it seemed necessary to create three such *Parameters* tables in the database at the middle data management level; i.e., in disk space. The *Parameters* tables are exact images of one another. One, *Parameters_second*, stores the *second* data. Another, *Parameters_minute*, stores *minute* data, and finally the *Parameters* table itself contains *half-hour* data.

The *Parameters* database table was replicated for the following reasons:

1. Keeping copies of the main memory resident data reduces the data lost due to an application break or system shutdown.

2. Second and minute parameter data can be displayed from either main memory or from disk storage depending on the urgency at hand. This reduces main memory operations and enhances system performance.

3. The Trend Analysis module could access data from the database rather than from main memory, enhancing the system performance and allowing the Trend Analysis module to analyse historical cases when required.

4. Second and minute parameter data is only kept for a finite period of time in disk space and is not archived to massive storage; i.e., the second and minute database tables require no extra space.

Chapter 5 Implementation, Results and Future Extensions

This chapter describes the physical implementation of the PDMS database. It delineates the derivation of the low level (physical) information model from the middle level model presented in Chapter 4. The physical model will then be translated into actual database tables according to the specifications of the implementation platform.

5.1 Implementation Platform

The database tables were implemented using the Relational Database Management Services of the OS/2 Extended Edition Database Manager®. The creation of the database tables was carried out in a series of SQL statements grouped under a single script file. The interaction with the database such as insertion, retrieval, sorting, viewing and archiving of data instances was handled by a series of source modules, that access the database, coded in the C programming language with embedded SQL statements.

The PDMS database prototype has been implemented in our CIM laboratory on a Local Area Network (LAN) of IBM PS/2® computers. The computers were connected in a Token-Ring mode and used the OS/2 LAN Manager® services for network control and communication protocols. Figure 5.1 illustrates the network topology of these computers.

The Extended Edition of Database Manager offers a database *distributed* feature which consists of providing a client application program access to databases located on Extended Services database servers. In order for a client application to access and manipulate data on a server database, the database *client application enabler* must be installed and configured on the client's workstation.

In order to test the PDMS prototype and its database, a program that simulates the HP CarePort Network of the bedside monitors has been implemented. This simulator transmits data similar to the data incoming from the CarePort and connects to the PDMS LAN via a RS-232 serial link to one of its Personal Computers (PC), named PC1. This

simulator has been implemented as part of an earlier work on the PDMS project [Fumai et al., 1991].

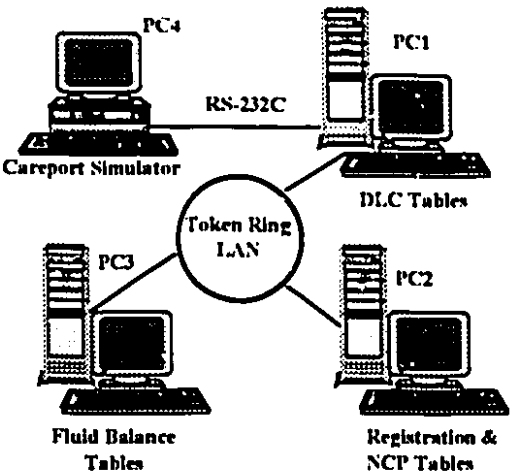


Figure 5.1: PDMS Network.

The user interfaces of the PDMS application modules were developed using the C programming language and the Application Interface Calls of OS/2 Presentation Manager.

5.2 Network Database Architecture

Using the distributed database capabilities of the Extended Edition of OS/2 Database Manager and the services of OS/2 LAN Manager, the database has been distributed as shown in Figure 5.1.

PC1

The first computer of the LAN, PC1, is dedicated to collecting electronic data from the network of bedside monitors. The DLC application module and the DLC database tables resident on this station are configured as a server database to the remaining computers on the LAN, and reciprocally PC1 is configured as a client to the databases located on PC2 and PC3.

This configuration allows PC1 to access the tables maintained on PC2 and PC3. It also allows PC2 and PC3 to access the database tables resident on PC1. PC1 contains

executable application images of the Fluid Balance module, the NCP module and the Registration module. When executed as client applications from PC1, these applications present the user with their own corresponding user interfaces, with embedded SQL statements that allow access and manipulation of database tables stored on PC2 and PC3.

PC2

The second PC is dedicated to managing the Registration module data and the NCP data; it also maintains the NCP and Registration application modules. The Registration and the NCP database tables on this station are configured as a server database to PC1 and PC3, and reciprocally PC2 is configured as a client of the database tables located on PC1 and PC2.

In the same manner as that of PC1, PC2 contains executable application images of the Fluid Balance module and the DLC module, so that when executed from PC2 these images present the user with their own user interfaces accessing the remaining database tables over the LAN.

PC3

The third PC is dedicated to managing Fluid Balance information; it maintains the Fluid Balance application module together with its database tables. Like the other PCs, the Fluid Balance tables that reside on this station are configured as a server database, while at the same time the station is configured as a client of the database tables of the other two PCs.

As well, this PC contains executable client applications of the NCP module, the DLC display and query modules (only) and the Registration module which facilitate access and display of data stored on PC1 and PC2.

5.3 Results

This section will examine and evaluate the physical data model derived from the middle level table data model using the OMT data modelling method.

5.3.1 DLC

Only the DLC module source code and the Data Transfer Engine (Process) contain the required embedded SQL statements that insert and retrieve data from the *Parameters* database tables.

All the database tables are created only once using a *script* file that contains all the SQL statements for creating the tables. The SQL statements are specific to the OS/2 Database Manager. Figure 5.2 illustrates how the *Parameters* table creation code has been derived from the middle level presentation of the *Parameters* class. As described is Section 4.7.2, similar code was used to create two other tables, namely the "Parameters_second" and "Parameters_minute" tables.

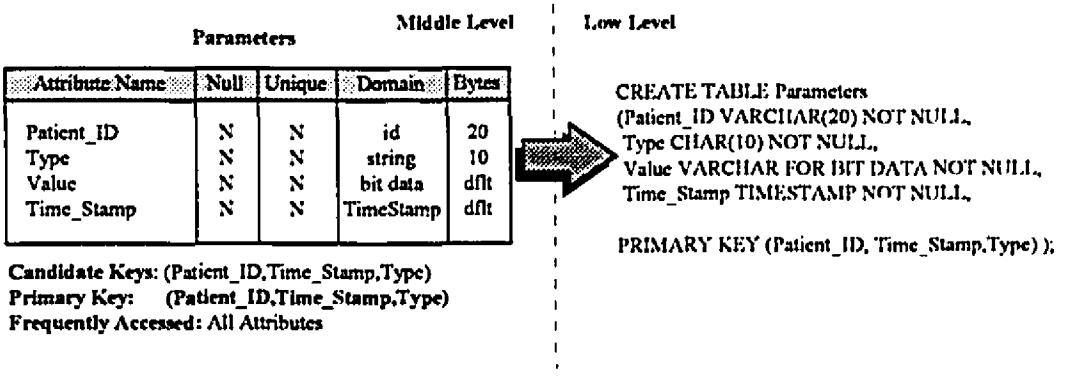


Figure 5.2: Parameters Class - Physical Creation Statement.

5.3.2 Registration

Figures 5.3 and 5.4 illustrate how the SQL statements that create the Registration database tables were derived from the middle level data model of the *Patient* and *Admittance* object classes. Only the Registration module is responsible for handling data in these database tables.

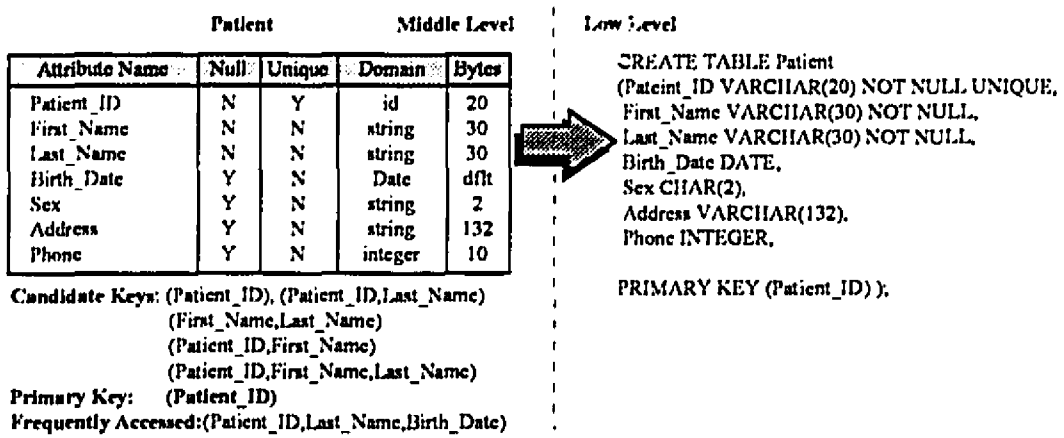


Figure 5.3: Patient Class - Physical Creation Statement.

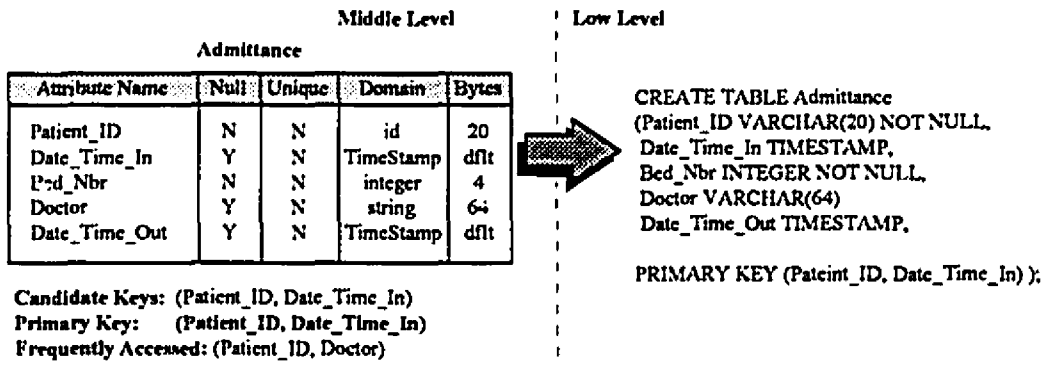


Figure 5.4: Admittance Class - Physical Creation Statement.

5.3.3 Fluid Balance - Ingesta

Figures 5.5, 5.6, 5.7, 5.8, 5.9 and 5.10 illustrate how the SQL statements that create the Fluid Balance database tables were derived from the middle level table model of the Fluid Balance object classes. Only the Fluid Balance modules contain the embedded SQL statements that manipulate the data instances of the Fluid Balance database tables.

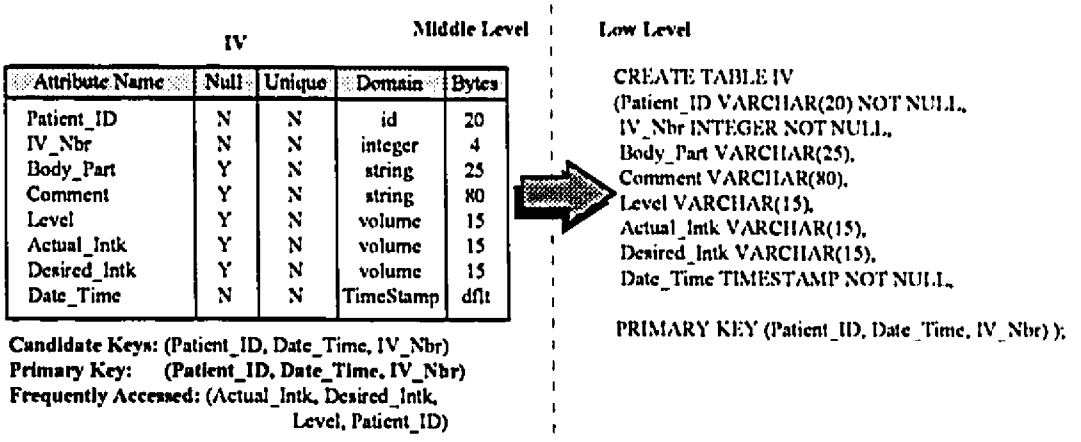


Figure 5.5: IV Class - Physical Creation Statement.

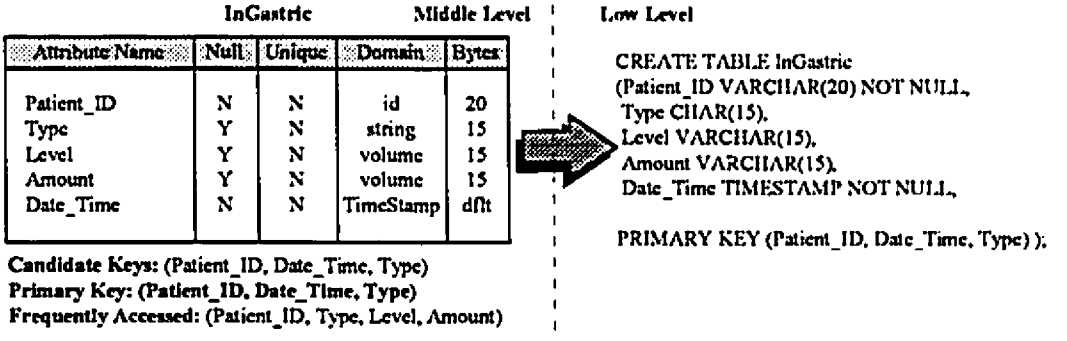


Figure 5.6: InGastric Class - Physical Creation Statement.

5.3.4 Fluid Balance - Excreta

This section presents the Excreta database tables of the Fluid Balance module, Figures 5.7 to 5.10.

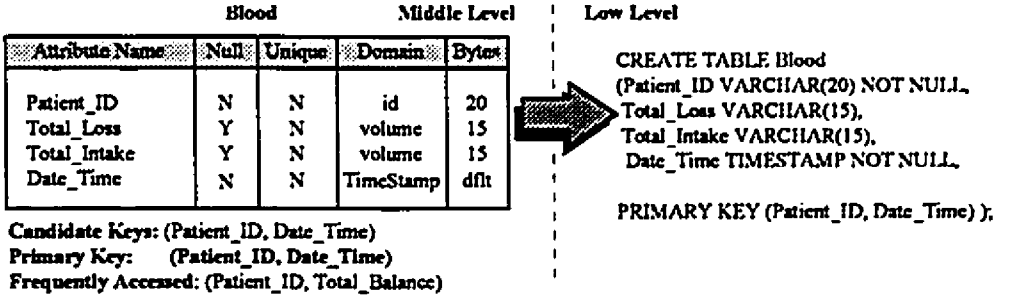


Figure 5.7: Blood Class - Physical Creation Statement.

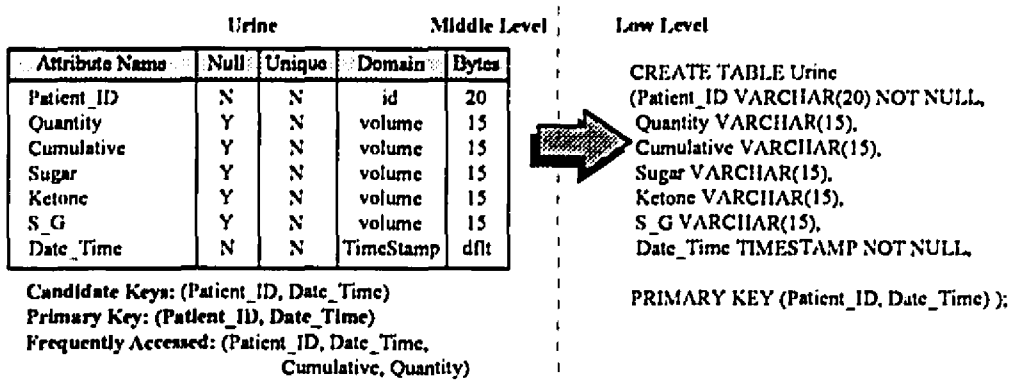


Figure 5.8: Urine Class - Physical Creation Statement.

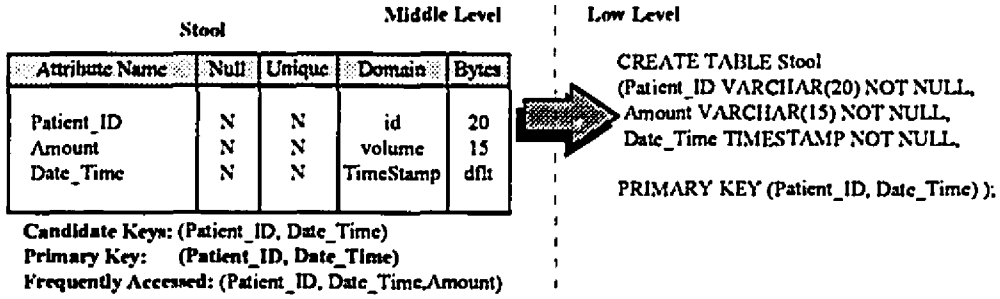


Figure 5.9: Stool Class - Physical Creation Statement.

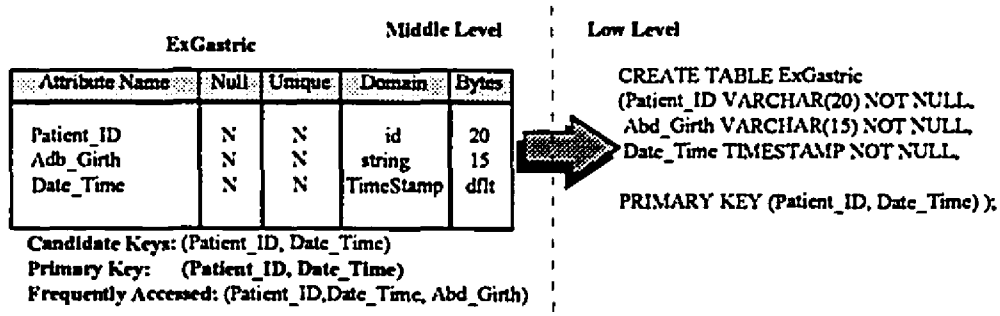
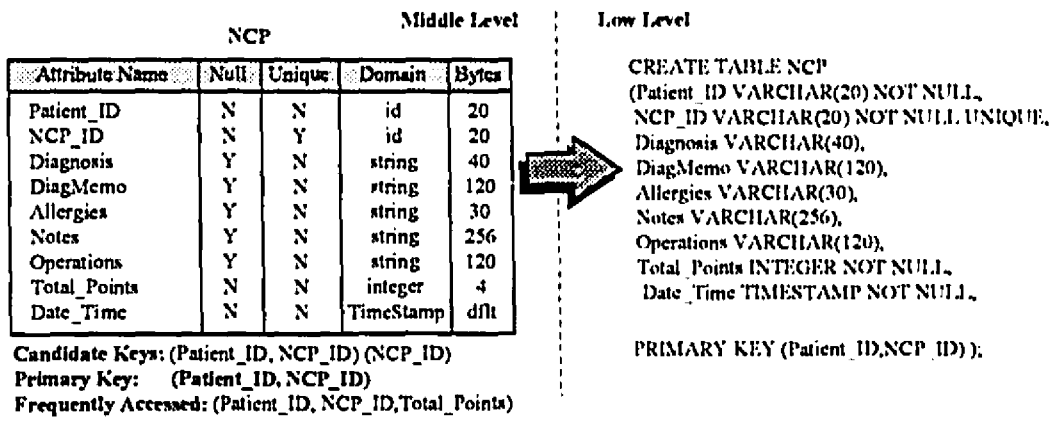


Figure 5.10: ExGastric Class - Physical Creation Statement.

5.3.5 Nursing Care Plan

Figures 5.11, 5.12, 5.13 and 5.14 illustrate how the SQL statements that create the *Nursing Care Plan* (*NCP*) database tables were derived from the middle level data model of the *NCP* object classes. Again, only the NCP modules contain the embedded SQL

statements that perform the insertion and retrieval functions on the *Nursing Care Plan* database tables.

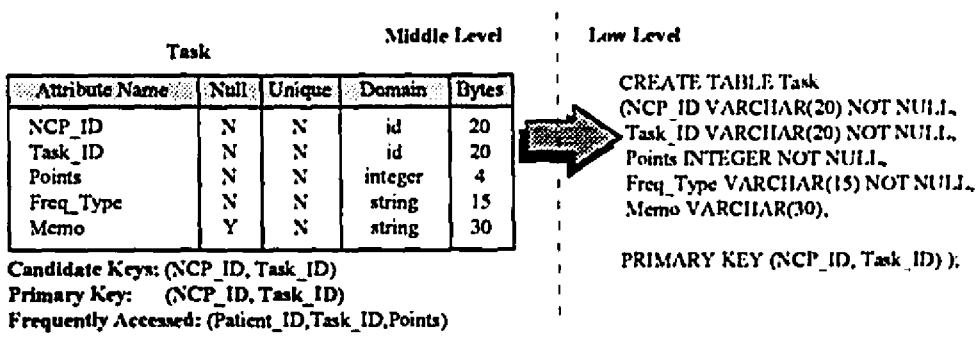


NCP

Attribute Name	Null	Unique	Domain	Bytes
Patient_ID	N	N	id	20
NCP_ID	N	Y	id	20
Diagnosis	Y	N	string	40
DiagMemo	Y	N	string	120
Allergies	Y	N	string	30
Notes	Y	N	string	256
Operations	Y	N	string	120
Total_Points	N	N	integer	4
Date_Time	N	N	TimeStamp	dflt

Candidate Keys: (Patient_ID, NCP_ID) (NCP_ID)
Primary Key: (Patient_ID, NCP_ID)
Frequently Accessed: (Patient_ID, NCP_ID,Total_Points)

Figure 5.11: NCP Class - Physical Creation Statement.

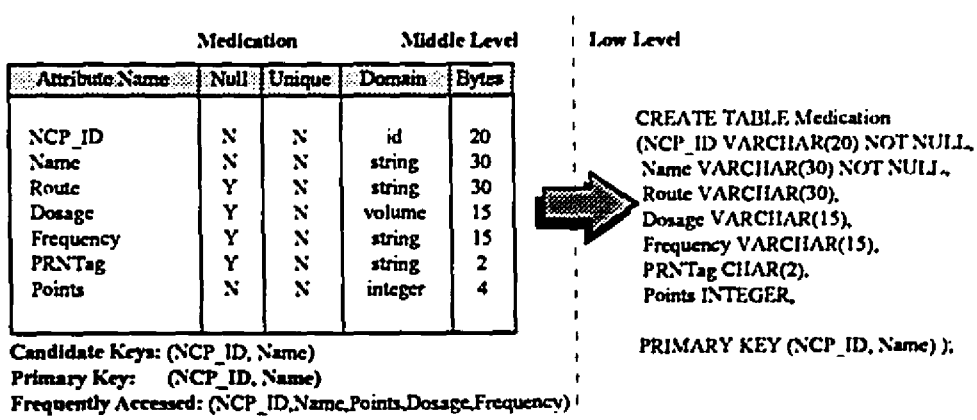


Task

Attribute Name	Null	Unique	Domain	Bytes
NCP_ID	N	N	id	20
Task_ID	N	N	id	20
Points	N	N	integer	4
Freq_Type	N	N	string	15
Memo	Y	N	string	30

Candidate Keys: (NCP_ID, Task_ID)
Primary Key: (NCP_ID, Task_ID)
Frequently Accessed: (Patient_ID,Task_ID,Points)

Figure 5.12: Task Class - Physical Creation Statement.



Medication

Attribute Name	Null	Unique	Domain	Bytes
NCP_ID	N	N	id	20
Name	N	N	string	30
Route	Y	N	string	30
Dosage	Y	N	volume	15
Frequency	Y	N	string	15
PRNTag	Y	N	string	2
Points	N	N	integer	4

Candidate Keys: (NCP_ID, Name)
Primary Key: (NCP_ID, Name)
Frequently Accessed: (NCP_ID,Name,Points,Dosage,Frequency)

Figure 5.13: Medication Class - Physical Creation Statement.

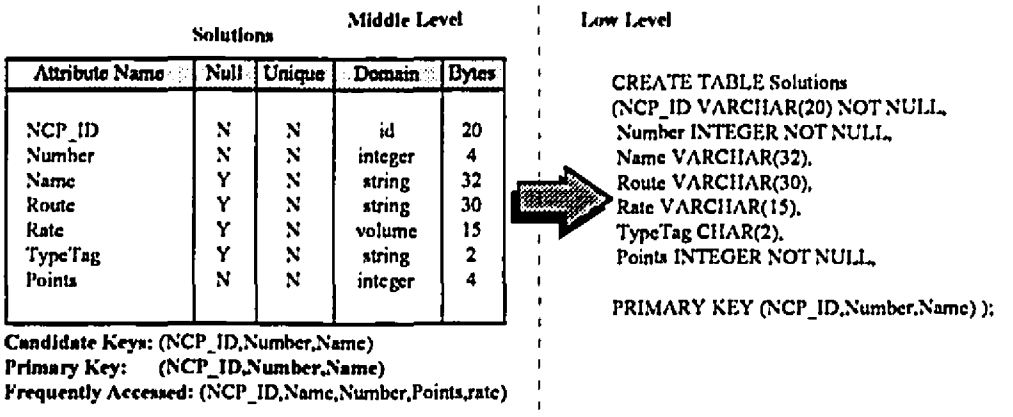


Figure 5.14: Solutions Class - Physical Creation Statement.

5.3.6 The Application Level

The programs that access or handle the PDMS tables were coded in the C programming language with embedded 4GL SQL statements. A sample program that inserts a tuple in the *InGastric* PDMS database table is described in this section. The example reads, from the input user interface, the patient's identification number, the gas type and level and the date and time the measurements were recorded. It then inserts these values into the corresponding database table. Such program is composed of three sections:

1. The first section declares the host variables. Host variables are reserved by the program to access the table attributes as they have been defined in the database table. These variables should match with their corresponding database table attributes in type and size. For example,

```
EXEC SQL BEGIN DECLARE SECTION;
char    hst_PatientId[20];
char    hst_GasLevel[15];
char    hst_GasType[15];
char    hst_GasAmount[15];
char    hst_DateTime[16];
EXEC SQL END DECLARE SECTION;
```

2. The second section handles the logic of the program and can be executed before and after the third section depending on the logical requirements of the program. For

example, after having read the input variables from the user interface into the application's local variables, this section of the application program inserts the values of the patient's *id.*, the gas type, the gas level the gas amount and the date/time variables into their respective host variables:

```
strcpy(hst_PateintId, PateintId);
strcpy(hst_GasType, GasType);
strcpy(hst_GasLevel, GasLevel);
strcpy(hst_GasAmount, GasAmount);
strcpy(hst_DateTime, DateTime);
```

3. The third section executes the SQL statement which could be an insertion, deletion or retrieval of one or more tuples in the database table. For instance, this section executes the insertion of the aforementioned tuple into the database table. The first step in executing the search SQL command is to prepare it. This is indicated by the PREPARE STATEMENT command illustrated below. The second step executes the prepared SQL statements and also indicates the variable values that should be respectively used in the SQL statement. The "?" sign in the SQL statement indicates that the values should be respectively read from the variables declared in the USING statement;

```
EXEC SQL;
        PREPARE STATEMENT:  "INSERT INTO InGastric Patient_ID,
        Type, Level, Amount, Date_Time VALUES ( ? ? ? ? ? ) ;"

EXEC SQL:
        EXECUTE STATEMENT USING: hst_PatientId, :hst_GasType,
        :hst_GasLevel, hst_GasAmount, :hst_DateTime;
```

5.4 Data Integrity

This section will discuss some of the data integrity issues that were addressed or considered during the design and development of the PDMS database.

5.4.1 Data Types

Note that in the translation of the attribute domains from the middle level data model into the low level code the VARCHAR data type was used to implement the object identifiers such as the *Patient_ID*, the *NCP_ID* and the like. It was also used to implement the volumes of fluids and solutions measured by the medical staff.

The VARCHAR FOR BIT DATA data type was used to implement the *value* attribute of the *Parameters* object table which is stored in *bit* format in main memory. The FOR BIT DATA data type allows the storing of data which may not have constant length. When read from main memory the BIT DATA type is treated as a structure with two fields: a data length indicator and a byte array which holds the values.

The TIMESTAMP, DATE, TIME and INTEGER data types are pre-defined by the SQL of the OS/2 database manager.

The correct mapping of data types between the middle level data model and the low level code as well as between the application module source code and the database guarantees the integrity of the data values. Once inserted in the database the OS/2 Database Manager is responsible for maintaining that integrity.

5.4.2 Data Constraints

In order to maintain the *referential integrity* that has been defined by the High Level data model as relationships among object classes, the middle level data model specifies which attributes should never be inserted in the database with a NULL value and which should contain a UNIQUE value in the database table as in the case of an object identifier type (ID). This is translated in the low level model (code) as follows:

1. First, by specifying that a certain attribute is NOT NULL UNIQUE in the SQL statement that creates the object table indicates to the database manager that when inserting a new instance of this attribute in the database it should first verify that the value of the attribute is not null and unique in the table before allowing the insertion.

2. Second, indicating to the database manager that a *referential constraint* exists on an attribute in a certain object table, will then require that the given value of that attribute exist in some other table before insertion is allowed. For example, one of the referential constraints that has been inserted in the database requires that every *Patient_ID* in the *NCP* object table must first exist in the *Patient* object table and in the *Admittance* object table. Hence, the database manager will not allow the insertion of new *Nursing Care Plan* for a *Patient* that does not exist in the database and has not been admitted to the ICU.

The referential constraints that has been introduced in the database are directly derived from the High Level Inheritance Diagram. Any child class cannot exist without the existence of its parent class. In other words, if the instance of an object identifier of parent object class, referred to as ID in the middle level Model, does not exist in the database an instance of the child object class cannot be introduced in the database.

5.4.3 Other Data Integrity Issues

The database tables are fragmented in a manner that helps avoiding data access concurrency. In any case, the OS/2 Database Manager is responsible for concurrency control and record locking.

In order to protect the database and the table definitions, the OS/2 Database Manager was configured to grant data read-only access for all the users that use the OS/2 query manager, data write-access for certain users that use the PDMS Application Modules and database administrator access to a single specific user. The database administrator has all the privileges granted by the OS/2 Database Manager such as altering the configuration of the Database Manager, changing the definitions of the database tables and the like.

5.5 Data Recovery

The design and implementation of the PDMS identifies three types of failure that can result in data loss. This section presents these failure types and assesses the impact of data loss in each case as well as the procedures in place to recover the data:

Transaction Failure

This type of failure usually ends with an *abort*, and occurs when a transaction terminates in a normal failure because inconsistencies in the database were detected, or the PDMS application modules surrounding the database were not able to correctly process the embedded SQL statement.

In this case only a single record is most likely to be lost from the database since the each embedded SQL statement in the PDMS Application Modules surrounding the database completes its operation with a COMMIT to database command, rather than performing a set of database transactions and then committing them all to the database. This procedure ensures that every transaction commits its data to the database before concluding. However, loss of a single record may only be true in the DLC module since the data acquisition is computerized. According to the PDMS requirements, the loss of a single parameter value over a second is considered minor and no data recovery is required. In the remaining PDMS modules the user would be prompted to repeat the transaction.

System Failure

This type of failure could be caused by anything that destroys the computer's main memory. The most common sources of such a failure are hardware failures in the CPU, bugs in the Database Manager code, bugs in the operating system and the like. This type of failure could also be caused by a power failure if the system is not connected to an Uninterrupted Power Supply.

There OS/2 Database Manager offers two features that can be used in this case, *restore* and *roll-forward*. The restore mechanism involves making copies of the database at scheduled times. The roll-forward recovery method allows us to rebuild the database to a specified point in time as well as to the end of the database log files.

However, the data resident in main memory is unrecoverable in a system failure. Assuming a transfer period of five minutes for the storing of parameter data from RAM queues, the worst case data loss is five minutes worth of Parameters data. Also note that during the down time of the system the DLC module cannot acquire data from the bedside monitors and therefore data will not be stored in the PDMS database during that period.

Media Failure

This type of failure occurs when a secondary storage medium encounters a malfunction, such as a head crash on the hard disk or a bug in the operating system I/O routines.

In order to prevent the media failure from destroying both the database and the log files needed to rebuild it, the database logs should be kept on a media device different from the database itself. In this manner, if the device that holds the database fails, it can be restored from the log files kept on the other medium, and vice-versa.

5.6 Performance and Sample Results

This section presents some benchmark results obtained while operating the PDMS in the development lab at McGill using the sample data generated by the CarePort simulator (Figure 5.1). The objective, in this case, was to evaluate the PDMS database implementation from a network perspective.

Table 5.1 illustrates the response times of various test conditions performed on the database. These times were measured, in our laboratory environment (Figure 5.1), on a two PS/2s model 80 computers with an 80386 16 MHz CPU, 8 Mega Bytes (MB) RAM, 150 MB hard disk storage (20 ms access time, 800 KB/sec transfer rate), PC1 and PC2. And on a PS/2 model 57 computer with an 80386 16 MHz CPU, 8 MB RAM, 150 MB hard disk (20 ms access time, 800 KB/sec transfer rate), PC3. The PC running the Careport simulator software was a PS/2 model 60 with an 80286 16MHz CPU, PC4. All computers ran version 1.3 of OS/2 and the Extended Services of OS/2 Database Manager version 2.2.

The response times were measured by inserting code statements in the application modules that read the OS/2 system clock before and after each operation execution. And the response time was calculated from the difference of these two time stamps.

The transaction response times were measured for three ICU patients registered with the PDMS. Analyzing the response times under the test conditions described in Table 5.1 allows a determination of the load effect on the database performance. These conditions were tested while PC1 collected parameters data for three patients simultaneously, during a one-hour time period. The various test conditions are described below:

Test Condition	Client	Server	Response Time
Register new patient	PC2	PC1	2010ms
Register new patient	PC1	PC2	2120ms
Update exsiting Patient	PC2	PC2	1000ms
Update existing patient	PC1	PC2	1500ms
Retrieve Patient Data (three Patients)	PC2	PC2	210ms
Retrieve Patient Data (three patients)	PC3	PC2	450ms
Retrieve Admittances (three patients)	PC2	PC2	200ms
Retrieve Admittances (three patients)	PC3	PC2	460ms
Insert parameter_minute data, 1patient	PC1	PC1	200ms
Insert parameter_minute data,3patients	PC1	PC1	500ms
Insert parameter(halfhour)data, 1patient	PC1	PC1	200ms
Insert parameter(halfhour)data,3patients	PC1	PC1	480ms
Half hour data retrieval (3 patients)	PC1	PC1	2055ms
Half hour data retrieval (3 patients)	PC2	PC1	5050ms

Table 5.1: Response Times

1. *Register new patient.*

This test condition involves admitting a new patient into the ICU via the *Registration* module. This transaction requires adding an instance in the *Patient* object and another in

the *Admittance* object table, as well as activating the DLC module to start collecting data from the patient's bed.

This transaction was carried out in two ways, the first locally in which data was inserted into the *Patient* and *Admittance* tables on the same station (PC2), and the second remotely over the LAN in which data was entered on PC1 and inserted in the corresponding tables located on PC2. The response times measured in both cases were acceptable.

2. *Update existing patient.*

This test condition involves admitting a patient for the second time into the ICU. That is, the patient's ID already exists in the system, via the Registration module. This transaction requires updating the information in the *Patient* object such as the patient's phone number or address as well as a new instance in the *Admittance* object table. Again the DLC module is activated to start collecting data from the patient's bed.

This transaction was also carried out in two ways locally and remotely. The response times measured in both cases were within favorable limits.

3. *Retrieve Patient and Admittance Data.*

These test conditions were carried out by querying the *Patient* and *Admittance* database tables via the OS/2 Database Query Manager. The queries were executed in two manners, on the same station that holds the data and the other remotely. In both cases the response time recorded was acceptable.

The local query that was carried out on the *Patient* table requested the patient hospital ID, last name, first name, birth date, sex and age of all patients in the ICU. This was accomplished by the following SQL statement:

```
SELECT Patient_ID, First_Name, Last_Name, Date_of_Birth, Sex, FROM
Patient;
```

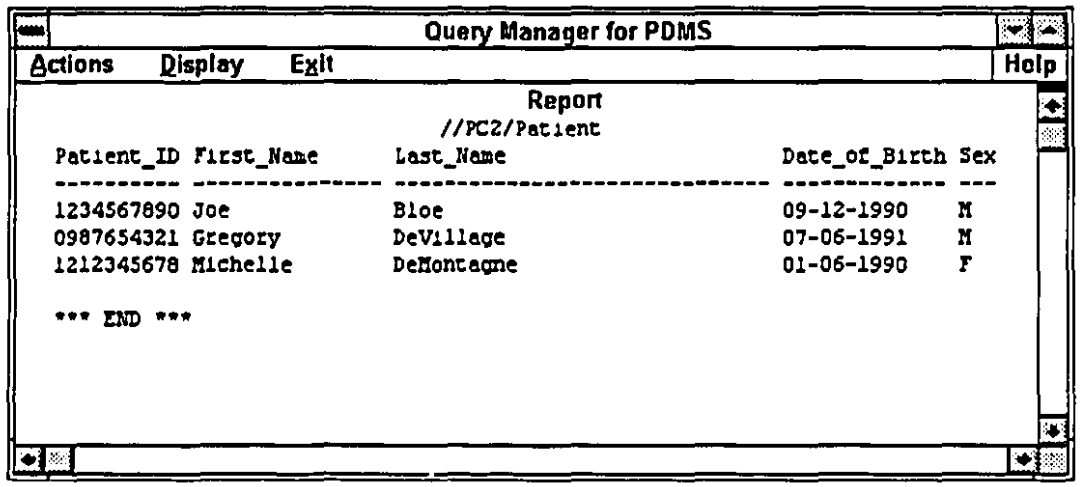


Figure 5.15: Sample Patient data retrieval.

The remote query that was performed on the *Patient* database table requested the same information, however the request was issued from a client station (PC3) to the corresponding database server station (PC2). This was accomplished by the following SQL statement:

```
SELECT Patient_ID, First_Name, Last_Name, Date_of_Birth, Sex, FROM
//PC2/Patient;
```

A sample of the remote query result is shown in Figure 5.15. Note that at that time only three patients were registered and active in the PDMS testing environment.

The queries that were performed on the *Admittance* table asked for all the information found in that table, and are presented below:

local query:

```
SELECT * FROM Admittance;
```

remote query:

```
SELECT * FROM //PC2/Admittance;
```

The asterisk "*"indicates that all the columns in that table are required. A sample of the local query result is shown in Figure 5.16.

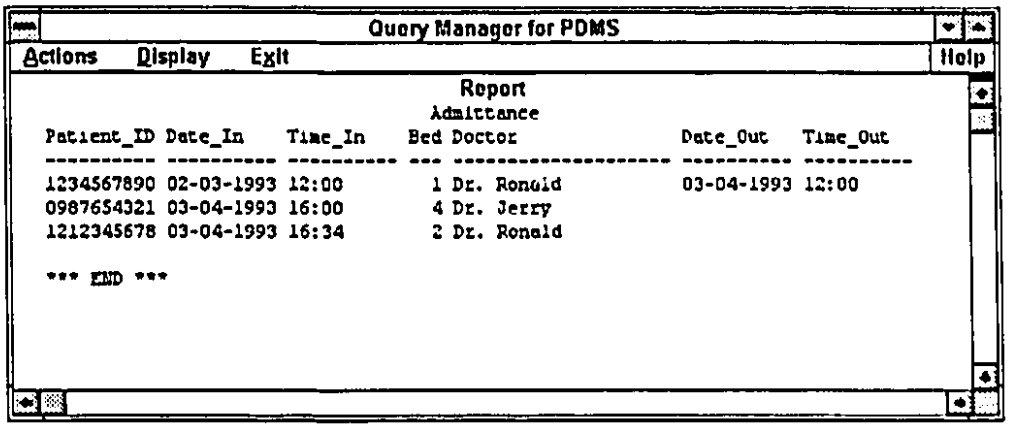


Figure 5.16: Sample Admittance data retrieval.

4. *Insert parameter data.*

The insertion transactions of parameters data were performed automatically by the data transfer process of the DLC module. These data insertions can only be performed locally, as currently intended by the design, in order to enhance the performance of the real-time DLC module.

The response times recorded by the DLC module for a single patient active in the ICU were acceptable. But for three or more active patients the results were somewhat slow due to the limited capability of PC1, an 80386 16MHz computer. However, with a 80486 66MHz CPU and a minimum of 16MB RAM computer the response time of the DLC module can be improved by an estimated factor of 3.

5. *Half hour data retrieval.*

These test conditions were carried out by querying the Parameters tables via the OS/2 Database Query Manager. The queries were executed in two ways, the first locally on PC1 and the other remotely. In both cases the response time recorded was acceptable.

The queries that were performed on the Parameters table asked for all the information stored in that table within a certain two-second period, and are presented below:

local query:

```
SELECT * FROM Parameters WHERE Time_Stamp <= "1993-03-04-
12:02:02" AND WHERE Time_Stamp <= "1993-03-04-12:02:04";
```

remote query:

```
SELECT * FROM //PC1/Parameters WHERE Time_Stamp <= "1993-03-04-
12:02:02" AND WHERE Time_Stamp <= "1993-03-04-12:02:04";
```

The 5 seconds response time of the remote query, in this case, is found to be somewhat slow since remote queries over large database tables are relatively slow. Moreover, the network communication time, over the LAN, is also a considerable factor. On the other hand, the 2 seconds response time of the local query is acceptable.

5.7 Future Extensions

The PDMS database currently operates on the 1.3 version of OS/2 under the Extended Edition of the OS/2 Database Manager Services. Migrating the PDMS to the 32 bit OS/2 2.1 can enhance its performance. Moreover, upgrading the hardware platform of the PDMS from 386 CPU computer to the 486 66MHz or 100MHz series can enhance its real-time capabilities by an estimated factor of 3 to 5 times.

Although we employed an Object-Oriented method to design the database of the PDMS, we had little choice over the implementation platform. Another research direction for the PDMS could be investigating the advantages of an Object-Oriented implementation versus a relational implementation.

Another interesting future extension to the design of the database is to encapsulate the procedures that manipulate the PDMS database in the object model. For example, displaying certain *Parameters* information or data insert and update operation. In fact,

this is one of the main reasons why OMT was chosen over an EER database design approach. This provides the database design with more flexibility, portability and reusability.

These initial tests focus on the performance of the database implementation. More extensive tests should also address other issues of a distributed implementation namely computing network traffic and anticipated workstation loading.

Chapter 6 Conclusion

This thesis identified the need of data management computerization in the ICU through a survey of types of data required and managed in an ICU and discussed the advantages that such a step could present. It also described some of the difficulties encountered in building a medical system software, namely, the high cost of software development and database maintenance due to design inadequacy and requirements misunderstanding.

The manuscript suggested modelling the database in the design phase of the medical software as a solution that reduces the cost of database maintenance and promotes understanding of the design and its requirements. Modelling also promotes the portability and reusability of the design across different hardware and software platforms as well as across similar intesive care organizations. It presented a literature survey and evaluation of data modelling techniques and applied one of them in the implementation of the database. It also evaluated the design produced by using this object-oriented database design technique, as well as the actual test results obtained after implementing this methodology on the Patient Data Management System for an Intensive Care Unit. Suggestions for future work are also proposed.

The rule-based school of data modelling is rarely applied in the current software industry. Moreover, there is scarce literature to be found on that subject. Therefore, in order to ensure the extension and reusability of the PDMS database model we employed a data modeling technique that belongs to the fact-based school of data modeling.

Both, an EER and object-oriented fact-based modeling methods serves us well in the design of the PDMS database, they both provide a coherent easy to communicate design schema as well as a "mechanized" mapping between a "clean" high level database design and a physical realizable data model. However, an object-oriented database model offers a more flexible data model which is less expensive to extend in the future, in terms of data processing (operations) and additional system functionality.

References

[Abrail, 1974] J. R. Abrail, "Data Semantics", in *Data Base Management* (Klimbie and Koffeman, eds). Amsterdam: North-Holland, 1974, pp. 1-61.

[Allen, 1991] S. K. Allen, "Selection and implementation of an automated care planning system for a health care institution", *Computers in Nursing*, vol. 9, pp. 61-68, March-April 1991.

[Andreoli and Musser, 1985] K. Andreoli and L. A. Musser, "Computers in nursing care: The state of the art", *Nursing Outlook*, vol. 33, pp. 16-21, January-February 1985.

[Aukburg *et al.*, 1989] S. J. Aukburg, P. H. Ketikidis, D. S. Kits, and B. B. Matschinsky, "Automation of physiological data presentation and alarms in the post anaesthesia care unit", in *Proceedings of the 13th Annual Symposium on Computer Applications in Medical Care (SCAMC)* (L. C. Kingsland, ed.), (Washington, D.C.), IEEE Press, 1989.

[Bachman, 1969] C. Bachman, "Data Structure Diagrams", *Communications of the ACM SIGBDP, Database*, vol. 1, No. 2, 1969.

[Bachman, 1977] C. Bachman, "The role concept in data models", in *Porc. 3rd International Conference on very Large Databases*, IEEE, New York, pp. 464-476, 1977.

[Biller and Neuhold, 1977] H. Biller and E. Neuhold, "Concepts for the conceptual schema", In *Architecture and Models in Database Management Systems*, G. Nijssen, Ed., North-Holland, Amsterdam, 1977, pp.1-30.

[Blum, 1984] B. I. Blum, "Information systems for patient care", *in Information Systems for Patient Care* (B. I. Blum, ed.), pp. 1-19, New York:Springer-Verlag, 1984.

[Booch, 1991] G. Booch, *Object-Oriented Design with Applications.* Redwood City, CA: Benjamin Cummings, 1991.

[Bradshaw *et al.*, 1984] K. Bradshaw, R. Gardner, and T. C., "Physician decision-making-evaluation of data used in a computerized ICU", *Intl J Cln Monitoring Comp*, vol. 1, pp. 81-91, 1984.

[Bradshaw *et al.*, 1989] K. E. Bradshaw, D. F. Sittig, R. M. Gardner, T. A. Pryor, and M. Budd, "Computer-based data entry for nurses in the ICU", *M.D. Computing*, vol. 6, p. 274-280, September-October 1989.

[Brodie, 1984] M.L. Brodie, "On the Development of Data Models", *Topics of Information Systems, on Conceptual Modelling (Perspectives from Artificial Intelligence, Databases, and Programming Languages)*. New York: Springer-Verlag, pp. 19-47, 1984.

[Brown, 1991] A. W. Brown, Object-Oriented Databases, Application in Software Engineering. London: McGraw-Hill, 1991.

[Chen, 1976] Peter Chen, "The entity-relationship model: toward a unified view of data", in *ACM Trans. on Database Systems*. Vol. 1 N.1, pp. 9-36, 1976.

[Coad and Yourdon, 1990] P. Coad and E. Yourdon, *Object-Oriented Analysis*, New York: Englewood Cliffs, Yourdon Press/Prentice Hall, 1990.

[Coad and Yourdon, 1991] P. Coad and E. Yourdon, *Object-Oriented Analysis*, 2nd. edn., New York: Englewood Cliffs, Yourdon Press/Prentice Hall, 1991.

[Coad and Yourdon, 1991a] P. Coad and E. Yourdon, *Object-Oriented Design*, New York: Englewood Cliffs, Yourdon Press/Prentice Hall, 1991.

[Codd, 1970] E. Codd, "A relational model of data for large shared data banks", in *Communications of the ACM*, vol. 13, pp. 377-387, June 1970.

[Collet *et al.*, 1989] C. Collet, N. Fumai, M. Petroni, S. Malowany, J. Panisset, A. Malowany, F. Carnevale, R. Gottesman, and A. Rousseau, "A patient data management system for an intensive care unit", in *Proceedings of the IEEE Pacific Rim Conference on*

Communications, Computers, and Signal Processing, (Victoria, B.C., Canada), pp. 594-597, June 1989.

[Collet *et al.*, 1990] C. Collet, L. Martini, M. Lovin, E. Masson, N. Fumai, M. Petroni, A. Malowany, F. Carnevale, R. Gottesman, and A. Rousseau, "Real-time trend analysis for an intensive care unit patient data management system", in *Proceedings of the IEEE Symposium on Computer-Based Medical Systems*, (North Carolina), pp. 337-344, June 1990.

[Dasta, 1990] J. Dasta, "Computers in critical care: opportunities and challenges", *DICP*, vol. 24, no. 11, pp. 1084-1092, 1990.

[Date, 1981] C. J. Date, *An Introduction to Database Systems* 3rd. edn. Reading MA: Addison-Wesley, 1981.

[Date, 1986] C. J. Date, *An Introduction to Database Systems*, Vol. 1, 4th. edn. MA: Addison-Wesley, 1986.

[Dawant *et al.*, 1993] B. M. Dawant, S. Uckun, E. J. Manders and D. P. Lindstrom, "The SIMON Project: A Model-Based Signal Acquisition, Analysis and Interpretation in Intelligent Patient Monitoring", in *IEEE Engineering in Medicine and Biology*, vol. 12, no. 4, pp. 82-91, December 1993.

[DeMarco, 1978] T. DeMarco, *Structured Analysis and System Specification*, Englewood Cliffs, N.J.: Prentice-Hall, 1978.

[Dyke and Kunz, 1989] R.P Ten Dyke and J C. Kunz, Object-Oriented Programming, *IBM Systems Journal*, vol. 28 no. 3, 1989.

[Ellis, 1987] D. Ellis, *Medical Computing and Applications*, Chichester, England: Ellis Horwood Ltd., 1987.

[Elmasri and Navathe, 1989] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems*. Reading, MA: Benjamin Cummings, 1989.

[Elmasri and Navathe, 1995] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems*, 2nd. edn. Reading, MA: Benjamin Cummings, 1989.

[EMBS, 1991] "Proceedings of the 13th annual international conference of the IEEE engineering in medicine and biology society (EMBS)", vol. 13, (Orlando, FL), IEEE, IEEE Press, November 1991.

[Everest, 1988] G.C. Everest, "ER Modelling versus Binary Modelling", In Proceedings of the 16th. Intl. Conf. on ER Approach, S.T. March, Ed., North-Holland, Amsterdam, 1988, pp. 63-78.

[Fairley, 1961] B. H. Fairely, "The Toronto general hospital respiratory unit", *Anaesthesia*, vol. 16, pp. 266-269, 1961.

[Fichman and Kemerer, 1992] R.G. Fichman and C.F. Kemerer, "Object-Oriented and Conventional Analysis and Design Methodologies", in *IEEE Computer*, vol. 25, no. 10, pp. 22-39, October 1992.

[Fumai *et al.*, 1991] N. Fumai, C. Collet, M. Petroni, K. Roger, A. Lam, E. Saab, A. Malowany, F. Carnevale, and R. Gottesman, "Database design for an intensive care unit", in *Proceedings of the Fourth Annual IEEE Symposium on Computer-Based Medical Systems*, (Baltimore, MD), pp. 78-85, May 1991.

[Gane and Sarson, 1979] C. Gane and T. Sarson, *Structured Systems Analysis: Tools and Techniques*. Englewood Cliffs, N.J.: Prentice-Hall, 1979.

[Gardner *et al.*, 1989a] R. Gardner, K. Bradshaw, and K. Hollongsworth, "Computerizing the intensive care unit: current status and future opportunities", *Journal of Cardiovascular Nurse*, vol. 4, pp. 69-78, 1989.

[Gardner *et al.*,1989b] R. M. Gardner, D. F. Sitting, and M. C. Budd, "Computers in the intensive care unit: match or mismatch?", in *Textbook of Critical Care* (W. C. Shoemaker, S. Ayres, A. grenvik, P. R. Holbrook, and W. L. Thompson, eds.), Ch. 26, Philadelphia: W. B. Saundres Co., 1989.

[Goldkhul and Lyytinen, 1982] G. Goldkhul and K. Lyytinen, "A language action view on information systems", in *Proc. 3rd. International Conference on Information Systems*, Ann Arbor, December 1982.

[Goldkuhl and Lyytinen, 1984] G. Goldkhul and K. Lyytinen, "Information System Specification as Rule Construction", in *Beyond Productivity: Information Systems for Organisational Effectiveness*. T. Bemelmans (ed.), Amsterdam, Holland, pp. 79-94, 1984.

[Graham, 1994] Ian Graham, *Object-Oriented Methods*, 2nd. edn. Addison-Wesley, 1994.

[Hammond *et al.*, 1991a] J. Hammond, H. M. Johnson, C. G. Ward, R. Varas, R. Dembicki, and E. Marcial, "Clinical evaluation of a computer-based patient monitoring and data management system", *Heart and Lung*, vol. 20, pp. 119-124, March 1991.

[Hammond, *et al.*, 1991b] J. Hammond, H. M. Johnson, R. Varas, and C. G. Ward, "A qualitative comparison of paper flowsheets vs. a computer-based clinical information system", *Chest*, vol. 99, pp. 155-157, January 1991.

[Hendrickson *et al.*, 1991] G. Hendrickson, J. B. Kelly, and L. Citrin, "Computers in oncology nursing: Present use and future potential", *Oncology Nursing Forum*, vol. 18, pp. 715-723, May-June 1991.

[Hilberman, 1975] M. Hilberman, "The evolution of intensive care units", *Critical Care Medicine*, vol. 3. pp. 159-165, 1975.

[Hirschhein, 1985] R. A. Hirschheim, *Office Automation: a social and organizational perspective*, Chichester Sussex; New York ; Wiley, 1985.

[Huber, 1983] G. Huber, "Cognitive style as a basis for MIS and DSS design: much to do about nothing?", *Management Science*, vol. 29, No. 5, May 1983.

[Kari *et al.*, 1990] A. Kari, E. Ruokonen, and J. Takala, "Comparison of acceptance and performance of automated and manual data management systems in intensive care units", *Intl J Clin Monit Comput*, vol. 7, no. 3, pp. 157-162, 1990.

[Kent, 1983] W. Kent, "Fact-based data analysis and design", *in Entity-Relationship Approach to Software Engineering*, C. Davis et al., Eds., North-Holland, 1983, pp. 3-53.

[Kim and March, 1995] Y.G. Kim and S.T. March, "Comparing Data Modelling Formalisms", In *Communications of the ACM*, vol. 38, no. 6, June 1995, pp. 103-114.

[King *et al.*, 1984] C. King, L. Manire, R. M. Strong, and L. Goldstein, "Data management systems in clinical research", in *Information Systems for Patient Care* (B. I. Blum, ed.), pp. 404-415, New York: Springer-Verlag, 1984.

[Korth and Silberschatz, 1991] H.F. Korth and A. Silberschatz, *Database System Concepts*. New York: McGraw-Hill, 1986.

[Kriewall and Long, 1991] T. J. Kriewall and J. M. Long, "Computer-based medical systems", *IEEE Computer Magazine*, vol. 24, pp. 9-12, March 1991.

[Larson, 1992] james A. Larson, *Intercative Software: Tools for Building User Interfaces*. Englewood Cliffs, NJ: Yourdon Press, Prentice-Hall, 1992.

[Lynngbaek and Kent, 1991] P. Lyngbaek and W. kent, "A data modelling methodology for the design and implementation of information systems", in *On Object-Oriented Database Systems*. (K. Dittrich, U Dayal and A. Buchman, eds.), Ch. 14, New York: Springer-Verlag, 1991.

[Lyytinen and Lehtinen, 1984] K. Lyytinen and E. Lehtinen, "On information modelling through illocutionary logic", in *Proc. 3rd. Annual Research Seminar on Information Modelling and Database Management*. Tamepre, 1984.

[McDonald *et al.*, 1988] C. J. McDonald, L. Blevins, W. M. Tierney, and D. K. Martin, "The regenstrief medical records", *M.D. Computing*, vol. 5, pp. 34-45, September-October 1988.

[Milholland, 1988] K. Millohand, "Patient data management systems: Computer technology for critical care nurses", *Computers in Nursing*, vol. 6, pp. 237-242, November - December, 1988.

[Mittra, 1991] Sintansu S. Mittra, *Principles of Relational Database Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1991.

[Mora *et al.*, 1993] F Mora, G. Passariello, G. Carrault and J.P. Le Pichon, "Intelligent Patient Monitoring and Management Systems: A Review", in *IEEE Engineering in Medicine and Biology*, vol. 12, no. 4, pp. 23-32, December 1993.

[Morret-Bonillo *et al.*, 1993] V. Morret-Bonillo, A. Alonso-Betanzos E. G. Martin, M. C. Canosa and B. G. Berdinas, "The PATRICIA Project: A Semantic-Based Methodology for Intelligent Monitoring in the ICU", in *IEEE Engineering in Medicine and Biology*, vol. 12, no. 4, pp. 59-68, December 1993.

[Morris, 1977] L. E. Morris, "History and ethical aspects of intensive care", in *The management of the acutely ill* (J. P. Wayne and D. W. Hill, eds.), pp. 2-3, England: Peter Peregrinus Ltd., 1977.

[Nightingale, 1963] F. Nightingale, *Notes on a Hospital*. London: Longman, third ed., 1963.

[Nijssen, 1977] G. Nijssen, "Current issues in conceptual schema concepts", In *Architecture and Models in Database Management Systems*, G. Nijssen, Ed., North-Holland, Amsterdam, 1977, pp.1-30.

[Nolan-Avila and Shabot, 1987] L. Nolan-Avila and M. Shabot, "Life without computers in the ICU, *Critical Care Nurse*, vol. 7, pp. 80-83, May-June 1987.

[OMG, 1993] Object Management Group, *Common Object Request Broker Architecture*. Framington, MA: Object Management Group, 1993.

[Petroni *et al.*, 1991] M. Petroni, C. Collet, N. Fumai, K. Roger, F. Groleau, C. Yien, A. Malowany, F. Carnevale, and R. Gottesman, "An automatic speech recognition system for bedside data entry in an intensive care unit", in *Proceedings of the Fourth Annual IEEE Symposium on Computer-Based Medical Systems*, (Baltimore, MD), pp. 358-365, May 1991.

[Prieto-Diaz and Freeman, 1987] R. Prieto-Diaz and P. Freeman, Classifying software for reusability, *IEEE Software*, Vol. 4 , No. 1, pp. 6-16, 1987.

[Roger *et al.*, 1992] K. Roger, C. Collet, N. Fumai, M. Petroni, A. Malowany, F. Carnevale, and R. Gottesman, "Nursing workload management for a patient data management system", in *Proceedings of the Fifth Annual IEEE Symposium on Computer-Based Medical Systems*, (Durham, North Carolina), pp. 216-223, IEEE Computer Society Press, June 14-17 1992.

[Rumaugh *et al.*, 1991] J. Rumbaugh, M. Blaha, W. Premmerlani, S. Eddy and W. Loresen, *Object-Oriented Modeling and Design*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[Shlaer and Mellor, 1988] S. Shlaer and S.J. Mellor, *Object-Oriented Analysis: Modeling the World in Data*. Englewood Cliffs NJ: Prentice-Hall, 1988.

[Shlaer and Mellor, 1991] S. Shlaer and S.J. Mellor, *Object Lifecycles: Modeling the World in States*. Englewood Cliffs NJ: Yourdon Press, 1991.

[Skarra and Zodnik, 1987] A. H. Sakarra and S. B. Zodnik, "Type Evolution in an Object-Oriented Database", Research Directions in Object-Oriented Programming. B. Sriver and P. Wegner (eds.), The MIT Press, Cambridge, Massachusetts. London, England, pp. 393-415, 1987.

[Smith and Smith, 1977] J.M Smith and D.C.P. Smith, Database abstractions - aggregation and generalization. *ACM Trans. on Database Systems*, vol. 2, pp. 105-133, 1977.

[Sommerville, 1989] I. Sommerville, *Software Engineering*, 3rd. edn. New York: Addison-Wesley, 1989.

[Sommerville, 1992] I. Sommerville, *Software Engineering*, 4th. edn. New York: Addison-Wesley, 1992.

[Soontit, 1987] E. Soontit, "Installing the first operational bedside nursing computer system", *Nursing Management*, vol. 18, pp. 23-25, July 1987.

[Staggers, 1988] N. Staggers, "Using computers in nursing: Documented benefits and needed studies", *Computers in Nursing*, vol. 6, pp. 164-169, July-August 1988.

[Stonebraker, 1991], M. Stonebraker, "Managing persistent objects in a multilevel store", *Electronic Research Laboratory, University of California, Technical Report* M91/16, March 1991.

[Strickland Jr., 1991] T. J. Strickland Jr., "Development of an information system to assist management of critically ill patients", in *Proceedings of the Fourth Annual IEEE Symposium on Computer-Based Medical Systems*, (Baltimore, MD), pp. 70-77, May 1991.

[Subramanian, 1989] S. Subramanian, "OB information management system: A microcomputer solution", in *Proceedings in the IEEE Engineering in Medicine and Biology Society*, (Seattle, WA), pp. 2005-2006, November 1989.

[Suko, 1976] M.E. Suko, "NIAM as a detailed example of the ANSI SPARC architecture", In *Modelling in Database Management Systems*, G. Nijssen, Ed., North-Holland, Amsterdam, 1976, pp. 73-94.

[Sukuvaara *et al.*, 1993] T. Sukuvaara, M. Sydanmaa, H. Nieminen, A. Heikela and E. M.J. Koski, "Object-Oriented Implementation of an Architecture for Patient Monitoring", in *IEEE Engineering in Medicine and Biology*, vol. 12, no. 4, pp. 69-81, December 1993.

[Tachakra *et al.*, 1990] S. Tachakra, D. Potts, and A. Idowu, "Evaluation of a computerized system for medical records in an accident and emergency departments", *Intl J Clin Monit Comput*, vol. 7, no. 3, pp. 187-191, 1990.

[Teorey *et al.*, 1986] T.J. Teorey and J.P. Fry, "A logical design methodology for relational databases using the extended entity-relationship model", *ACM Computing Surveys*, vol. 28, June 1986.

[Thull et al., 1993] B. Thull, H.J. Popp and G. Rau, "Man-Machine Intercation in Critical Care Settings", in *IEEE Engineering in Medicine and Biology*, vol. 12, no. 4, pp. 42-49, December 1993.

[Ullman, 1981] J.D. Ullman, *Principles of Database Systems*. Maryland: Computer Science Press, 1981.

[Wasserman *et al.*, 1989] A.I. Wasserman, Pircher, P.A. and R.J. Muller, "An Object-Oriented Structured Design Method for Code Generation", *Software Engineering Notices*, Vol. 4, No. 1, pp. 32-55, January 1989.

[Watt *et al.*, 1993] R. Watt, E. S. Maslana and K. C. Mylrea, "Alarm and Anesthesia: Challenges in the Design of Intelligent Systems for Patient Monitoring", in *IEEE Engineering in Medicine and Biology*, vol. 12, no. 4, pp. 34-40, December 1993.

[Weil *et al.*, 1989] M. H. Weil, M. V. Planta, and E.C. Rackow, "Critical care medicine: Introduction and historical perspective" in *Textbook of Critical Care*. (W.C. Shomemaker, S. Ayres, A. Grenvik, P.R. Holbrook and W.L. Thompson, eds.), Ch. 1 , pp. 1-5, Philadelphia: W.B. Saundres Co., 1989.

[Weiss and Page-Jones] S. Wiess and M. Page-Jones, *Synthesis: An Object-Oriented Analysis and Design Method*. London: Macmillan, 1991.

[Whitinh-O'Keefe *et al.*, 1985] Q. Whitinh-O'Keefe, D. Simborg and W. Epstein, "A computerized summary medical record systems can provide more information than the standard medical record", *JAMA*, vol. 254, pp. 1185, 1985.

[Winston, 1984] P.H. Winston, *Artificial Intelligence*, 2nd. edn.. Addison-Wesley, 1984.

[Won, 1990] Kim Won, *Introduction to Object-Oriented Databases*. Cambridge, Mass.: MIT Press, 1990.

[Won, 1995] Kim Won, *Modern Database Systems: The object-model, interoprality, and beyond*. New York, N.Y.: ACM Press: Addison-Wesley, 1995.

[Yourdon and Constantine, 1979] E. Yourdon and L.L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[Zumd, 1979] R. Zmud, "Individual differences and MIS success: a review of the empirical literature", *Management Science*, No. 25, October 1979.

Appendix A OMG Terminology

Term	*Specialisations*
Value	Object
	Non-Object
	Concept
Non-object	Relationship
Concept	Modelling concept
	Object model concept
Modelling concept	
	Strategic modelling concept
	Analysis modelling concept
	Implementation modelling concept
	Deliverable
	Activity type
	Technique
Object model concept	Rule concept
	Objet behavioural concept
	Group and view concept
	Object structure concept
Rule concept	Constraint
	Assertion
Object behavioural concept	Operation
	Event

	State
	Transition
	Message
Message	Request
Group and view concepts	Diagram
	Schema
	Quality concept
	Strategic model G&V concepts
	Analysis model G&V concepts
	Design model G&V concepts
	Implementation mode G&V
	Concepts
Object structure concept	Attribute type
	Object type
	Relationship type
Object type	Strategic model object type
	Analysis model object type
	Design model object type
	Implementation model object type
Relationship type	Association
	Aggregation
	Specialisation
	Instantiation
	Usage