A Formal Specification of Electronic Patient Record Processing for an Integrated Distributed Health Care System

by

Srinivasan Sampath

A dissertation submitted in partial satisfaction of the requirements for the degree of

Master of Science

Department of Computer Science

Faculty of Graduate Studies
University of Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

A FORMAL SPECIFICATION OF ELECTRONIC PATIENT RECORD PROCESSING FOR AN INTEGRATED DISTRIBUTED HEALTH CARE SYSTEM

BY

Srinivasan Sampath

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University

of Manitoba in partial fulfillment of the requirements of the degree

of

Master of Science

Srinivasan Sampath © 2003

Abstract

Using a distributed healthcare application, doctors can diagnose and treat patients based not only on the limited information available to them at that time but also based on each patient's entire medical history. An electronic patient record is at the core of such a system. Making such a record available via a distributed system means that even if a patient requires emergency care in a remote location and is unconscious the patient's information will be available so that healthcare personnel can offer the best possible treatment.

The motivation for the use of a distributed electronic patient record is the current complexity of maintaining and accessing patient records and other medical information resources scattered across many locations in an unstructured manner. This results in the inefficient use of both healthcare facilities and personnel as well as, sometimes, leading to sub-optimal care. Changing from paper to electronic media offers the potential to avoid these inefficiencies as well as improving overall healthcare. Doing so, however, is a complex and potentially life-critical process. The patient's record's confidentiality should not be compromised; and the right mapping of the patient to patient's medical records must be guaranteed. This makes a formal specification of any such electronic patient record essential to guarantee that there is no information loss (relative to existing systems) that could endanger patients. Current implementations of electronic patient record systems seem generally well-designed and organized but lack a formal specification of their underlying operations. These implementations have not been verified mathematically and therefore are relatively untrustworthy in the eyes of many practitioners. Since the healthcare domain is safety-critical, we should not rely on human intuition in the development of such systems.Thus, the goal of this research is to present a formal specification of an electronic patient record and how it interfaces to the software that uses it in an integrated distributed healthcare system. This research provides a good case study for learning and transferring skills in formal software design. The lessons learned from applying formal methods also directly benefits developers producing such systems.

Acknowledgements

I am deeply grateful to my supervisor, Dr. Sylvanus Ehikioya, for taking me on as a student in the first place. I really thank him for his encouragement, appreciation, personal guidance and the constructive comments he provided during the completion of this thesis. Throughout the thesis-writing period he provided sound advice, great teaching and a lot of ideas for improving this thesis. I would have been lost without him. Thank you, Sir.

I am grateful to Dr. Peter Graham, and Dr. Jose Rueda for their insight, for serving as my committee members, and for reviewing the thesis manuscript. I am especially thankful to Dr. Peter Graham for providing useful comments and excellent editing of the initial thesis proposal.

Special thanks to Dr. Thulasiraman for providing me great encouragement and guidance.
I am indebted to my many friends for helping me get through the difficult times, and for all the emotional support, entertainment, and caring they provided during my stay in Winnipeg.
My special gratitude is due to my sister and my brother in-law for their loving support.
Lastly, and most importantly, I wish to thank my parents, Sampath and Shanthi. They raised me, supported me, taught me, and loved me. To them I dedicate this thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"Computers are magnificent tools for the realization of our dreams, but no machine can replace the human spark of spirit, compassion, love, and understanding." – Louis Gerstner

Most present health care systems employ the traditional method for maintaining patient records using paper and pen. Paper has been an effective and simple medium for recording interactions between patients and their doctors and other health care workers. Paper allows the doctor to write down patient information and his/her views easily in a format the doctor is comfortable with [Po94]. The use of paper also gives the doctor more freedom to express his/her views more conveniently than with existing computer systems. Paper also supports reasonable portability as it can be carried with the patient from one health care practitioner to another. The problems with paper-based systems are, however, many. The impact of losing a patient's paper record is severe. For example, important information about the patient may be missing during his next interaction with another doctor. Paper also suffers from being illegible at times, is often poorly formatted, and relatively cumbersome to deal with [Di97]. Finally, it is not a convenient format to distribute quickly, on-demand.

One serious drawback of paper-based systems is that they are unstructured and not amenable to electronic access. Hence, maintaining files of records and retrieving the contents of such records are sometimes difficult tasks. Since a patient's history should be reviewed before recommending any treatment or medication, patient records must be available prior to treatment. Further, doctors should be able to retrieve information

about a patient from remote locations.
Computers were first introduced into the health care system to improve administrative practices rather than to improve patient care [Lu96]. Clinical computing, as compared to administrative computing, is centered on the patients [Ha94]. Computers, which have made marked impacts almost everywhere in modern life, have affected the documentation of clinical care very little [Lu96]. There are clear benefits to be gained by exploiting computer-based automation in providing an integrated healthcare service, and at the core of this is the electronic patient records system.
An electronic patient record contains the patient's medical history, giving comprehensive information about the patient. Electronic medical records attempt to transfer the information about a patient that is currently maintained on paper (and other media such as x-ray film) to a computer. Patient records are "a place where we store everything we need to know about a patient and where we keep this information forever so we can do a great job of delivering care" [Sa94].
Current electronic medical record systems [Bi95, Ku99, NA99] contain such information as the patient's history, family history, past diagnoses, test results, known allergies, immunization facts, health and psychological problems, medications prescribed and responses to treatments, etc. An electronic medical record may also include information about instances of hospitalization for the patient, extra services the patient received, the amount that was spent when the patient was treated and details of the patient's interactions with other health care professionals such as pharmacists. In the future, electronic medical records may also include such things as online imagery (e.g., X-rays) and video (e.g., a telemedicine session or a functional MRI record) [NA99].

1.1 Motivation

An integrated distributed health care system enables a patient to get all aspects of medical care from a single system, regardless of location. In such systems, wide ranges of services are supported including: emergency care, inpatient hospitalization, outpatient follow-up care and rehabilitation, as well as ongoing health education and outreach services. Further, patient documentation resulting from all such services is maintained in the patient's electronic medical record, which is subsequently available to all service providers. By integrating patient information from different sources in this way, both operational cost effectiveness and improved health care delivery can be achieved.

There are a number of challenges to be overcome before electronic medical records (and the integrated health care systems they will support) can be accepted in the medical community and by the public at large. These challenges include both technical and perception issues.

One of the biggest technical problems in computerized data management is incompatibility between heterogeneous information systems. Database systems that are developed using different DBMSs are classified as heterogeneous databases. Such databases are common in healthcare systems now and will undoubtedly be a part of any fully integrated healthcare system. Heterogeneous databases pose real problems for integrating and retrieving patient records since the structure and content of patient records from different, existing service providers (e.g. pharmacy vs. hospital records) are not uniform. Because accessing these records requires only electronic authorization, the security and privacy of the records could also be compromised. Other technological problems include the need to reliably interconnect all health service providers and issues of high-availability related to both medical data and service provider interconnection. While these are important problems they are not the focus of this thesis.

An even greater challenge in the conversion from paper to electronic medical records is one of perception by the medical community and the public generally. There is a good deal of distrust of many computerized systems outside the Computer Science and Engineering communities. This has arisen, primarily, due to inaccuracies in the design and implementation of past systems, which have lead to serious operational flaws. This is particularly problematic for large-scale systems where the complexity is high and in life-critical systems where there is no margin for error.

The public's lack of trust may be at least partially addressed by the use of formal design techniques. For example, formal specifications can be used during the media change from paper medical records to electronic ones to guarantee that there is no information loss thus providing assurance that the new system will, at least, be no worse than the old one. Although existing implementations of computerized medical systems may seem well-designed they lack a formal specification of their underlying operations and their implementations have not been formally verified. Since the healthcare domain is safety-critical, we cannot rely on human intuition alone in the development of such systems. The goal of this research is therefore to present a formal specification of an electronic patient record and how it interfaces to the software that will use it in an integrated distributed healthcare system.

1.2 Problem Definition

To investigate the application of formal specifications to electronic patient record systems, three things are done:

1. determine what information should be included in a patient record and where/how those records will be used in an integrated distributed health care system,

2. formally model the design of those aspects of a health care system that directly impact electronic patient records and use tools to illustrate the correctness of the model developed, and

3. implement certain parts of an integrated health care system (a subset of those that will interface with electronic patient records) to show that the formal model is, in fact, useful in developing such applications.

Significant work must be done to accurately assess what information should be contained in an electronic patient record and how such records would be used in an integrated distributed healthcare system (this is requirements gathering). An electronic patient record should certainly include the contents of all existing, paper-based systems but must also be expanded to include information necessary to support such records' use in an integrated distributed healthcare system. Note also that an electronic patient record will eventually contain information relevant to all users of an integrated healthcare system (doctors, pharmacists, physiotherapists, health insurance providers, etc.). To assess the needed additions to electronic patient records for such an integrated healthcare system, their use in such a system must be carefully considered. To do this also requires an assumed model/architecture for an integrated distributed healthcare system. (Such an architecture is proposed in the following section.) Electronic patient records will be used by many parts of an integrated healthcare system and the processing performed on them by each part of the integrated system will have to be specified. Systems that will use electronic patient records include (among many others):

- Patient system flow,
- Health care resource management,
- Patient care and treatment, and
- Billing for health care resources utilization.

1.3 Brief Outline of the Thesis Work

Once a set of requirements has been formulated, the use of electronic patient records in the processing described must be formally modeled (the formal and informal requirements are explained in detail in Chapter 3 - *Specifications*). This modeling was done using the UML [Oe99, Dy98] and Z [Da96, Sp92] notations. These two notations are complementary and their use together provides more detail than would be possible with just UML or Z alone. Z-EVES [Sa95] proves the consistency of the Z part of the specifications developed. Doing the most extensive possible verification is important since the specification will eventually be used to build a system that is safety-critical (e.g., medical doctors may use such a system to guide courses of treatment of a patient so errors could be life threatening).

Finally, a proof-of-concept implementation was developed on a select subset of the system's specification to illustrate the applicability of using such formal specifications in this problem domain. This allows the production of equivalence mappings between the proven formal specification described and the prototype implementation. This further increases confidence in the final implementation and thereby in the approach of using formal specifications for the design of integrated distributed health care systems.

1.4 Limitations

The thesis provides outlines for integrating various health care departments necessary to model an electronic medical record. There were issues that require more concentration and collection of intrinsic details, e.g., the main objective of the thesis was to identify the participating departments and the transactions that are performed in these departments that get into the medical record of a patient. The actual design of the proposed system should be more efficient to accommodate various security features and the functionality of each department are generalized to save and retrieve information, while the actual design must be customized to suite the individual departments. Since the focus was on identifying the transactions in these departments, the actual design of the database is not specified (though the transactions are specified). Further, verification and validation of a user is based on the login name and password (in the real world this would require more rigorous encryption strategies).

1.5 Contributions

The research provides a good case study for learning and transferring skills in formal software design. To the best of our knowledge, no formal techniques have been applied to the design of health care systems although numerous commercial and academic prototypes are available. Since this is a safety-critical domain where correctness is paramount, experience and lessons learned from applying formal methods in this thesis can directly benefit developers working on producing such systems.

1.6 Organization

The rest of this thesis is organized as follows. Chapter 2 (*Literature Survey*) provides some of the relevant literature required for pursuing this research. A formal specification of the health care system is described in Chapter 3 (*Specifications*). Chapter 4 (*System Design*) describes an integrated health care system architecture, including the different object models needed, developed using UML. The implementation of the system is described in Chapter 5 (*Object Model and Implementation*). Finally, conclusions and future work of the thesis are presented in Chapter 6 (*Conclusions and Future Work*)

Chapter 2

Literature Survey

"It is difficult to say what is impossible, for the dream of yesterday is the hope of today and the reality of tomorrow." - Robert H. Goddard

Introduction

There has been significant research demonstrating the use of computer-based medical records and the advantages in using them. Those research efforts focus on issues like creating and maintaining electronic medical records, designing appropriate user interfaces, scheduling mechanisms, transaction processing in the health care domain, etc. I briefly discuss some of the most relevant related research below.

2.1 Use of Formal Notations

The use of formal methods in software design and development generally tends to increase the reliability of systems produced. Formal methods are useful in exploring system design alternatives, provide a general approach for generating test data, and help to guarantee that the design and implementation of a system are free from errors. The need for and advantages of formal specification have been demonstrated repeatedly (e.g., [Eh00, Eh01, He97, Ho95]) in non-medical domains. Helke et al. [He97] explain that the application of formal notation(s) during the initial stages of software development helps in improving the quality, and reduces the cost, of the application

software. Further, Horcher [Ho95] demonstrates the success of formal specifications in deriving integration test cases that satisfy the requirements specification of an application. Finally, according to Ehikioya [Eh00, Eh01], formal modeling leads to "correct, fail-safe, and robust transaction-processing environments" (as are typical of patient record processing in an integrated distributed health care environment). To employ formal specification successfully in the software lifecycle requires formal training and education. The notations used in formal methods are intimidating, which makes software engineers to skip this phase totally. Also, formally specifying a model consumes significant resources leading to increased costs initially.Barden et al. [Ba92] document the problems of using Z by the industry practitioners.
Formal specifications rely on the use of certain notations (and tools). The use of UML [Oe99, Ev98, Ho99] in designing and modeling software is discussed in the literature. According to Oestereich [Oe99], "The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system". The usefulness of UML to model software systems has been discussed in [Ev98, Ho99]. For example, Evans et al. [Ev98] show the advantages of designing a software system using UML with rigorous reasoning techniques. Hofmeister et al. [Ho99] describe the benefits in designing the software architecture of a system in UML.

2.2 Medical Record Formats

The use of the SOAP (Subjective Data, Objective Data, Assessment, Plan model) in creating and maintaining patient's medical records has been demonstrated in [Po94, Yv92, Po00]. In SOAP, the doctor starts by viewing a patient's medical history and then the doctor observes the characteristics of the patient's problem. From the observed characteristics, the doctor determines the diagnosis and finally plans any treatment and instructs the patient on any follow up visits. In [Po94] doctors record their opinions in three different ways: circle, underline or strikethrough the words in the list. The doctor can write notes on information about the patient. The system, PureMD [Yv92], automates hospital services and decreases the amount of written work, which facilitates the process of information retrieval and data entry.

The objective of Potamias, et al [Po00] in developing the Patient Clinical Data Directory (PCDD) was to deliver an "encounter-centered" view of the medical record. An encounter-centered view of medical record is the abstracted history of a patient's encounter with a health care provider over a period of time. To provide a consistent view of locating and accessing patient information, PCDD supports an uniform interface called the PCDD Update Interface.

2.3 Encounter based Medical Record

Goble and Crowther [Go00] introduce the idea of generating a medical record from a series of existing patient records (essentially the integration of multiple records into one). This approach supports multiple autonomous (i.e. non-integrated) healthcare systems, each providing a part of an integrated medical record, and then describes how they may be integrated. No attempt is made, however, to formally specify either the resulting patient records or the integration process.

2.4 Database Integration in Medical Domain

The needs and ways of computerizing hospitals and integrating various departments in hospitals have also been discussed [Po00, Ev99, Ga91, La98]. Among these, [Ev99, La98] support ways for updating and editing patients' medical records at the point of encounter with the medical personnel. In Potamias, et al [Po00] "autonomous information systems" are integrated to provide various views of a medical record to authorized users. In this research, I develop specifications and implement (integration is reflected in the implementation) respective parts of the system, which are necessary for capturing patient flow in a health care unit.

2.5 Applications of the Internet in Medical Systems

The use of the World Wide Web (WWW) as a medium for creating, accessing, and maintaining patient medical records is discussed in [Ko96, Ku99, Le99]. Because the Internet is widely available, it provides the necessary infrastructure to permit distributed access to patient records in an integrated healthcare system. Due to the sensitive nature of patient records, however, adequate security features such as password-secure access, logging of transactions and cryptographic technologies must be provided. Some of these approaches are discussed in [Ku99, Le99]. Recently, [Eh02] describe mechanisms that support the ubiquitous and efficient exchange of electronic medical records data across multiple heterogeneous environments. In my specification, I will model the retrieval of information across the Internet as well as the provision of basic security via login-based user verification.

2.6 Use of Agents in Medical Systems

A common theme that has recently emerged in medical systems automation is the use of agents. For example, Miksh et al. [Mi97] employ "adaptive agents" to track health-related information about patients. Adaptive agents interpret data from the various departments where a patient's health information is maintained and based on the information observed, patient's medication is determined. Hannebauer et al. [Ha99] observe that most existing systems lack coordination among departments thereby leading to "sub-optimal patient throughput and resource usage". They propose "composable agents" that can exchange information dynamically to support the dynamic allocation of resources. The use of agents in hospital operations scheduling is also discussed in the literature. For example, Decker et al. [De98] use the Generalized Partial Global Planning (GPGP) [De92] approach where an agent constructs its local view of the structures and relationships for a task and can then receive additional information from other agents to increase the accuracy/detail of its view. The use of agents for patient record processing will be considered only briefly in this thesis.

2.7 User Interfaces

The problems associated with designing suitable user interfaces for medical applications have been one of the many reasons for the slow adoption of computerized medical records. This problem has also been extensively studied [Yv92, La98, Wi90, Di95, Pl98]. While it is evident from the literature that the design of user interfaces for effective and convenient access to medical records is an area of ongoing research this problem will not be directly considered in my thesis (only a simple user interface will be specified).

2.8 Summary

Based on this review of the literature, it appears that formal methods have not been used in the design and implementation of an integrated distributed health care system despite the safety-critical nature of such systems. Thus, the research proposed herein is, to the best of my knowledge, an original contribution to the field of software engineering.

Chapter 3

Specifications

"The job of formal methods is to elucidate the assumptions upon which formal correctness depends." - Tony Hoare

The first step in developing a system is to have a sound understanding of the application domain and to resolve any uncertainties that might exist concerning the functionality of the system. Present medical record systems lack formal specifications for their implementations. Some of the important features of formally specifying software systems in general include [Jo94] :

- Formal specifications describe *what* the software to be developed does rather than *how* we develop the software. That is, it defines what a system outputs when certain inputs are given with no account about how the outputs were derived.

- Notations used in formal languages and notations used in formal verification of software are tightly coupled which makes verification of a product specified in a language like Z easier.

- Specifications expressed in formal notations can be assured of consistency.

- Formal specifications can be considered to be "high-level nonprocedural programming language" [Jo94]. This line of thinking is useful especially when we consider the specifications as a prototype for the software to be developed. Using formal specifications, misunderstandings can be detected early.

- Tool support for formal specifications helps in automated analysis and helps reducing intuitions (i.e, informal specification of software would still require some thinking while information specified formally is analyzed automatically and also a tool supports ways for locating incompleteness, repetition and incorrect organizations)

Additional features are provided in [Eh97, Eh99, BH95, So92, Ro01].
Software systems, though generally versatile, are often prone to errors and incurs high cost for repair [Ke97]. The main reason for the failure of such systems is commonly the lack of a strong foundation for the development of the software. This illustrates the importance of applying formal methods in designing software.

3.1 Informal Operational Requirements

This section summarizes some of the informal requirments of a health care application, which could be eventually implemented based on the formal specifications.

- Health care applications should be easy to use and the navigation should be simple because typical users are not computer savvies. A Distributed and Integrated Health Care Application (DIHA) integrates various departments within a hospital, accordingly the interfaces have to be designed so that they best suit the individual departments. However, there are also situations when a user in one department has to access information in other departments. In such scenarios, a user must not be allowed to search for the operation the user wishes to perform. Hence, though developed as interfaces for different departments the overall interface design has to be standardized.

- Health care applications support a multi-user environment, where the users could be geographically distributed as in a hospital. Thus, a DIHA should allow users to execute queries across the network. A hospital environment supports various kinds of users like doctors, nurses, pharmacist, laboratory technicians, frontdesk users and system administrators. Typically, these users are not neighbors. Hence, particular users should be able to get information from other kinds of users.

- Health care applications supports access only to authorized users who can have access to patient's records. A medical record is a document that contains a patient's medical history. Such documents have to be safeguarded so that unauthorized people do not access them. Therefore, access-oriented security mechanisms become necessary.

- Distributed health care applications should support heterogeneous databases. Information about patients is derived from various hospitals/departments. These hospitals/departments may not all necessarily use the same database management system. Thus, the application should support heterogeneity in databases.

- Stopping one of the functional processes in the application should not halt the application altogether. Since the field of health care is critical, patient's medical information should be made available at all times to authorized users. The developed system should provide both reliability and consistency.

- Health care systems should consider temporal issues like the real-world time a transaction occurs, past and future plans, the sequence of states in which events may take place (e.g., a supply order placed, bill payment, etc.) and the time periods (e.g., scheduling).

- Cyclic tasks exist in health care applications and must be supported. For example, a patient may visit a doctor who refers him to have a laboratory test performed. After the laboratory test is done, the patient returns to the doctor.

- Patients are the major players in health care and they are mobile. As a result, a health care application should be able to transfer information about a patient to another health care organization that participates (number of health care organization can exchange information) in information sharing during emergencies and at other times. That is, the electronic information should be transferred to the hospital where the patient is receiving emergency care. Also, electronic communication (e.g. via email) of patient medical record information between a participating and a non-participating health care centers should be possible in periods of acute emergency.

- Overall system performance should be acceptable, e.g., should offer good response times, etc.

3.2 Functional Requirements

One of the fundamental goals of this thesis is developing an Electronic Medical Record of patients based on patient-medical personnel encounter. To develop such encounter based medical records, the system must provide ways to:

- Create and maintain electronic patient medical records.
- Record a patient's problem according to the patient's description.
- Facilitate recording of encounter notes at the place where the patient is medicated or diagnosed. This decreases the time of data entry as well as the possibility of misinterpretation, which might exist when patient information is updated at a later time by a third party.
- Provide special options for viewing the medical record (e.g., support zooming on images).
- The application should be capable of generating the medical record with information from various other participating medical institutions. A patient can receive care from any number of health care centers. If there are ways of communicating between the participating health care centers then redundant operations like laboratory tests, observation, etc, can be avoided. Effective communication among the participating health care centers can result in improved care.
- A report of a patient's medical record should be developed. Users of medical records are more bothered about the information in the medical record rather than the source, so users should not be allowed to search for patient related information.

3.3 Task Specific Requirements

DIHA integrates operations in different departments in a hospital into a single application. In each of the departments various kinds of operations are performed to record facts in the patient's medical records. This gives rise to the following requirements:

- Every user in each department has to be authenticated for security.
- DIHA should identify the various departments that would use the application e.g., frontdesk, medical personnel, pharmacists, and laboratory technicians. A number of departments coexist in a hospital. The departments support various kinds of users (e.g., doctors and nurses fall in to the department that delivers care to patients). Then it becomes a definite need to identify the various departments and the users in those departments.
- DIHA should allow users to insert, search, update, and change information regarding patients in their own domain. For example, a user at the frontdesk can perform the above operations in the particular domain (e.g., doctor's domain, pharmacist domain, etc) with no control over transactions that affect the database of a different domain (e.g., that of a doctor).
- In addition to the above operations, frontdesk users can schedule patients for appointments with doctors. Scheduling is an important function in any domain. In a safety critical domain like health care, resource scheduling is a must for efficient utilization and monitoring of resources.
- Doctors can have full access to the medical records of the patients maintained in the hospital that the doctor works in. This ensures that doctors can view the entire information about patients in the hospital. This includes the treatment and medication of the patient by another doctor.
- A search by a doctor about a patient results in a number of instances that a patient received care. Doctors should have the freedom to choose and view any of the displayed instances that interests the doctor.

- Doctors can also view the record of a patient in the other participating hospitals. This ensures that a doctor can gather more information about a patient when needed. Health care organizations can determine users who can access information between organizations.

- Doctors should be able to prescribe medications. The application aims at reducing the amount of written work. Hence, doctors should be given the option of performing all the operations they would normally do using pen and paper electronically. Prescription writing is one of them.

- Users in the pharmacy department should be able to do periodic updates of the stock and the expiry date of medicines. Information in the database of a pharmacy is one of the databases in which information has to be updated everyday. Hence, users in the pharmacy department should be able to reflect changes (e.g., refilling, inventory updates, etc) in their database(s).

- Users in the laboratory also perform basic operations. These users should be able to transfer information about any test results (e.g., x-ray reports, blood reports, etc) in the best suited way to doctors (and/or the appropriate requester(s)). Though several customized test might be performed and the forms for these test would be different, I developed a generic form which captures the type of test and the result of the test.

- An important requirement of this application is sharing of information among participating hospitals. This is of prime importance as this helps in requesting services including operating theaters, wardbeds, etc of one hospital by another.

- Granting privileges to add and maintain users and devices in the health care domain by authorized users is a key requirement. Since the computers in a hospital are connected through a network, system administrators in the domain becomes absolute necessary. They are responsible for authorizing access to the database and monitoring the use of the privileges (because this is not a direct requirement of this thesis, I would only implement parts to manage users and their privileges).

- Whenever patients receive regular care for a particular ailment, then these treatments can be classified as periodic treatments. Periodic medications should be maintained automatically.

- DIHA should be able to generate different kinds of reports. These reports are basically the output of interactions between a patient and medical personnel or could be a result of any transaction (e.g., they might be a drug order form, doctor prescription, insurance provider's template, etc.).

The information thus gathered from the above requirements is used to decide which parts of the system can be formally specified (since Z is not intended to do everything). Issues like concurrency are often ignored and the design of user interfaces is difficult to achieve using Z. So, formal specifications that I develop reflect the various transactions that could happen in the health care domain with respect to patients and the UML models reflects various process chains that happen in the identified departments.

3.4 Formal Specifications

This section presents a formal specification of the transaction processing of a patient medical record in a hospital. I start the specifications by formally defining a basic type *STRING* to represent alphanumeric data type.

$[STRING]$

The system should generate various kinds of messages to display results of an attempted transaction. The following definition, *Message*, shows some of the possible messages.

$$\begin{aligned} Message ::= {} & PatientAdded \mid PatientExists \mid PatientdoesntExist \mid OverFlow \\ & \mid NoAppointmentsAvailable \mid AppointmentCancelled \mid Failure \\ & \mid Available \mid NotAvailable \mid AppointmentAvailable \mid Success \\ & \mid WrongParameters \mid DrugExists \mid RunningOutofStock \\ & \mid Updated \mid DatabaseWillbeClosed \mid PasswordUpdated \\ & \mid DatabaseNotOpen \mid Cancelled \end{aligned}$$

Users in the health care domain, fundamentally share some common demographic details. These users are modeled as *Person* in the domain. For obvious reasons, all person in the health care have a contact person, *Contactname*.

$$\begin{array}{|l}
\hline \mathit{Person} \\
\mathit{name} : \mathit{STRING} \\
\mathit{SIN}, \mathit{Phone}, \mathit{ContactPhone} : \mathbb{N}_1 \\
\mathit{address}, \mathit{Contactname} : \mathit{STRING} \\
\hline
\end{array}$$

A *Patient* is a person who also has other attributes in addition to the attributes of a *Person*. Hence, a patient inherits the attributes of a *Person* along with other special attributes.

$$\begin{array}{|l}
\hline \mathit{Patient} \\
\mathit{Person} \\
\mathit{pid} : \mathbb{N}_1 \\
\mathit{insurancedetails} : \mathit{STRING} \\
\hline
\end{array}$$

To preserve the individual identity of patients, one or a combination of attributes should be combined to make patient's record unique. Hence a particular patient's name should be associated with only one attribute of the patient.

$$\forall x, y : \mathit{Patient} \bullet x \neq y \Rightarrow x.\mathit{pid} \neq y.\mathit{pid}$$

Patients must be added to the patient database before they can receive care. So, I first define a state schema that captures the list of patients in a hospital.

$$\begin{array}{|l}
\hline \mathit{PatientDB} \\
\mathit{PatientList} : \mathbb{P}\, \mathit{Patient} \\
\hline
\end{array}$$

The following schema captures the process of adding a patient to the list of patients treated in the hospital.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{AddPatient}} \\
\Delta \textit{PatientDB} \\
\textit{PatientList}, \textit{PatientList}' : \mathbb{P}\, \textit{Patient} \\
\textit{NewPatient}? : \textit{Patient} \\
\textit{message}! : \textit{Message} \\
\hline
\textit{NewPatient}? \notin \textit{PatientList} \\
(\forall x : \textit{Patient} \mid x \in \textit{PatientList} \bullet x.\textit{pid} \neq \textit{NewPatient}?.\textit{pid}) \\
\textit{PatientList}' = \textit{PatientList} \cup \{\textit{NewPatient}?\}\textit{message}! = \textit{PatientAdded} \\
\hline
\end{array}$$

It is very likely that patients' information could change and there could be a need to update them. To do this the particular patient's record has to be retrieved to capture the changes. This search can either be based on the unique patient identity number (for efficient retrieval) or, if the patients do not have access to them, then it can be through their name (when search is done by names a list of all patients' by that name are generated).

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{SearchbyId}} \\
\Xi \textit{PatientDB} \\
\textit{patId}? : \mathbb{N}_1 \\
\textit{Response}! : \textit{Message} \\
\hline
(\exists x : \textit{Patient} \bullet x.\textit{pid} = \textit{patId}?) \\
\textit{Response}! = \textit{Success} \\
\hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{SearchbyName}} \\
\Xi \textit{PatientDB} \\
\textit{patName}? : \textit{STRING} \\
\textit{Response}! : \textit{Message} \\
\hline
(\exists x : \textit{Patient} \bullet x.\textit{name} = \textit{patName}?) \\
\textit{Response}! = \textit{Success} \\
\hline
\end{array}$$

The process of searching for a patient by name or the patient number is defined as:

$$\textit{PatientSearch} \mathrel{\widehat{=}} \textit{SearchbyId} \lor \textit{SearchbyName}$$

A patient generally visits a hospital to receive care where a patient consults a doctor. We need to formally model the resource *Doctor*. A *Doctor* is inherently a *Person* with a unique identifier, *drid*, and domain speciality.

$$\begin{array}{|l}
\textit{Doctor} \\
\hline
\textit{Person} \\
\textit{drid} : \mathbb{N}_1 \\
\textit{Specialization} : \textit{STRING} \\
\hline
\end{array}$$

To show that every doctor in the domain is unique the following should always be true.

$$\forall x, y : \textit{Doctor} \bullet x \neq y \Rightarrow x.\textit{drid} \neq y.\textit{drid}$$

Date and Time

Because *DATE* can assume various standards in various software, we define *DATE* as a basic type.

$$[\textit{DATE}]$$

TIME is defined as a natural number to enable us to manipulate time entities.

$$\textit{TIME} == \mathbb{N}_1$$

Next I specify operations relating to rooms in a hospital. Although a room is a resource, in this section we isolate it to give it greater attention.

Rooms

Hospitals have a finite number of rooms, which are available for patients' care. A hospital maintains a database to record the assignment of *Rooms* to *Patients.* A room typically has a number or name. We define *rmNo* as a basic type, which represents the name or the number of a room in a hospital.

$$[\textit{rmNo}]$$

As described earlier, a hospital has a finite number of rooms, *MaxRooms.* All transactions on rooms should be less than or equal to this finite number, *MaxRooms.*

$$\begin{array}{|l}
\textit{MaxRooms} : \mathbb{N}
\end{array}$$

Rooms in hospitals are of various types. The following free type definition of *RoomType* explains the various kinds of rooms.

$$RoomType ::= Executive \mid Hospitality \mid Maternity$$

Every room has an associated cost when used (*cost*). This cost is dependent on the type of room. Records of rooms that have been booked for various dates should be available. The variable *book* is defined as a partial injective function mapping *rmNo* and the Cartesian product of *Patient* and *DATE*. The relation is modeled as a partial injective function to avoid duplication of the same room on a particular day, thus avoiding over booking. The number of rooms in a hospital may be added to or deleted from (in cases where a room is changed to service something else), I introduce *booking* as a subset of *rmNo*. Only those rooms, which can be scheduled, can be booked.

Rooms

$$\begin{array}{l}
RoomNo : rmNo \\
BasicFacilities : STRING \\
rmType : RoomType \\
cost : \mathbb{N}_1 \\
booking : \mathbb{P}\, rmNo \\
book : rmNo \rightarrowtail\!\!\!\!\!\mapsto Patient \times DATE \\
\hline
\operatorname{dom} book \subseteq booking
\end{array}$$

The schema *Rooms* describes the attributes necessary to book a room. This schema defines a variable *book* which denotes the current bookings in the hospital for rooms. Table 3.1 shows an instance of the partial injective function *book*.

Room Number	Patient	Date
1	Mathew	12/12/2001
2	Joe	12/12/2001
3	Randy	25/12/2001

Table 3.1: Room Booking

As shown above in Table 3.1, a list of Patients that have booked rooms on various/particular dates can be generated by the schema *Rooms*.

To make the schema definition of *Rooms* valid we must enforce a strict constraint that room numbers are unique. This universal constraint is specified as follows:

$$\forall\, r1, r2 : Rooms \bullet r1 \neq r2 \Rightarrow r1.RoomNo \neq r2.RoomNo$$

The above specification simply states that no two rooms are the same. The schema *RoomInit* describes the initialization of the room booking system in a hospital. When the system is first started, there are no bookings for rooms. The predicate *book'* = { } represents this initial state of the booking system.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{RoomInit}} \\
\Delta Rooms \\
\hline
book' = \{\}
\end{array}$$

Searching for the availability of a room requires one input (e.g., date in the considered case but could be by room type, etc) and booking a room requires three inputs, the date for which the room is requested, the patient's name and the room itself. The constraint here is that the type of room requested should be available. The operation of booking a room is captured in the following schema.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{BookRoom}} \\
\Delta Rooms \\
Date? : DATE \\
Pat? : Patient \\
roomno? : rmNo \\
Response! : Message \\
\hline
\#(\mathrm{dom}\ book) < MaxRooms \\
roomno? \in booking \setminus \mathrm{dom}\ book \\
book' = book \cup \{roomno? \mapsto (Pat?, Date?)\} \\
booking' = booking \\
Response! = Available
\end{array}$$

The cost of staying in a room depends on the actual cost of the room and the number of days the room is occupied by a patient. The following schema, *RoomCost*, captures the above requirement and computes the cost of staying in a given room.

$$
\begin{array}{|l}
\textit{RoomCost} \\
\hline
\Xi \textit{Rooms} \\
\textit{room}? : \textit{Rooms} \\
\textit{NumberofDays}? : \mathbb{N}_1 \\
\textit{RoomCost}! : \mathbb{N}_1 \\
\hline
\exists\, r1 : \textit{Rooms} \mid r1.\textit{RoomNo} \in \textit{booking} \bullet \\
\qquad r1.\textit{RoomNo} = \textit{room}?.\textit{RoomNo} \land \\
\qquad \textit{RoomCost}! = r1.\textit{cost} * \textit{NumberofDays}? \\
\hline
\end{array}
$$

The date a patient was admitted, the room number a patient stayed in and the cost a patient paid should be captured for future reference. Schema *PatientPays* captures this requirement.

$$
\begin{array}{|l}
\textit{PatientPays} \\
\hline
\textit{Pays} : \textit{Patient} \nrightarrow \textit{rmNo} \times \textit{DATE} \times \textit{DATE} \times \textit{RoomCost} \\
\hline
\end{array}
$$

Table 3.2 shows instances of patient stay dates in a hospital and the cost a patient paid, which could be used when developing a report of usage of various rooms.

Room Number	Patient	From	To	Room Cost
1	Robert Barry	12/12/2001	15/12/2001	$ 250
2	Roger Moore	13/01/2002	17/02/2002	$ 450
3	Shiela Murray	24/02/2002	29/02/2002	$ 675

Table 3.2: Patient's Stay and Cost

Quite possibly, when there is a scarcity of rooms, the system may output the fact that there are no rooms presently available. The schema *NoRoom* captures this requirement.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{NoRoom}} \\
\hline
Rooms \\
Date? : DATE \\
Response! : Message \\
\hline
\exists r : rmNo \mid r \in booking \bullet \neg\, (r \in \operatorname{dom} book) \\
Response! = NotAvailable \\
\hline
\end{array}$$

After booking a room a patient may decide to cancel the booking. In such scenarios, the room may be returned to the booking list of the hospital so that it is made available for another patient. The operation of canceling a room is described by the schema *CancelRoom.*

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{CancelRoom}} \\
\hline
\Delta Rooms \\
date? : DATE \\
rooms? : rmNo \\
Response! : Message \\
\hline
rooms? \in \operatorname{dom} book \\
book' = \{rooms?\} \mathbin{⩤} book \\
Response! = Cancelled \\
\hline
\end{array}$$

Notice that in the schema *CancelRoom*, a room can be cancelled if and only if it was already booked.

Personnel at the frontdesk of a hospital should be able to retrieve information about a patient's room number. Since a patient can not be associated with two different rooms on any given day, a function to retrieve this information is described as follows:

$$\begin{array}{|l}
PatientIn : Patient \rightarrowtail\!\!\!\!\rightarrow rmNo \\
\hline
\forall x, y : Patient \bullet x = y \Rightarrow x.pid = y.pid
\end{array}$$

The overall process in defining the system of booking and canceling a room is combined in *RoomBooking.*

$$RoomBooking \mathrel{\widehat{=}} BookRoom \lor NoRoom \lor CancelRoom$$

When a patient wishes to change rooms then the system should be flexible to accommodate the room change. Such operations involve first paying for the room used, booking a new room and then canceling the existing room. This is defined in the following specification, *ChangeRoom.*

$$ChangeRoom \mathrel{\widehat=} RoomCost \land BookRoom \land CancelRoom$$

Resources

Every room in a hospital has basic needs (like beds, pans, lights, etc.). For patient's convenience, additional resources (like television sets, books, etc.). can also be ordered. Such resources are defined by the schema *Resources.* Various departments provide certain resources for patients' service. To guarantee that a resource is identified uniformly anywhere in the hospital, we define a basic type *resNo* for this purpose.

$$[resNo]$$

Thus, associated with every resource is an unique resource identity number (*resNo*) and a cost for the use of the resource. We define *Order* as a function mapping resource to the cartesian product of the date and the room that requested the resource. Order is mapped as a partial function because it is very likely that there may be requests for two units of the same resource on a particular date by the same room number. The number of resources in a hospital may be added to or deleted from, hence we introduce *ResOrdering* as a subset of *resNo.*

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Resources}} \\
\hline
Name : STRING \\
ResourceNo : resNo \\
cost : \mathbb{N}_1 \\
ResOrdering : \mathbb{P}\, resNo \\
Order : resNo \nrightarrow DATE \times Rooms \\
\hline
\operatorname{dom} Order \subseteq ResOrdering \\
\hline
\end{array}$$

Table 3.3 shows instances of an order made on a resource to a particular room on a particular date based on the definition of *Resources.*
The operation of ordering a resource is captured in the following schema, *BookResource.*

Resource No	Date	Room Number
A67491	27/12/2001	14
E98432	08/07/1978	08
F56793	15/08/1994	18

Table 3.3: Resource Allocation

BookResource

$$\begin{array}{l}
\Delta Resources \\
Date? : DATE \\
room? : Rooms \\
resno? : resNo \\
Response! : Message \\
\hline
resno? \in ResOrdering \setminus \operatorname{dom} Order \\
Order' = Order \cup \{resno? \mapsto (Date?, room?)\} \\
ResOrdering' = ResOrdering \\
Response! = Available
\end{array}$$

Resources also carry a cost and depending on the duration of usage the cost is calculated. The following schema, *ResourceCost*, models the cost involved in using resources.

ResourceCost

$$\begin{array}{l}
\Xi Resources \\
ResourceType : \mathbb{P}\ Resources \\
Res? : Resources \\
NumberofDays? : \mathbb{N} \\
ResourceCost! : \mathbb{N} \\
\hline
Res? \in ResourceType \\
\exists\, res : Resources \mid res \in ResourceType \bullet \\
\qquad res.ResourceNo = Res?.ResourceNo \land \\
\qquad ResourceCost! = NumberofDays? * res.cost
\end{array}$$

As with rooms, *Resources* may also be unavailable, cancelled and/or the person searching for a resource could have supplied a incorrect resource number. The following schema, *NoResource*, captures the situation when a resource is unavailable.

$$\begin{array}{|l}
\hline
\textit{NoResource} \\
\Xi \mathit{Resources} \\
\mathit{date?} : \mathit{DATE} \\
\mathit{room?} : \mathit{Rooms} \\
\mathit{Response!} : \mathit{Message} \\
\hline
\neg\, (\exists\, \mathit{res} : \mathit{resNo} \bullet \mathit{res} \mapsto (\mathit{date?}, \mathit{room?}) \notin \mathit{Order}) \\
\mathit{Response!} = \mathit{NotAvailable} \\
\hline
\end{array}$$

Factors such as cost (e.g., lesser coverage of insurance) and unforseen contingencies could sometimes make a user cancel a previous request for a resource. The schema *CancelResource* captures the act of cancelling a resource.

$$\begin{array}{|l}
\hline
\textit{CancelResource} \\
\Delta \mathit{Resources} \\
\mathit{res?} : \mathit{resNo} \\
\mathit{room?} : \mathit{Rooms} \\
\mathit{Response!} : \mathit{Message} \\
\hline
\mathit{res?} \notin \operatorname{dom} \mathit{Order} \\
\mathit{Order}' = \{\mathit{res?}\} \lhd \mathit{Order} \\
\mathit{Response!} = \mathit{Cancelled} \\
\hline
\end{array}$$

Notice that in schema *CancelResource*, a resource can be cancelled if and only if it was already requested.

The overall process in defining the system of ordering and canceling a resources is combined in *ResourceOrdering*.

$$\mathit{ResourceOrdering} \mathrel{\widehat{=}} \mathit{BookResource} \lor \mathit{NoResource} \lor \mathit{ResourceCost} \lor \mathit{CancelResource}$$

The resource used by a particular patient is captured in the schema *ResourceFor-Patient*. In this schema we define a function mapping the *Patient* to the cartesian product of the resources the patient used and the cost for using the resource.

$$\begin{array}{|l}
\hline
\textit{ResourceForPatient} \\
\mathit{res} : \mathit{Patient} \nrightarrow \mathit{Resources} \times \mathit{ResourceCost} \\
\hline
\end{array}$$

Table 5.3 shows an instance of *res* values. I model the *TotalCost* that a patient has accumulated staying in a room and associated resources requests as a combination of *PatientPays* and *ResourceForPatient*.

$$TotalCost \mathrel{\widehat=} PatientPays \land ResourceForPatient$$

Patient	Resource	Resource Cost
Mathew	Extra Bed	\$14
Borris	Television	\$08
Stephen	Computer	\$18

Table 3.4: Resources used by a Patient

Frontdesk

Before a patient meets a doctor, the patient reports to the frontdesk personnel requesting an appointment. Here the patients have the freedom to express the way they feel. This is particularly important in deriving similarities in symptoms, for future observations and drawing association rules (e.g., data discovery from patient medical records when an intelligent system will be modelled). The patient information (patient's name and problem in the patient's words), the date and the opinion of the doctor after the encounter are contained in the schema, *Description*.

Description

$$
\begin{array}{|l}
Pat : Patient \\
Doc : Doctor \\
days : \mathbb{N}_1 \\
Date : DATE \\
PatWords, DocWords : \mathbb{P}\, STRING \\
\hline
PatWords \neq \emptyset
\end{array}
$$

With the first encounter of a patient with the frontdesk personnel, the medical records for that particular patient is created in that hospital (and, at times, the medical record itself becomes active in a networked hospital environment). Every appointment of a patient with a doctor is captured by the function *appointment* which is a partial injective function mapping the *Patient* to the cartesian of *DATE*, *TIME*(s) (for the start time and end time of the appointment), and *Doctor*. It is modeled in this way because on any day in a hospital at a particular time a number of doctors could be treating any number of patients. The partial injection ensures that a particular *Patient* is related only to one particular *Doctor* at a given date and time. The variable *duration* in the schema *DrAppointment* computes the time a doctor spends with a patient. Practically, the start times and the end times are defined (e.g.. the first appointment slot for the day could be between 10:00am (start time) and 10:20am (end time) and the next slot could start at 10:30)

$$
\begin{array}{|l}
\textit{DrAppointment} \\
\hline
\textit{appointment} : \textit{Patient} \rightarrowtail\!\!\!\!\rightarrow \textit{DATE} \times \textit{TIME} \times \textit{TIME} \times \textit{Doctor} \\
\textit{duration} : \mathbb{N} \\
\hline
\forall\, \textit{Date} : \textit{DATE};\ \textit{Pat1}, \textit{Pat2} : \textit{Patient};\ \textit{prevStrt}, \textit{strtTime}, \textit{endTime}, \\
\qquad \textit{prevEnd} : \textit{TIME};\ \textit{Dr} : \textit{Doctor} \mid \textit{Pat1} \neq \textit{Pat2} \bullet \\
\qquad (\textit{Pat1} \mapsto (\textit{Date}, \textit{strtTime}, \textit{endTime}, \textit{Dr})) \in \textit{appointment} \wedge \\
\qquad (\textit{Pat2} \mapsto (\textit{Date}, \textit{prevStrt}, \textit{prevEnd}, \textit{Dr})) \in \textit{appointment} \wedge \\
\qquad \textit{endTime} > \textit{strtTime} \wedge \\
\qquad \textit{duration} = \textit{endTime} - \textit{strtTime} \wedge \\
\qquad \textit{strtTime} = \textit{prevStrt} + \textit{duration} \wedge \\
\qquad \textit{prevEnd} = \textit{strtTime} \\
\hline
\end{array}
$$

Instances of the function *appointment* are shown in Table 3.5

Patient	Date	Start Time	End Time	Doctor
Roy Emmerson	11/12/2001	11:30	12:00	Dr. Harry
Stark Living	27/12/2001	12:15	12:45	Dr.Williams
Marc Robinson	08/07/2002	14:00	14:30	Dr. Mathew

Table 3.5: Appointment List

Similar to room booking, a doctor's appointment should also support functionality to book and cancel appointments. The schema *DrAppointmentInit* initializes a doctor's appointment book. Initially, there are no appointments.

$$
\begin{array}{|l}
\textit{DrAppointmentInit} \\
\hline
\Delta \textit{DrAppointment} \\
\hline
\textit{appointment}' = \{\} \\
\hline
\end{array}
$$

To book an appointment, the date and doctor information are the inputs. If any appointment time is available then the time is given as the output and the appointment information is inserted into the appointment book. The operation is formally specified by the schema *BookDrAppoint*, given below.

$$
\begin{array}{|l}
\textit{BookDrAppoint} \\
\hline
\Delta \textit{DrAppointment} \\
\textit{Date}? : \textit{DATE} \\
\textit{Dr}? : \textit{Doctor} \\
\textit{Pat}? : \textit{Patient} \\
\textit{Response}! : \textit{Message} \\
\hline
\exists\, \textit{strtTime}, \textit{endTime} : \textit{TIME} \bullet \\
\quad (\textit{Date}?, \textit{strtTime}, \textit{endTime}, \textit{Dr}?) \notin \operatorname{ran} \textit{appointment} \land \\
\quad \textit{appointment}' = \textit{appointment} \cup \{\textit{Pat}? \mapsto (\textit{Date}?, \textit{strtTime}, \textit{endTime}, \textit{Dr}?)\} \\
\textit{appointment}' = \textit{appointment} \\
\textit{Response}! = \textit{Available} \\
\hline
\end{array}
$$

Alternatively, if the hospital maintains a time interval to serve patients, then the free type definition of *Slots* (below) can be employed to represent the scheduling as a finite number of intervals/appointments a doctor/health care personnel can devote to a patient. As an example, *Slots* (below)shows a two interval appointment scheduling (e.g., slot 1 is scheduled for the AM session and slot 2 is scheduled for the PM session). They can be customized from hospital to hospital and according to the schedule interval (e.g., 10:00 - 10:15 could be a slot for a doctor-patient appointment and 10:00 - 14:00 could be a slot for booking a operating theater).

$$\textit{Slots} ::= \textit{AM} \mid \textit{PM}$$

$$
\begin{array}{|l}
\textit{slot}1, \textit{slot}2 : \textit{Slots} \\
\hline
\textit{slot}1 = \textit{AM} \\
\textit{slot}2 = \textit{PM}
\end{array}
$$

To allow individual organizations to choose their type of scheduling, I use the type defined for time for patient scheduling. However, if there are no appointments available with a particular doctor, the fact is specified in the *NoAppoint* schema.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{NoAppoint}} \\
\Xi DrAppointment \\
Date? : DATE \\
Dr? : Doctor \\
Response! : Message \\
\hline
\neg\,(\exists\, strtTime, endTime : TIME \bullet \\
\qquad\qquad (Date?, strtTime, endTime, Dr?) \notin \operatorname{ran} appointment) \\
Response! = NotAvailable \\
\hline
\end{array}
$$

When a patient cancels an appointment with a doctor, the appointment details like the patient and doctors' names and the date of the appointment are supplied as the input, the time of appointment is retrieved and is removed from the appointments book. The schema *CancelDrAppoint* captures this specification.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{CancelDrAppoint}} \\
\Delta DrAppointment \\
Dr? : Doctor \\
Date? : DATE \\
Pat? : Patient \\
Response! : Message \\
\hline
Pat? \in \operatorname{dom} appointment \\
\exists\, strtTime, endTime : TIME \bullet \\
\qquad\qquad (Date?, strtTime, endTime, Dr?) \in \operatorname{ran} appointment \wedge \\
\qquad\qquad Pat? \mapsto (Date?, strtTime, endTime, Dr?) \in appointment \wedge \\
\qquad\qquad appointment' = \{Pat?\} \lhd appointment \\
Response! = AppointmentCancelled \\
\hline
\end{array}
$$

At times a patient only knows that he/she has an appointment on a particular date (or just that the patient has an appointment). In this situation, the system should be able to retrieve the appointments of patients to cancel the appointment. In such scenarios, from the multiple responses that the system outputs, the patient chooses the appropriate appointment for cancellation. Then there arises a need for generating the list of appointmnets for the day (which will be discussed shortly).

CancelsDrAppoint

$$\begin{array}{l}
\Delta DrAppointment \\
Pat? : Patient \\
Response! : Message \\
\hline
Pat? \in \operatorname{dom} appointment \\
\exists\, strtTime, endTime : TIME;\ Dr : Doctor;\ date : DATE \bullet \\
\qquad (date, strtTime, endTime, Dr) \in \operatorname{ran} appointment \land \\
\qquad Pat? \mapsto (date, strtTime, endTime, Dr) \in appointment \land \\
\qquad appointment' = \{Pat?\} \lhd appointment \\
Response! = AppointmentCancelled
\end{array}$$

The scenario when an appointment is cancelled can then be specified as *CancelDrAppointment* using the logical compostion of the two schemas *CancelDrAppoint* and *CancelsDrAppoint.*

$$CancelDrAppointment \mathrel{\widehat{=}} CancelDrAppoint \lor CancelsDrAppoint$$

The overall process of scheduling an appointment with a doctor can then be modeled as a combination of the above schemas.

$$DoctorAppointment \mathrel{\widehat{=}} BookDrAppoint \lor NoAppoint \lor CancelDrAppointment$$

An important operation, in terms of retrieving appointments, is to generate a list of appointments on a particular day. The operation *TodaysAppointment* takes in one input, the date, and outputs the relevant attributes of the appointment.

TodaysAppointment

$$\begin{array}{l}
\Delta DrAppointment \\
Today? : DATE \\
\hline
\forall\, Pat : Patient \bullet \\
(\exists\, strtTime, endTime : TIME;\ Dr : Doctor \bullet \\
\qquad (Pat \mapsto (Today?, strtTime, endTime, Dr)) \in appointment \land \\
\qquad Pat \in \operatorname{dom} appointment)
\end{array}$$

Table 3.6 shows an example of a doctor's appointments with patients for the day. The slots for which a doctor is scheduled with a patient are populated with the patient's name and the slots without a name represent available appointment times that a patient can request an appointment with the corresponding doctor for that day.

Doctor	10:00 - 12:00	12:00 - 2:00	2:00 - 4:00
Dr. Roy Emmerson	Peter		Harry
Dr. Stark Living	Jacob	Prince	Michael
Dr. Marc Robinson	Williams		

Table 3.6: Todays Appointment

All hospitals and clinics have times which are allotted for emergencies. There is no advance booking required during this time and patients who arrive in emergency care are treated immediately or as soon as possible. *Status* is a basic type which corresponds to the level of emergency of a patient. It could be customized (e.g., different colors may represent various emergency levels) for a particular hospital.

$[Status]$

Also, a queue has to be formed with respect to the level of emergency. The variable *queue* is a partial function that binds a doctor to the cartesian product of *Patient* and *Status*. (queue is so defined because, at any time an available doctor can treat patients based on the status.

Emergency

$$queue : Doctor \rightarrowtail\!\!\!\!\!\!+\; Patient \times Status$$

The following schema, *EmergencyBooking*, defines the appointment booking process during an emergency. For an emergency, a patient is scheduled with any available doctor.

EmergencyBooking

$$\Delta Emergency$$
$$Pat? : Patient$$
$$status? : Status$$

$$\exists Dr : Doctor \bullet Dr \notin \operatorname{dom} queue \land$$
$$queue' = queue \cup \{Dr \mapsto (Pat?, status?)\}$$

After a patient is treated, the doctor is ready to treat the next patient in emergency. The fact that the doctor is free should be captured by removing the doctor from the list of doctors who are presently treating patients.

$$
\begin{array}{|l}
\hline
\textit{DrInQueue} \\
\Delta \textit{Emergency} \\
\textit{Dr}? : \textit{Doctor} \\
\hline
\textit{Dr}? \in \operatorname{dom} \textit{queue} \\
\textit{queue}' = \{\textit{Dr}?\} ⩤ \textit{queue} \\
\hline
\end{array}
$$

The various operations during an emergency are captured in *Emergencies.*

$$\textit{Emergencies} \mathrel{\widehat{=}} \textit{Emergency} \lor \textit{DrInQueue}$$

The following definition of *Frontdesk* combines the operations of scheduling patients (regular and emergency), patient search and adding patients to the health care organization.

$$\textit{Frontdesk} \mathrel{\widehat{=}} \textit{DoctorAppointment} \land \textit{Emergencies} \land \textit{PatientSearch} \lor \textit{TodaysAppointment}$$

Pharmacy

The following schema definitions model some of the transaction processing in the pharmacy department of a hospital. Inherently, a pharmacy database contains information about *Drug.* This information includes details like the name of the drug, the quantity in stock, the cost per unit of the drug, and the unique identification that identifies the drug, *DrugId.* I introduce *DrugId* as a basic type as follows.

$$[\textit{DrugId}]$$

Then I specify the attributes (structure) of a drug in the following schema *Drug.*

$$
\begin{array}{|l}
\hline
\textit{Drug} \\
\textit{DrugName}, \textit{Purpose}, \textit{Manufacturer} : \mathbb{P}\, \textit{STRING} \\
\textit{DrugCost}, \textit{Stock}, \textit{ReorderPoint} : \mathbb{N}_1 \\
\textit{DrugNo} : \mathbb{P}\, \textit{DrugId} \\
\hline
\textit{DrugNo} \neq \emptyset \\
\textit{DrugName} \neq \emptyset \\
\textit{Purpose} \neq \emptyset \\
\textit{Manufacturer} \neq \emptyset \\
\hline
\end{array}
$$

To make the above schema definition valid we must enforce a strict constraint that drug numbers are unique in any drug database, *DrugDB*. This is captured by the predicate part of the *DrugDB* schema.

$$\begin{array}{|l}
\hline \textit{DrugDB} \\
\textit{DrugList} : \mathbb{P}\, \textit{Drug} \\
\hline
\forall x, y : \textit{Drug} \bullet x \neq y \Rightarrow x.\textit{DrugNo} \neq y.\textit{DrugNo} \\
\hline
\end{array}$$

A drug can be added to and supplied if and only if that drug exists in the pharmacy. So, we should define the maximum and the minimum drug stock in the pharmacy. The stock of a particular drug cannot exceed *MaxDrug* units of the drug or reduce below *MinDrug* units of the drug. We define these variables as follows:

$$\begin{array}{|l}
\textit{MaxDrug} : \mathbb{N} \\
\textit{MinDrug} : \mathbb{N} \\
\hline
\textit{MinDrug} = 0 \\
\textit{MaxDrug} > 0
\end{array}$$

The operation *AddDrug* adds new drugs to the pharmacy database. To add drugs we need as inputs all those variables defined in the schema *Drug*. The new drug is added after verifying that the drug does not already exist in the pharmacy database.

$$\begin{array}{|l}
\hline \textit{AddDrug} \\
\Delta \textit{DrugDB} \\
\textit{DrugList}, \textit{DrugList}' : \mathbb{P}\, \textit{Drug} \\
\textit{drug}? : \textit{Drug} \\
\hline
\textit{drug}? \notin \textit{DrugList} \\
\textit{drug}?.\textit{Stock} > \textit{MinDrug} \\
\textit{drug}?.\textit{Stock} \leq \textit{MaxDrug} \\
(\forall d : \textit{DrugList} \bullet d.\textit{DrugNo} \neq \textit{drug}?.\textit{DrugNo}) \\
\textit{DrugList}' = \textit{DrugList} \cup \{\textit{drug}?\} \\
\hline
\end{array}$$

It should be possible to check if a particular drug is available in the pharmacy database. The schema, *DrugExist*, performs the checking for the existence of a drug in the pharmacy database.

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\; \mathit{DrugExist}\; \_\_\_\_\_\_\_\_\_\_} \\
\Xi \mathit{DrugDB} \\
\mathit{drug?} : \mathit{Drug} \\
\mathit{Response!} : \mathit{Message} \\
\hline
\mathit{drug?} \in \mathit{DrugList} \\
\mathit{Response!} = \mathit{DrugExists} \\
\hline
\end{array}$$

AddDrugs combines the process of successfully adding a new drug to the database or checking if the drug exists.

$$\mathit{AddDrugs} \mathrel{\widehat{=}} \mathit{AddDrug} \lor \mathit{DrugExist}$$

It is important that the pharmacy never runs out of stock. So we have to make sure that the system warns the user when a drug's stock is below a threshold (e.g., ≤ 50 units). The following schema, *DrugLessthan*, warns the user whenever the stock of a drug falls below the threshold value.

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\; \mathit{DrugLessthan}\; \_\_\_\_\_\_\_\_\_\_} \\
\Xi \mathit{DrugDB} \\
\mathit{DrugList} : \mathbb{P}\, \mathit{Drug} \\
\mathit{OnStock!} : \mathbb{N} \\
\mathit{Response!} : \mathit{Message} \\
\hline
\forall\, y : \mathit{Drug} \mid y \in \mathit{DrugList} \bullet \\
\qquad\qquad y.\mathit{Stock} \leq y.\mathit{ReorderPoint} \land \mathit{OnStock!} = y.\mathit{Stock} \\
\mathit{Response!} = \mathit{RunningOutofStock} \\
\hline
\end{array}$$

On any given day a particular drug's stock could be checked. So, the system should be able to retrieve the stock of a particular drug. The schema *DrugStocks* models the retrieval of the quantity of a drug on hand .

$$\begin{array}{|l}
\multicolumn{1}{l}{\_\; \mathit{DrugStocks}\; \_\_\_\_\_\_\_\_\_\_} \\
\Xi \mathit{DrugDB} \\
\mathit{drug?} : \mathit{Drug} \\
\mathit{Stock!} : \mathbb{N} \\
\hline
(\exists_1\, x : \mathit{Drug} \mid x \in \mathit{DrugList} \bullet \\
\qquad\qquad x.\mathit{DrugNo} = \mathit{drug?}.\mathit{DrugNo} \land \mathit{Stock!} = x.\mathit{Stock}) \\
\hline
\end{array}$$

Continuing research in medicine could discover potential threats in consuming certain drugs or improved drugs replacing old ones, hence the system should give the appropriate users (e.g., system administrators) the liberty to delete such drugs from the hospital database. The schema *DeleteDrug* captures the operation of deleting a drug from the database.

$$\begin{array}{|l}
\textit{DeleteDrug} \\
\hline
\Delta \textit{DrugDB} \\
\textit{DrugList}, \textit{DrugList}' : \mathbb{P}\, \textit{Drug} \\
\textit{drug}? : \textit{Drug} \\
\textit{Response}! : \textit{Message} \\
\hline
\textit{drug}? \in \textit{DrugList} \\
\textit{DrugList}' = \textit{DrugList} \setminus \{\textit{drug}?\} \\
\textit{drug}? \notin \textit{DrugList}' \\
\hline
\end{array}$$

One of the periodic updates in the pharmacy database, is the refilling of certain drug stocks. The schema *AddStock* captures the scenario where a newly added quantity is supplied as an input is added to the existing stock of the drugs.

$$\begin{array}{|l}
\textit{AddStock} \\
\hline
\Delta \textit{DrugDB} \\
\textit{DrugList}', \textit{DrugList} : \mathbb{P}\, \textit{Drug} \\
\textit{drug}? : \textit{Drug} \\
\textit{add}? : \mathbb{N}_1 \\
\textit{Response}! : \textit{Message} \\
\hline
\textit{drug}? \in \textit{DrugList} \\
(\exists_1\, x : \textit{Drug} \mid x \in \textit{DrugList} \bullet \\
\qquad x.\textit{DrugNo} = \textit{drug}?.\textit{DrugNo} \land \\
\qquad \textit{drug}?.\textit{Stock} = x.\textit{Stock} + \textit{add}? \land \\
\qquad \textit{drug}?.\textit{Stock} \leq \textit{MaxDrug} \land \\
\qquad \textit{DrugList}' = \textit{DrugList} \setminus \{x\} \cup \{\textit{drug}?\}) \\
\textit{Response}! = \textit{Updated} \\
\hline
\end{array}$$

Every time a drug is sold, it reduces the quantity in the pharmacy database by the same amount as the number of drugs dispensed. *ReduceStock* captures this fact. Since the transactions of adding or reducing stock effects the state of a particular drug, we must define that there exists a particular drug in the database for which a transaction might be done. Since, there exists number of drugs in the pharmacy database, we must specify that the transaction to be done is indeed performed on that particular drug. This is captured in the precondition of the schema *ReduceStock*

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ReduceStock}} \\
\hline
\Delta DrugDB \\
DrugList', DrugList : \mathbb{P}\, Drug \\
drug? : Drug \\
sub? : \mathbb{N}_1 \\
Response! : Message \\
\hline
drug? \in DrugList \\
(\exists_1\, x : Drug \mid x \in DrugList \bullet \\
\qquad x.DrugNo = drug?.DrugNo \wedge \\
\qquad drug?.Stock = x.Stock - sub? \wedge \\
\qquad drug?.Stock \geq MinDrug \wedge \\
\qquad DrugList' = DrugList \setminus \{x\} \cup \{drug?\}) \\
Response! = Updated \\
\hline
\end{array}$$

Finally, the cost of a drug dispensed is modeled by the schema *DrugsCost*.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{DrugsCost}} \\
\hline
\Xi DrugDB \\
drug? : Drug \\
qty?, cost! : \mathbb{N}_1 \\
\hline
(\exists_1\, d : Drug \mid d \in DrugList \bullet \\
\qquad d.DrugNo = drug?.DrugNo \wedge \\
\qquad cost! = qty? * drug?.DrugCost) \\
\hline
\end{array}$$

The transactions in *Pharmacy* are then modelled as:

$$\begin{aligned}
Pharmacy \mathrel{\widehat{=}} {} & AddDrugs \vee DrugLessthan \vee DeleteDrug \vee AddStock \vee \\
& ReduceStock \vee DrugsCost \vee DrugStocks
\end{aligned}$$

Encounter

Every encounter between a patient and any health care personnel is recorded. During an encounter, the date and time of the interaction is defined along with other details, (e.g., the identity number and name of the actors of the interaction as defined in

Description). An *EncounterRecord* is a sequence of *Descriptions*. A patient can have several encounters with one or more doctors. In such records, if the patient identity number repeats then it means that we are referring to the same person since patient identities are unique.

$$\begin{array}{|l}\hline \textit{EncounterRecord} \\ rec : \mathrm{seq}\, Description \\ \hline \end{array}$$

Doctors may write prescriptions after interacting with a patient. Such prescriptions consists of drug name(s), quantities, instruction, patients name and doctor name. A patient could be allergic to one or more drugs. The function *allergy* relates *Patient* to *Drug*.

$$\begin{array}{|l}\hline \textit{DefineAllergy} \\ allergy : Patient \rightarrow Drug \\ \hline \end{array}$$

Initially, the allergy list is empty. This fact is captured in the schema *AllergyInit*.

$$\begin{array}{|l}\hline \textit{AllergyInit} \\ \Delta DefineAllergy \\ \hline allergy' = \{\} \\ \hline \end{array}$$

When a doctor diagnoses a patient to be allergic to a drug the doctor adds this fact to the allergy list using the *AddAllergy* schema.

$$\begin{array}{|l}\hline \textit{AddAllergy} \\ DefineAllergy \\ drug? : Drug \\ patient? : Patient \\ \hline (patient?, drug?) \notin allergy \\ allergy = allergy \cup \{(patient? \mapsto drug?)\} \\ \hline \end{array}$$

Now, the system should not allow a doctor to prescribe drugs to which the patient is allergic. The schema *Prescription* captures this requirement and the process of a doctor prescribing medicine.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Prescription}} \\
\hline
\Xi \mathit{DefineAllergy} \\
\mathit{Doc}? : \mathit{Doctor} \\
\mathit{Pat}? : \mathit{Patient} \\
\mathit{drug}? : \mathit{Drug} \\
\mathit{PrescriptionList}, \mathit{PrescriptionList}' : \mathbb{P}\, \mathit{Drug} \\
\hline
(\mathit{Pat}? \mapsto \mathit{drug}?) \notin \mathit{allergy} \\
\mathit{PrescriptionList}' = \mathit{PrescriptionList} \cup \{\mathit{drug}?\} \\
\hline
\end{array}$$

During the course of treatment, doctors may refer a patient to another doctor (e.g, a specialist) or to a laboratory technician. Hence, the system should capture various personnel involved during the referral process and the reason for the referral. The schema *Refer* captures this specification.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Refer}} \\
\hline
\Xi \mathit{PatientDB} \\
\mathit{ReferringDrname}?, \mathit{ReferedDrname}? : \mathit{Doctor} \\
\mathit{pat}? : \mathit{Patient} \\
\mathit{reason}? : \mathbb{P}\, \mathit{STRING} \\
\hline
\mathit{reason}? \neq \emptyset \\
\mathit{ReferringDrname}? \neq \mathit{ReferedDrname}? \\
\mathit{pat}? \in \mathit{PatientList} \\
\hline
\end{array}$$

The overall process of interaction of a patient with a doctor is combined in *Encounter*

$$\begin{aligned}
\mathit{Encounter} \mathrel{\widehat{=}} {} & \mathit{EncounterRecord} \mathbin{⨟} ((((\mathit{Prescription} \land \mathit{AddAllergy}) \lor \mathit{Prescription}) \\
& \lor \mathit{Refer}) \\
& \lor \qquad ((\mathit{Prescription} \lor (\mathit{Prescription} \land \mathit{AddAllergy})) \land \mathit{Refer}))
\end{aligned}$$

Reports

In a health care system various personnel offer care to patients. These different care providers are captured as *Personnel*

$$Personnel ::= Doctors \mid Technicians \mid Nurse$$

A report of an interaction between a patient and a care provider may need to be generated. The schema *Report* contains the basic structure of a report. A report contains the report headers, body, date and time and the person who generates the report. A report may also contain images. This basic structure is extensible in order to create specific reports for certain special operations. We define *images* as a basic type, as follows:

$$[images]$$

Thus, a generic report is specified as:

$$\begin{array}{|l}
\textit{Report} \\
\hline
reportId, dept, Banner : STRING \\
body : \mathbb{P}\, STRING \\
date : DATE \\
time : TIME \\
Image : \mathbb{P}\, images \\
reportby : Personnel \\
\hline
body \neq \{\} \\
\hline
\end{array}$$

Image in the schema *Report* is a power set of the basic type *images* since some reports (e.g., Labarotory report), may have one or more images while other reports (e.g., a doctor's encounter with a patient), may not have any images at all. If the departments in the hospital were to generate a report, then the reports can be customized. For instance, a report from a Pharmacy can be generated with drug names, date, time, cost, department, and the banner of the hospital, and similarly laboratories can customize their reports with the above parameters (as in *Report*).

```
PharmacyReport
```
$Report$
$date? : DATE$
$time? : TIME$
$PatientDetails?, DrugDetails? : \mathbb{P}\,STRING$
$generatedby? : Personnel$
$Qty? : \mathbb{N}_1$
$report! : Report$

$Qty? \geq 1$
$report!.body = (DrugDetails? \cup PatientDetails?)$
$report!.date = date?$
$report!.time = time?$
$report!.reportby = generatedby?$

Network

Health care facility in the present model are networked. Hence, I specify the network with which the various sub-systems in the health care application communicate. The following definition of network is adopted from [In01].
Fundamentally, a network consist of nodes (or computers) with links between them. At this level of abstraction, we are more concerned about the relevant information (e.g, appointment and resource availabilities, etc.)from a node rather than the information itself, hence a unique address that maps a particular computer to a network node is defined as a basic type, *Ipaddress*.

$[Ipaddress]$

The schema *Node* contains the *Ipaddress* of the system.

```
Node
```
$ip : Ipaddress$

A *Network* is formed when a number of such nodes are connected. Because, a medical record is a confidential document, easy access to every one could jeopardize the privacy of medical records, so secured access to these records is a must. By secured access, I mean that only those systems in the network can communicate between other systems in the domain of participating health care organizations (which is a finite set).

$$\begin{array}{|l}
\textit{Network} \\
\hline
\textit{NetworkName} : \textit{STRING} \\
\textit{nodes} : \mathbb{F}\, \textit{Node} \\
\textit{network} : \textit{Node} \leftrightarrow \textit{Node} \\
\hline
\forall\, x, y : \textit{Node} \bullet (x.ip \neq y.ip) \land ((x, y) \in \textit{network} \Rightarrow (y, x) \in \textit{network}) \\
\hline
\end{array}$$

An information system may grow, which could result in many networks integrating to form a single network. This operation is captured as a new node being added to the existing system and to the existing network the added node becomes an integral part. The schema *AddToNetwork* captures the fact.

$$\begin{array}{|l}
\textit{AddToNetwork} \\
\hline
\Delta \textit{Network} \\
\textit{node}?, \textit{network}? : \textit{Node} \\
\hline
\textit{node}? \notin \textit{nodes} \\
\textit{nodes}' = \textit{nodes} \cup \{\textit{node}?\} \\
\textit{network}' = \textit{network} \cup \{(\textit{node}?, \textit{network}?)\} \\
\hline
\end{array}$$

The process of adding new computer systems to the network can be combined in *Distribution.*

$$\textit{Distribution} \mathrel{\widehat=} \textit{Network} \land \textit{AddToNetwork}$$

Now that computers can communicate, information between them can be transferred. But again access to shared files is given only to certain user groups in the network. Access rights are determined when users login to any health care application. When users login, irrespective of their access rights the application opens a connection to the database. We represent the access types that are available in the system as defined in [In01]:

$$\textit{Accesstype} ::= \textit{Select} \mid \textit{Insert} \mid \textit{Delete} \mid \textit{Update}$$

Once contacts is established with a remote database, the operation flow is similar as in local database. The electronic medical record from the remote database is modeled similarly as in the case of the local database.

Laboratory

Patients require the services of the laboratory personnel too. Hence, transactions in a laboratory should also be captured. In a laboratory, different kinds of tests may be performed. For the sake of uniformity, I model all the different tests under a single umbrella of tests. I define the types of test available in the system as follows:

$$\mathit{Test} ::= \mathit{Blood} \mid \mathit{XRay} \mid \mathit{Scan} \mid \mathit{Others}$$

A test that has to be performed requires a resource. Although a resource may come from another department and may be of different resource types, basically all resources share common features, so they can be placed under the already defined schema for resources, *Resources.*

The record of a laboratory test is captured in the schema *Laboratory.*

$$\begin{array}{|l}
\hline
\mathit{Laboratory} \\
\mathit{ReferedDrname} : \mathit{Doctor} \\
\mathit{pat} : \mathit{Patient} \\
\mathit{test} : \mathit{Test} \\
\mathit{Result} : \mathit{Report} \\
\hline
\end{array}$$

An electronic medical record is modeled as a combination of the patient's description of the problem, the patient's encounter with doctors, the doctor's description of the problem, the prescription for an ailment, the allergy list of the patient and doctor's referral to other offices (such as specialist doctor, laboratory, etc.). The following schema *LocalMedicalRecord* (since the medical record is specific to the interaction a patient encounters in one hospital, I use the term Local medical record) is defined as follows:

$$\begin{aligned}\mathit{LocalMedicalRecord} \mathrel{\widehat=} {} & \mathit{DoctorAppointment} \wedge \mathit{Encounter} \\ & \wedge \mathit{TotalCost} \wedge \mathit{Laboratory}\end{aligned}$$

Agents

The specification so far assumes that although distributed, the heterogeneous databases are part of the participating hospital's database. This is of prime importance because, as explained earlier, secured access of medical records is one of the primary constraints in designing such a system. In future, if all health care delivery institutions start to communicate in secured environments (which would be ideal for medical records) then we could use the assistance of agents in retrieving medical records of patients. Although the formal specification of agents in health care is not a main objective of this research, the use of agents in developing medical records (e.g., integrating patient information from various databases) is abundant in the literature [Mi97, Ha99, De92, De98]. This has motivated me to give the initial specifications below (as specifying agents for this domain is, by itself, I believe, a research topic for further investigation). At this abstract level, I designed *Agent* to possess various attributes like a reference identity number, type of agent, its goal and status of initializing the agent. I represent the computations involved as a basic type, *Compute*, and an agent's goal as the basic type *Aim*.

$$[\mathit{Aim}, \mathit{Compute}]$$

Agents are often generated for a specific role (e.g., an agent can assume the role of an information supplier or an agent requesting information).

$$\mathit{Role} ::= \mathit{InformationSupplier} \mid \mathit{InformationRequest}$$

An agent could detect valuable information or could not. Hence the result of an agent computation can be represented as a boolean.

$$\mathit{Bool} ::= \mathit{True} \mid \mathit{False}$$

Ehikioya's model [Eh99a] of an agent is defined as follows.

$$\begin{array}{|l}
\hline
\mathit{Agent} \\
\mathit{agentno} : \mathbb{N} \\
\mathit{role} : \mathit{Role} \\
\mathit{status} : \mathit{Bool} \\
\mathit{aim} : \mathbb{F}_1\, \mathit{Aim} \\
\hline
\end{array}$$

Agents in Ehikioya's model terminates after performing a certain number of computations. Although the time required for performing a computation is a major attribute, time-based agent systems are out of the scope of the work presented in this thesis.

Agentprocess

$$\begin{array}{|l}
compute : \mathbb{F}\; Compute \\
Response! : Message \\
\hline
Response! = Success \lor Response! = Failure
\end{array}$$

The agent approach used by [IE01] in the domain for e-commerce transactions can be suitably adapted to the medical domain as well. The execution of an agent's task is defined as the cartesian product of the agent and its aim. Once an agent performs its computation it generates a report.

$$\mid\; action : Agent \times Aim \rightarrow Report$$

The definition of *Agent* and the process agents are commited to are combined in *Agentbased* as follows,

$$Agentbased \mathrel{\widehat{=}} Agent \land Agentprocess$$

Users

Finally, the various users (such as Doctors, Nurses, Labarotory technicians, Pharmacists, etc) can be modeled as users of the integrated application. So they can be defined using a single schema *Users* (*Users* could be a subset of *Persons*). All users of the application have a login account which lets them access the records to which they have permission. The login name, composed of a *username* and a *password*, authenticates users.

$$username == STRING$$

$$password == STRING$$

$$group ::= Dr \mid LabPersonnel \mid Pharmacists \mid Nurses \mid Administrators$$

Database

The following specifications for designing a relational schema are based on the definition of a relational schema by Elmasri and Navathe [EN94]. In describing a relational schema we first define *DataType* and *name* as basic types.

$[DataType, name]$

I then define *Attribute* as a function mapping a name to a datatype. This corresponds to a column in a table.

$$Attribute == name \pfun DataType$$

Every defined attribute has a associated value. So, in this specification we assume *Value* as a basic type.

$[Value]$

The value for every attribute (or one of the several attributes) forms a *Tuple* (corresponding to rows in a table).

$$Tuple == Attribute \pfun Value$$

Finally, I define a table (or *Relation*) itself.

$$\begin{array}{|l}
\textit{Relation} \\
\hline
Columns : \mathrm{seq}_1\ Attribute \\
Rows : \mathbb{F}\ Tuple \\
\hline
\#Columns \geq 1 \\
\hline
\end{array}$$

An example of a two column table is shown in Table 3.7where (Roy Emmerson, 22) is an example of one of the rows and "Name" and "Age" correspond to *Attributes* with STRING and real as their data types, respectively.

Name	Age
Roy Emmerson	22
Stark Living	27
Marc Robinson	20

Table 3.7: Two Column Table

Support for maintaining backups for the patient information is an important operation, so the schema *Backup* is specified to fulfill this functionality.

$$\begin{array}{|l}
\textit{Backup} \\
\hline
\textit{present} : \textit{Relation} \\
\textit{backup} : \textit{Relation} \\
\hline
\end{array}$$

The operation of doing a backup can be performed as described in the schema *Copy*

$$\begin{array}{|l}
\textit{Copy} \\
\hline
\Delta \textit{Backup} \\
\textit{present}?, \textit{present}! : \textit{Relation} \\
\textit{backup}! : \textit{Relation} \\
\hline
\textit{present}! = \textit{present}? \\
\textit{backup}! = \textit{present}? \\
\hline
\end{array}$$

Additionally, when an erroneous transaction is committed, there is a need to restore the table to its earlier state. This operation is contained in the schema *Rollback*.

$$\begin{array}{|l}
\textit{Rollback} \\
\hline
\Delta \textit{Backup} \\
\textit{present}! : \textit{Relation} \\
\textit{backup}?, \textit{backup}! : \textit{Relation} \\
\hline
\textit{present}! = \textit{backup}? \\
\textit{backup}! = \textit{backup}? \\
\hline
\end{array}$$

A database is a collection of related information [EN94]. This related infprmation is s tored in tables. Therefore, I define *Database* as a collection of tables (*Relation*).

$$\textit{Database} == \mathbb{P}\, \textit{Relation}$$

Information from a database either can or can not be obtained based on the mode of the database (a database can be open or closed). The variable *connected* is a boolean representing the open (*dbOpen*) or the closed mode (*dbClose*) of the database

$$\textit{connected} ::= \textit{dbOpen} \mid \textit{dbClose}$$

The following schema, *Db2* (I chose the name *DB2* arbitarily), initializes the database to be in the closed mode.

$$
\begin{array}{|l}
\hline
\textit{Db2} \\
db : \textit{Database} \\
open : \textit{connected} \\
\hline
open = \textit{dbClose} \\
\hline
\end{array}
$$

The schema *OpenDatabase* establishes connection with a database.

$$
\begin{array}{|l}
\hline
\textit{OpenDatabase} \\
\textit{Databases} : \mathbb{P}\,\textit{Db2} \\
\textit{database}', \textit{database}? : \textit{Db2} \\
\hline
\exists_1\, x : \textit{Db2} \mid x \in \textit{Databases} \bullet x.open = \textit{dbClose} \\
\textit{database}' = \textit{database}? \\
\textit{database}'.open = \textit{dbOpen} \\
\hline
\end{array}
$$

After a transaction is committed, the database should be closed. This is specified in the schema *CloseDatabase*.

$$
\begin{array}{|l}
\hline
\textit{CloseDatabase} \\
\textit{Databases} : \mathbb{P}\,\textit{Db2} \\
\textit{database}', \textit{database}? : \textit{Db2} \\
\textit{Response}! : \textit{Message} \\
\hline
\textbf{if } \textit{database}?.open = \textit{dbClose} \\
\textbf{then } \textit{Response}! = \textit{DatabaseNotOpen} \\
\textbf{else } \textit{Response}! = \textit{DatabaseWillbeClosed} \land \\
\qquad\qquad (\exists_1\, x : \textit{Db2} \mid x \in \textit{Databases} \bullet x.open = \textit{dbOpen} \land \\
\qquad\qquad \textit{database}' = \textit{database}? \land \\
\qquad\qquad \textit{database}'.open = \textit{dbClose}) \\
\hline
\end{array}
$$

A hospital information system is a safety critical system, so all users must have an account in the hospital for authentication. Typically, a username and a password is required for basic authentication. Also, different users exist in the health care domain (e.g., doctors, nurses, pharmacists, laboratory technicians, receptionists, and system administrators) and each of the above groups have different access rights in the hospital. For example, doctors have the right to diagnose and prescribe drugs while other user groups cannot. These access rights determine if a user that wishes to perform a certain operation is authorized to do so. The information about a user's account and access rights is stored in *User*, defined formally as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{User}} \\
\hline
user : username \rightarrowtail\!\!\!\!\!\!+\;\; password \\
usergroup : username \nrightarrow group \\
access : username \leftrightarrow (Relation \times Accesstype) \\
\hline
\mathrm{dom}\, access \subseteq \mathrm{dom}\, user \\
\hline
\end{array}$$

Every user who uses the information system software has to supply his/her username and password. If the username and password matches the stored values for these variables/attributes then the user gains entry and a suitable message is displayed to the user. This use case is captured by the operation *Login*, defined as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Login}} \\
\hline
\Xi User \\
uname? : username \\
password? : password \\
message! : Message \\
\hline
(uname?, password?) \in user \\
message! = Success \\
\hline
\end{array}$$

For various reasons users in the health care domain may want to change their password periodically. To change a password, the user must first login to the system. The user then submits his/her old password, new password and a confirmation of the new password.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{ChangePassword}} \\
\hline
\Delta User \\
uname? : username \\
oldpassword?, newpassword?, confirmpassword? : password \\
\hline
(uname?, oldpassword?) \in user \\
newpassword? = confirmpassword? \\
user' = user \oplus \{uname? \mapsto newpassword?\} \\
\hline
\end{array}$$

New users to the system have to be added and then their access levels set. This is one of the jobs of the system administrator. To add a new user, the system administrator must first login and then supply the new user's username and password. The system then checks the username for uniqueness and the administrator assigns the user to a usergroup which the new user belongs to so that the user inherits the usergroup access levels.

$$
\begin{array}{|l}
\textit{AddUser} \\
\hline
\Delta \textit{User} \\
\textit{admin?}, \textit{newuser?} : \textit{username} \\
\textit{adminpassword?}, \textit{newuserpassword?} : \textit{password} \\
\textit{ugroup?} : \textit{group} \\
\hline
\textit{newuser?} \notin \operatorname{dom} \textit{user} \\
(\textit{admin?}, \textit{Administrators}) \in \textit{usergroup} \\
\textit{user}' = \textit{user} \oplus \{\textit{newuser?} \mapsto \textit{newuserpassword?}\} \\
\textit{usergroup}' = \textit{usergroup} \oplus \{\textit{newuser?} \mapsto \textit{ugroup?}\} \\
\hline
\end{array}
$$

Though users privelages are not likely to change, system administrators should have the privilege to grant new access levels to existing users. To add privileges, the administrator first checks for the username and then specifies the table(s) which the new privelage is/are applicable.

$$
\begin{array}{|l}
\textit{Grant} \\
\hline
\Delta \textit{User} \\
\textit{admin?}, \textit{uname?} : \textit{username} \\
\textit{table?} : \textit{Relation} \\
\textit{grant?} : \textit{Accesstype} \\
\hline
\textit{uname?} \in \operatorname{dom} \textit{user} \\
(\textit{admin?} \mapsto \textit{Administrators}) \in \textit{usergroup} \\
\textit{access}' = \textit{access} \cup \{\textit{uname?} \mapsto (\textit{table?}, \textit{grant?})\} \\
\hline
\end{array}
$$

Administrators should also be able to delete users from the system.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Remove}} \\
\hline
\Delta \textit{User} \\
\textit{admin?}, \textit{DeleteUser?} : \textit{username} \\
\hline
\textit{DeleteUser?} \in \operatorname{dom} \textit{user} \\
(\textit{admin?}, \textit{Administrators}) \in \textit{usergroup} \\
\textit{user}' = \{\textit{DeleteUser?}\} \lhd \textit{user} \\
\textit{usergroup} = \{\textit{DeleteUser?}\} \lhd \textit{usergroup} \\
\textit{access}' = \{\textit{DeleteUser?}\} \lhd \textit{access} \\
\hline
\end{array}
$$

Query Modeling

I now specify the data manipulation commands (DML) that users use in this domain. These commands (queries) consist of type of access, the table on which the access is requested, and possibly one or more predicates and constraints. A simple command for retrieving a doctor's appointment for a particular day can be modeled as:

Select Drname
From Doctor
Where date = '12/01/2002'
Order by Drname;

In the above SQL statement, Select determines the kind of access the query performs on the table Doctor based on the condition (predicate, that its on the 12th day of January 2002) and the results are sorted based on the doctor's name. So, to model a DML query we need to specify predicates too. First, we define the following basic types:

$$[\textit{COMPARATOR}, \textit{NUMBERS}, \textit{EXPRESSIONS}, \textit{Column}]$$

The COMPARATOR basic type can assume one of the following operators ($=, \neq, <, >, \geq, \leq$) to compare two attributes. The basic type NUMBERS includes the natural numbers (only natural numbers because of the restriction posed by the Z specification language). Although we consider only natural numbers, practically NUMBER is universal and can assume any data type (integer, real, etc) when applicable.
For example, in the SQL statement,

Select Drname **From** Doctor **Where** DrID = 12

the field DrID is a NUMBER
Compare is a basic type which embeds constants and attribute

$$Compare ::= col\langle\langle Column\rangle\rangle \mid num\langle\langle NUMBERS\rangle\rangle \mid exp\langle\langle EXPRESSIONS\rangle\rangle$$

In predicates we, therefore, compare two operands (operand1 and operand2) of which operand1 is always an attribute to be used for comparison in the table and the other could be a constant. We represent *Predicate* based on [EN94] and as defined in [In01].

Predicate

$$\begin{array}{l} operand1 : Column \\ operator : COMPARATOR \\ operand2 : Compare \end{array}$$

So a query can be formally defined as

Query

$$\begin{array}{l} \Xi User \\ rel : \mathbb{F}\, Relation \\ records : \mathbb{P}\, Tuple \\ groupby : \mathrm{seq}\, Attribute \\ predicates : \mathbb{F}\, Predicate \\ \hline (\exists\, r : Relation \bullet r.Rows = records \land r.Columns = groupby) \end{array}$$

In order to generate a report, the application must first accept a query to process the required report. The schema, *AskQuery* accepts a user's query.

AskQuery

$$\begin{array}{l} \Xi User \\ uname? : username \\ query? : Query \\ \hline uname? \in \mathrm{dom}\, access \end{array}$$

The process of executing a query based on the user rights has been demonstrated in [In01] and will not be repeated here.

The individually specified transactions and departments constitute the distributed and integrated health care system. Thus,

$$\begin{aligned} \mathit{DIHA} \mathrel{\widehat{=}} {} & \mathit{RoomBooking} \land \mathit{ResourceOrdering} \land \mathit{Frontdesk} \land \\ & \mathit{Distribution} \land \mathit{LocalMedicalRecord} \land \\ & \mathit{Pharmacy} \land \mathit{Report} \end{aligned}$$

Chapter 4

System Design

"You know you have achieved perfection in design, not when you have nothing more to add, but when you have nothing more to take away." - Antoine de Saint Exupery

This chapter discusses the system architecture of a health care information system that interfaces with other participating health care organizations. Section 4.1 gives a description of the architecture and describes the different components of the architecture. Section 4.2 briefs the reader about query processing in the assumed network environment. Section 4.3 explains the flow of a patient record in the health care system, Section 4.4 presents the UML diagrams and some of the relevant pseudocodes for the transactions implemented. Finally Section 4.5 provides a summary of the chapter.

4.1 Architecture Design

Any proposed architecture for an integrated distributed health care system must capture, store, search for, retrieve, update and make available information contained in a patient's medical record. This information can, however, be created and maintained in geographically distributed locations.

Figure 4.1 gives an overall, logical view of the architecture of the distributed and integrated health care system presented in this thesis. Figure 4.2 shows the system's 3-tier physical structure which consists of a Database Repository, Middleware layer, and the Client's Interface.

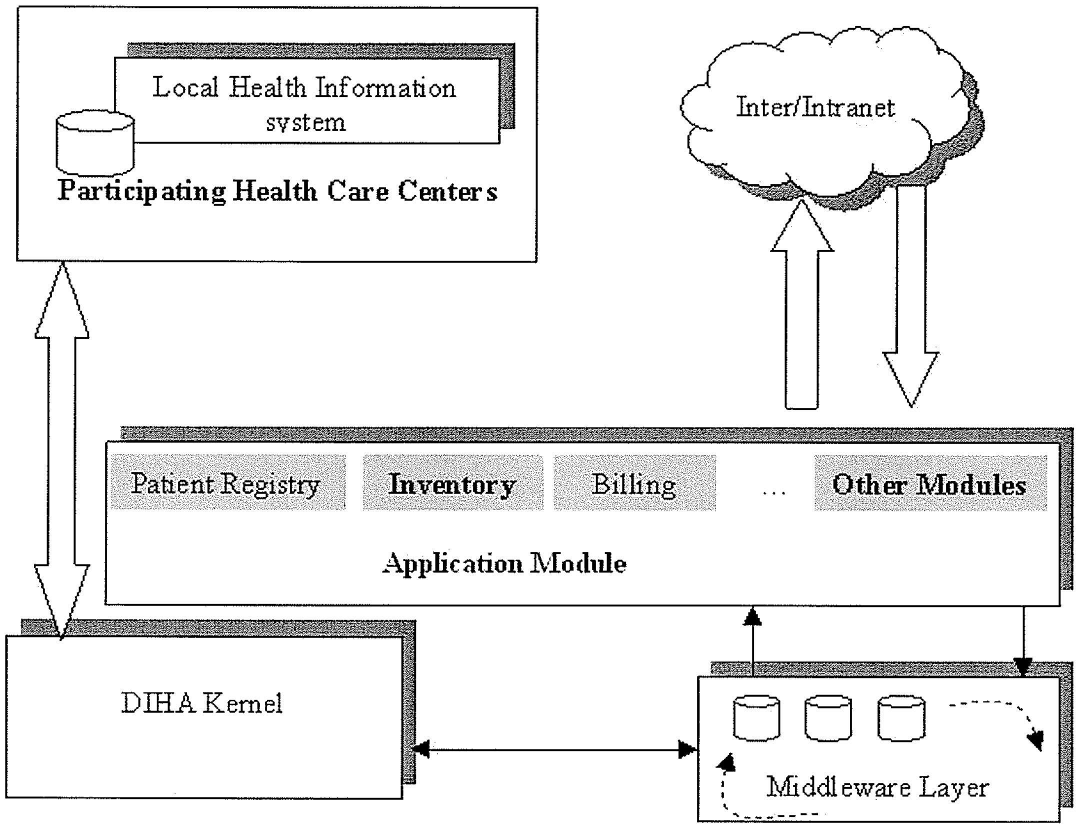


Figure 4.1: Architecture of a HIS

The Application Module (Figure 4.1) is the client-level interface to the application. This interface lacks intelligence as is appropriate for a web-based design (in this thesis a web-based design is adopted only to reflect that health care applications can be web-based) and only passes a user's query to the next tier. Components in the second tier include middleware and the Distributed and Integrated Health care Application (DIHA) kernel. The middleware layer receives a user request (i.e., queries) from the interface, and makes decisions on issues such as authentication, transaction processing, QoS, load balancing, etc. and then passes the actual query to the kernel where it is executed. The execution of a query involves contacting the database and performing the required operations (based on the user request). The kernel receives the result of query execution from the database, which is then displayed to the user through the middleware layer. For remote applications that need to access patient record information stored, for example, in a remote hospital's database, the client interface of the remote application will contact the middleware layer and then the processing will continue as before in the local system. The separation of the database from the client through the middleware layer helps to address security issues in remote queries and in the local domain. While some of the characteristics of this architecture impact the formal specifications I have done and presented , other details, for example in the middleware layer (i.e., QoS, load balancing, etc.), are beyond the scope of this thesis.

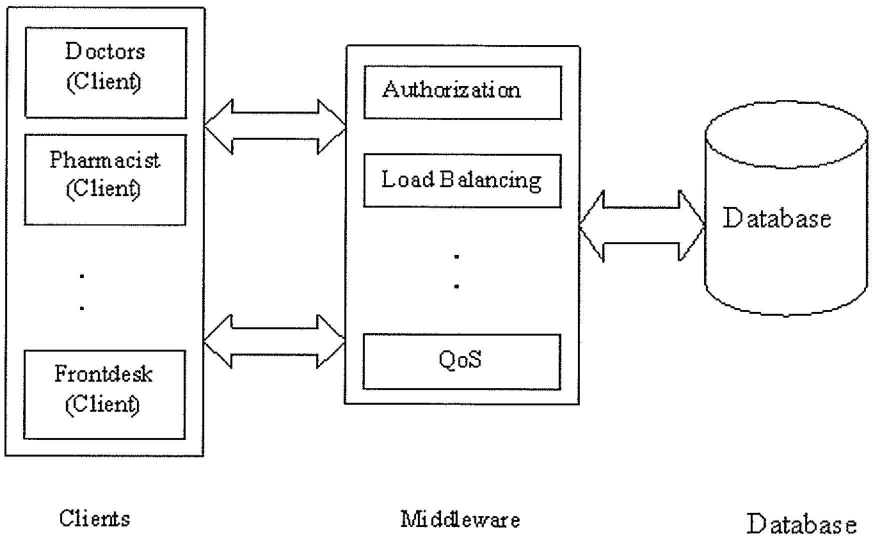


Figure 4.2: The Three-Tier Architecture of a Health Care System

Architecture of the Implementation

Figure 4.3 shows the architecture of the prototype implementation. Users access the application from various workstations spread across a hospital. The application's user interface allows users to access the database. All users are authenticated before they can access any of the access-based modules of the system (e.g., the modules for doctors). Once a user is authenticated the user's query is passed to the database server where the user's query is executed. Authentication of the user determines the privileges of the user. People at the frontdesk (Patient Scheduler (PS) in Figure 4.3) acts as an interface between patients and doctors. Patients interact with these personnel to arrange appointment times with doctors and/or care-providing personnel in the hospital. They are also responsible for updating the personal information of patients (as shown in Figure 4.4(a) – Frontdesk Package). From the PS, patient records move to the Care Providers (CP). CPs represent all the personnel in the hospital who diagnose patients. The encounter between patients and CP personnel results in CP medicating the patients (e.g, the CP could only prescribe medicines and/or forward patients to laboratory to have some tests performed and/or refer the patients to another CP to obtain a second opinion, etc (as shown in Figure 4.4 (b) – Doctor Package). Medication results in patients either responding to the treatment of the doctor or visiting the doctor again for post medication problems.

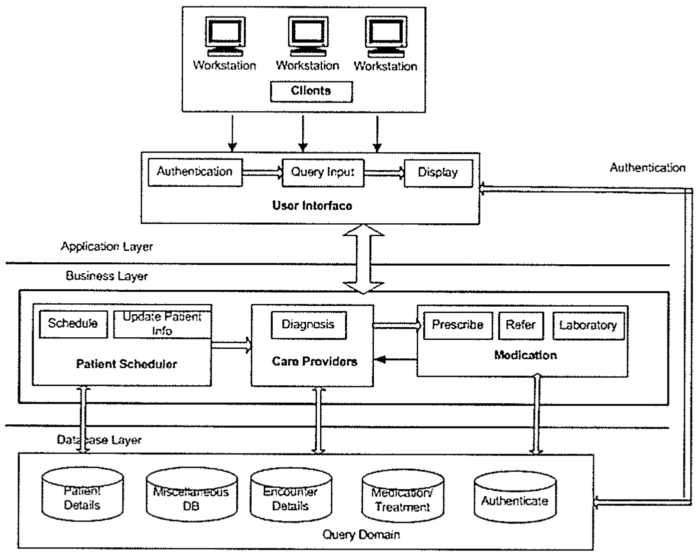


Figure 4.3: Architecture of the Present Implementation

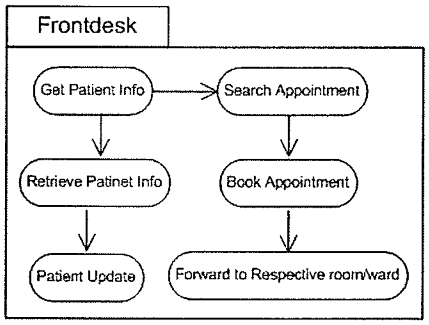


Figure 4.4 (a)

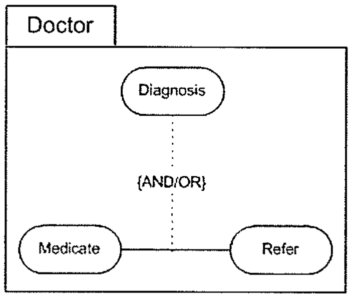


Figure 4.4 (b)

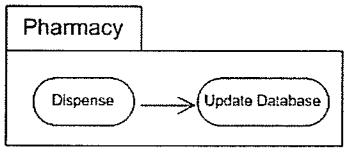


Figure 4.4 (c)

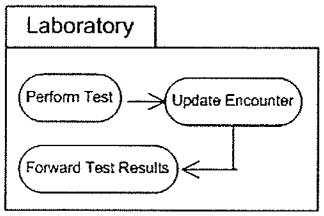


Figure 4.4 (d)

Figure 4.4: Components of the Software Architecture

4.2 Flow between Interface-Database

Figure 4.5 illustrates the application-database connection in the proposed integrated health care architecture. Both local and remote users' interface are shown. It is assumed that the client interface applications will interface to the network using industry-standard protocols (i.e., the Transmission Control Protocol/Internet Protocol (TCP/IP)) and that database queries will be supported using Open Database Connectivity (ODBC).

In a distributed environment, various departments/health care system cooperate in offering care to a patient. Health care application gives privilege to access information from a database in the local area network and the database of the participating hospital across the Internet. If the software is developed as a web application and a browser is used as the interface then information from databases should be made compatible to the browser by the use of appropriate markup languages, e.g, XML.

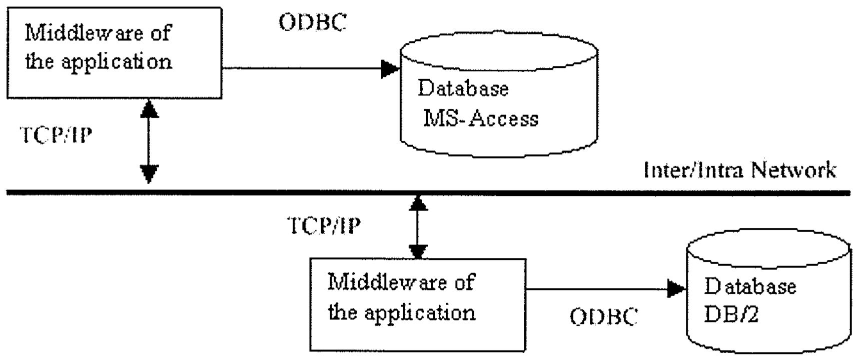


Figure 4.5: Flow from/to Database from/to User Interface

4.3 Patient Flow

The patient record is initialized when the patient visits the hospital for treatment. The patient's record is then accessed by the people at the front desk to review the patient's situation who forward the patient to a doctor. The doctor then views the patient's medical record and listens to the patient to determine the cause of the patient's visit and also to make an initial diagnosis about the patients health. Then depending on the condition of the patient, the doctor either directs the patient to a laboratory to obtain test reports (laboratory tests might include blood, urine and saliva tests). Doctors can also send patients to obtain reports (such as X-ray and ultrasound) or the doctor might just prescribe medication and update the medical record of the patient (generally laboratory tests and diagnostic imaging are offered in the same institution). A doctor also ensures that the patient is not allergic to the drug he/she prescribes by checking the patient's medical record. The patient takes any prescriptions to the pharmacist and the patient's medical record updated. Laboratory personnel and the imaging specialists perform their respective tests and forward the reports to the doctor (if need be). The doctor then explains to the patient the state of his/her health. Figure 4.6 illustrates this patient flow process.

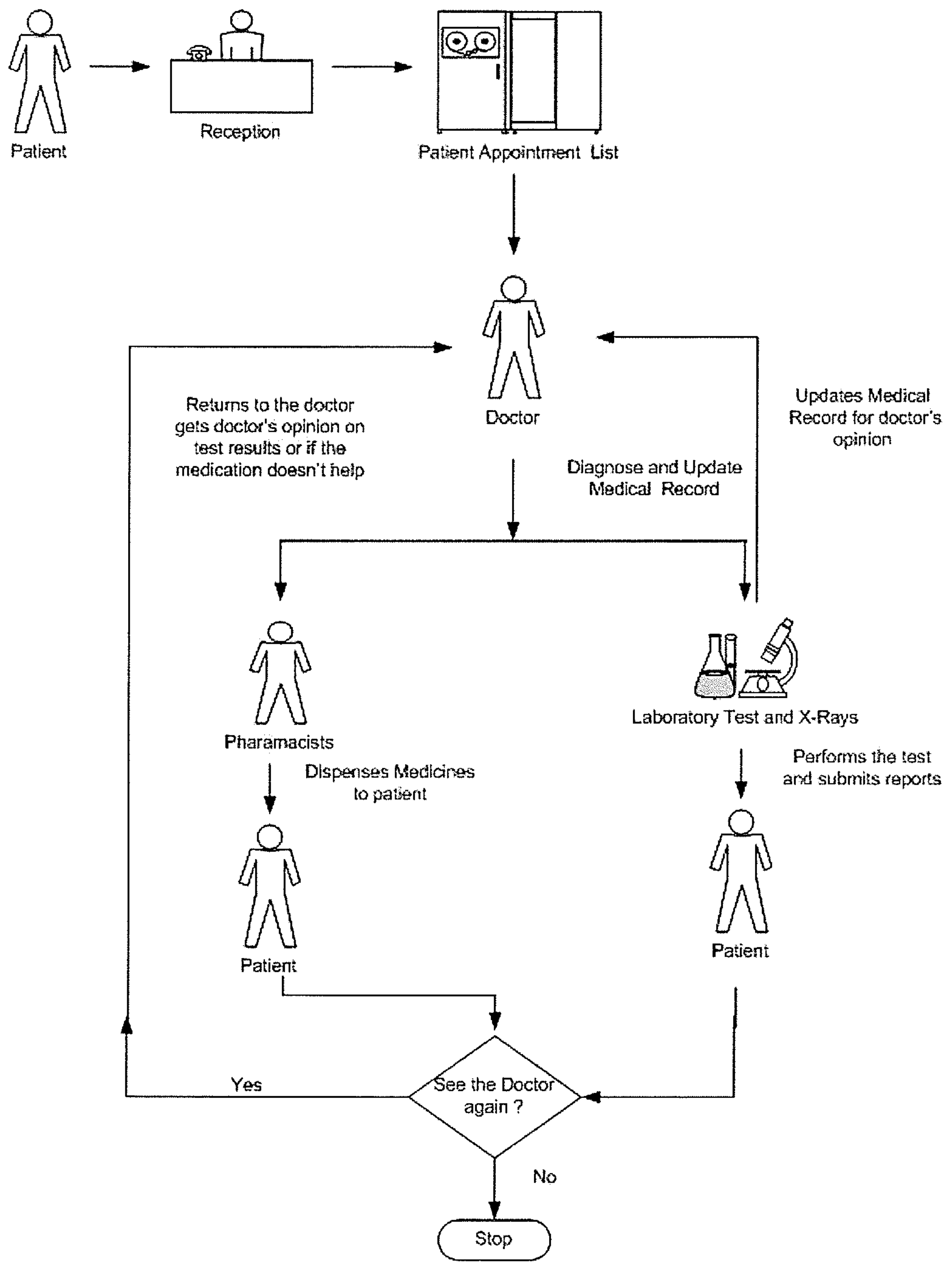

Patient
Reception
Patient Appointment List
Doctor
Returns to the doctor gets doctor's opinion on test results or if the medication doesn't help
Updates Medical Record for doctor's opinion
Diagnose and Update Medical Record
Pharamacists
Laboratory Test and X-Rays
Dispenses Medicines to patient
Performs the test and submits reports
Patient
Patient
Yes
See the Doctor again ?
No
Stop

Figure 4.6: Patient Flow in Health care

4.4 Algorithms and UML Diagrams

This section presents the object model and algorithms that capture the various functionalities of the design. The purpose of these algorithms and diagrams is to provide better understanding of the specifications and enhance communication between the persons developing the software and the persons who requested the software. "Developing software is a continuous process and is never complete" [Oe99]. Additions and modifications to the developed software is a regular process and, at times, if the design of the system is not well thought out, the modifications can prove expensive. Hence, software developers should interact with the users of the system at an intermediate stage to verify that the system being developed is indeed the system sought. The various UML diagrams represent the activities (e.g., starting from a patient's visit for ailment, medication received for the problem, etc.) and they help in visualizing the system in different perspectives. Some of the UML diagrams used in this thesis includes Use Case, State, Activity, Sequence, and Deployment diagram.

The follwong sub-sections explain the alogithms that is defined by the name of the sub-section.

Patient Search

One of the important activities in a health care organization is to maintain patients' records. A health care facility assigns a unique identity number to every patient who gets care from that facility. Information about a patient in that health care can be retrieved when this identity number is given as an input (patient's information retrieval is also necessary to commit updates on the record). Searching records by a patient's name is generally more useful because patients do not often remember their medical number. Hence, a name-based search should also be supported. Figure 4.7 searches for a patient record based either on the medical number or the name of the patient.

Algorithm 1: *Search Patient*

```
/* Variables
    x : Patient's Name or Medical Number
    $P_j$ : Patient j in the Patient List P
    name, medno : Attributes of Patients

    found : Boolean */

        SearchPatient(x) operation:
        found = false
        FOREVERY($P_j$ in the Patient List P)
            If (($P_j$.name == x) or ($P_j$.medno == x)) THEN
                found = true
                DisplayDetails($P_j$)
            ELSE
                found = false
                ShowMessage ("No such patient exists")
            ENDIF
        END{FOR}
```

Figure 4.7: Search for a Record — Patient

Create Patient Record

When a patient visits a health care organization, the personnel at the frontdesk searches for the patient's details. If no details about the patient exist, the front-desk personnel then creates a new record for the patient. This new record consists of the demographic and insurance details and the vital statistics of the patient. On committing the transaction, the unique patient number (generated automatically or one of the unique input fields, such as social insurance number) is mapped to that particular patient. Figure 4.8 shows the process of creating a patient record.

Password Verification

In a hospital several users access patient records. As described earlier, the users include doctors, frontdesk personnel, and laboratory personnel among others. Since patients medical records are confidential,unauthorized users should not be allowed to access these records. This requires that the respective personnel are allowed to view and edit only that information to which they are authorized. Figure 4.9 shows the user authentication algorithm based on the password verification approach. On authentication the system checks the role of the personnel in the health care system

Algorithm 2: *Insert Patient Record*

```
/* Variables
    x : Patient's Name
    $P_j$ : Patient j in the Patient List P
    name, address, contacts, insurance, and sin : Patient's Attributes
    medno : unique identifier

    found : Boolean */

    Create(record) operation:
    SearchPatient(x)
    IF SearchPatient(x).found = false THEN
        Get x.name, x.address, x.contacts, x.insurance, x.dob, x.sin, x.vitals
        Insert(x.name, x.address, x.contacts, x.insurance, x.sin, x.vitals)
        ShowMessage("Patient's Medical Record Created Successfully")
        x.medno = $P_j$.medno + 1
    ELSE
        ShowMessage("Patient details exist in the database")
    ENDIF
```

Figure 4.8: Insertion of Patient Record

and accordingly lets the user access parts of the software the user is authorized to access.

Room Booking

A patient is admitted to a hospital on the recommendation of a doctor. To book a room the necessary inputs are the patient details (e.g., name) and the dates for which the room is required.

On receiving the inputs the application checks the availability of a room of desired type for that particular date. A user can request a particular kind of room (e.g. corner room, non air conditioned room, etc); if such a room is available then the result is displayed as a success and the patient is mapped to that room. If the type of room requested by the patient is not available then a general search of room availability is performed. If any acceptable room is available, the room is reserved for the patient. Figure 4.10 captures this process .

Algorithm 3: *Password Verification*

```
/* Variables
    user : User of the application
    $User_j$ : User j in the User List, User
    uid, pwd : Attributes for authentication */

    PasswordVerification(Username, Password) operation:
        IF ((user.id == $User_j$) and (user.pwd == $User_j$.pwd)) THEN
            CASE
                IF (user == Doctor) THEN
                    Display(Doctor Form)
                ELSE
                IF (user == Frontdesk Personnel) THEN
                    Display(Frontdesk Form)
                ELSE
                IF (user == Laboratory Personnel) THEN
                    Display (Laboratory Form)
                ENDIF
            END{CASE}
        ELSE
            ShowMessage("Authentication Failure")
        ENDIF
```

Figure 4.9: Password Verification

Patient Medical Record Retrieval

A patient medical record can be generated in various ways. In this thesis, we generate a comprehensive medical record of the patient based on earlier encounters of the patient with health care professionals. To generate encounter-based medical records, search is not limited to only the encounters recorded in the patient-doctor encounter database but also records from the databases of laboratories (for image files and test results) and pharmacies (for drugs that the patient bought) should be retrieved. Figure 4.11 illustrates the retrieval of medical record.

Pharmacy Processing

Patients purchase drugs from the pharmacy. When a patient purchases drugs, the details (such as cost and quantity purchased) are recorded. A report is generated for the patient and the stock of drug is reduced by the same quatity purchased by the patient. Figure 4.12 illustrates code which captures this scenario.

Algorithm 4: *Book Room*

```
/* Variables
   $All_j$ : Room j in the List of all Rooms
   $Par_j$ : Room j (heated/non-heated, etc.) in the List of Rooms Par
           Note that Par $\subseteq$ All
   y : Patient's Name
   x : Date room is requested */

   found : Boolean */

   BookRoom(y, x) operation:
   found = false
   FOREVERY($Par_j$ in the room list Par)
       IF ($Par_j$.date == x) THEN
           found = true
           ShowMessage ("Room unavailable")
           CASE
               FOREVERY( $All_j$ in the Room list )
                   IF ($All_j$.date == x) THEN
                       found = true
                       ShowMessage("Room unavailable ")
                   ELSE
                       found = false
                       ShowMessage("Room of type $All_j$ Available")
                       IF patient accepts room $All_j$ THEN
                           Insert(y, x)
                       ENDIF
                   ENDIF
               END {FOR}
           END {CASE}
       ELSE
           ShowMessage("Room of type $Par_j$ Available")
           Insert(y, x)
       ENDIF
   END{FOR}
```

Figure 4.10: Book Room

UML Diagrams

Algorithm 5: *Medical Record*

```
/* Variables
   MedicalRecord : encounter records
   $Pat_j$ : Patient j in the Patient List Pat
   medno : unique patient identifier,
   Encounter, Laboratory and Department list : Database of interactions */

   Retrieve(MedicalRecord):
   FOREVERY( $Pat_j$ in the Patient List Pat)
      If ($Pat_j$.medno == x) THEN
         found = true
         FOREVERY ($Pat_j$ in Encounter List)
            RetrieveEncounter($Pat_j$.medno)
         END{FOR}
         FOREVERY ($Pat_j$ in Laboratory List)
            RetrieveLabTest($Pat_j$.medno)
         END{FOR}
         FOREVERY ($Pat_j$ in Department List)
            MiscDetails($Pat_j$.medno)
         END{FOR}
         GenerateReport(Encounter, Laboratory, Department)
      ELSE
         ShowMessage ("New Patient! No Record")
      ENDIF
   END{FOR}
```

Figure 4.11: Retrieve Medical Record

Algorithm 6: *Pharmacy Processing*

```
/* Variables
   x : Drug name
   qty : Quantity
   $Med_j$ : Medicine j in the Medicine List Med
   medname, cost : Attributes of $Med_j$ */

   Pharmacy Processing(x)
   GetMedicine(x, qty)
   FOREVERY($Med_j$ in the Medicine List 'Med')
      If ($Med_j$.medname == x) THEN
         RetriveDetails($Med_j$)
         ComputeCost($Med_j$.price, $Med_j$.qty)
         GenerateReport($Med_j$.medname, $Med_j$.qty, $Med_j$.cost, date)
         ReduceStock(x, qty)
      ENDIF
      If ($Med_j$.qty <= qty) THEN
         ShowMessage("Medicine running out of Stock ")
      ENDIF
   END{FOR}
```

Figure 4.12: Pharmacy

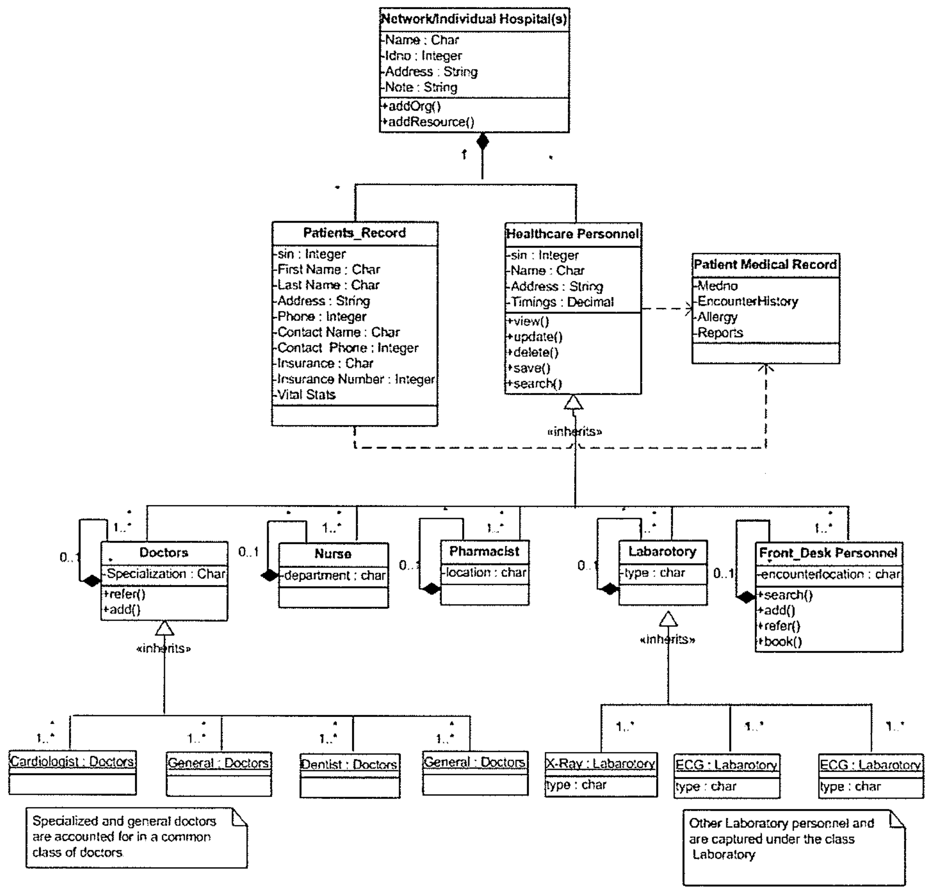


Figure 4.13: Patient Medical Record as Developed : Class- Object Diagram

Figure 4.13 represents the class model for the electronic patient record system described in this thesis. At the root is the hospital where a patient receives care and where these records are stored. In a hospital, there are several classes of employees and care providers who directly or indirectly interact with the medical records of a patient. Again, these include doctors, nurses, laboratory technicians, the front desk personnel, etc. All users inherit attributes (name, sin, address, etc) from the health care personnel class and groups of users in the environment also have special attributes (e.g., doctors have additional attributes like specialization). In Figure 4.13 there exists certain generic classes (e.g., doctors and laboratory) which represents other subclasses (such as specialist doctors and laboratory specialists). Each user has a distinct set of permissible executable operations on a patient's medical record (e.g., a nurse can update existing information about a patient but cannot add any information (or new finding) about the patient as only doctors will be authorized to do so). Patient medical records are used by the various personnel depicted in the above diagram.

Figures 4.14 and 4.15 depict the flow of patients in a health care system using Use case and sequence diagrams respectively.

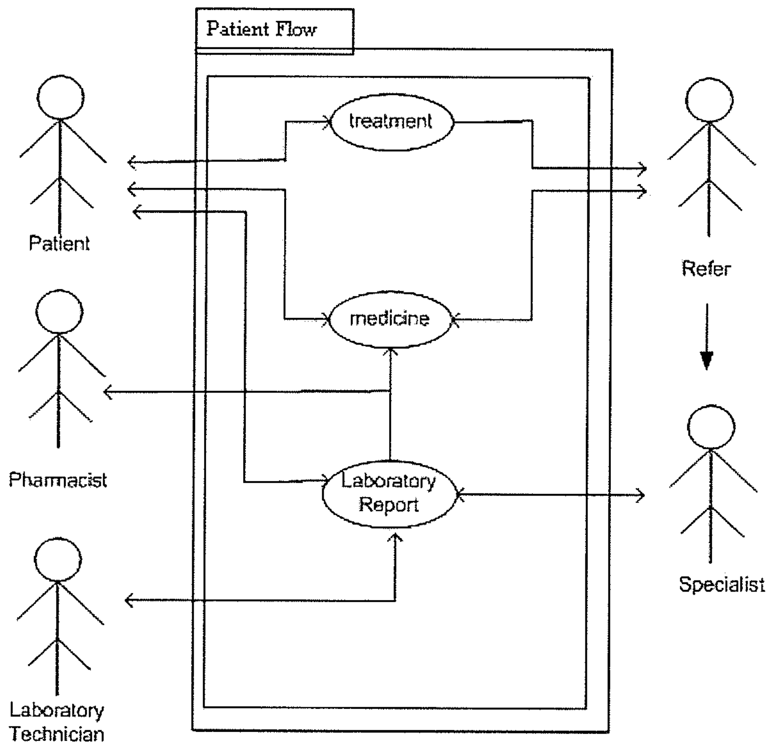


Figure 4.14: Patient Flow - Use Case Diagram

Figure 4.14 shows a use case diagram for a typical interaction between a patient and a doctor. To start, a patient goes to the doctor for treatment. The doctor then examines the patient and prescribes (in this case) drugs for the patient. The pharmacist dispenses any drugs prescribed for the patient. Sometimes when the doctor feels there is a need for a specialist's opinion, the doctor may refer a patient to a specialist who in turn may request a specific laboratory test. The patient then returns to the specialist with the test results.

The following sequence diagram shows one of the possible sequence of patient flow in a health care.

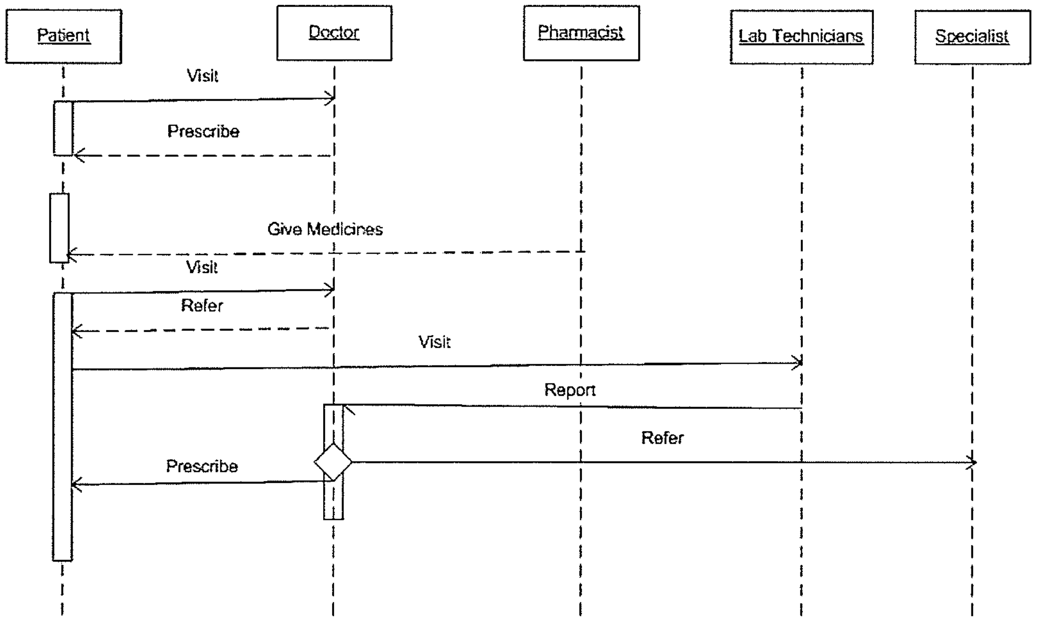


Figure 4.15: Patient Flow Sequence Diagram

The activities in Figure 4.15 are similar to that of Figure 4.14 but depicted as a sequence of events.

To view a patient's medical record the user needs authorization. Authentication is done to assure the safety and confidentiality of medical records. Sufficient security checks are performed to ensure that the medical records are not accessed by unauthorized personnel. The doctor (or any authorized user) after being authenticated searches for the medical record of a patient. If the search is success then the record is displayed otherwise an error message is displayed. Possible reasons for error messages include the patient being a first time visitor or the patient's records having been deleted (i.e., remove the patient record as active and put it in an archive). Figure 4.16 shows the activity diagram for authenticating an user and processing a user's request to view a patient's medical record.

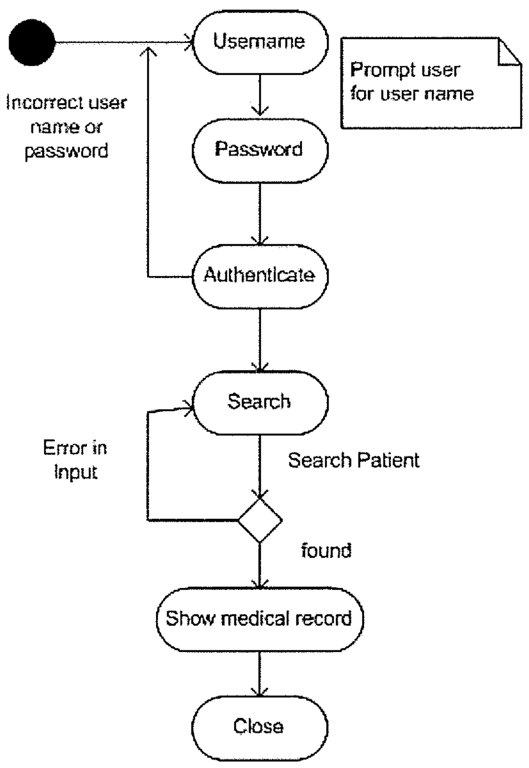


Figure 4.16: User Authentication/ Search Operation

The flow of patients in the health care system begins with the interaction between patients and the frontdesk personnel. The following diagrams (Figures 4.17 and 4.18) show the activities in the frontdesk using activitiy and State chart diagrams.

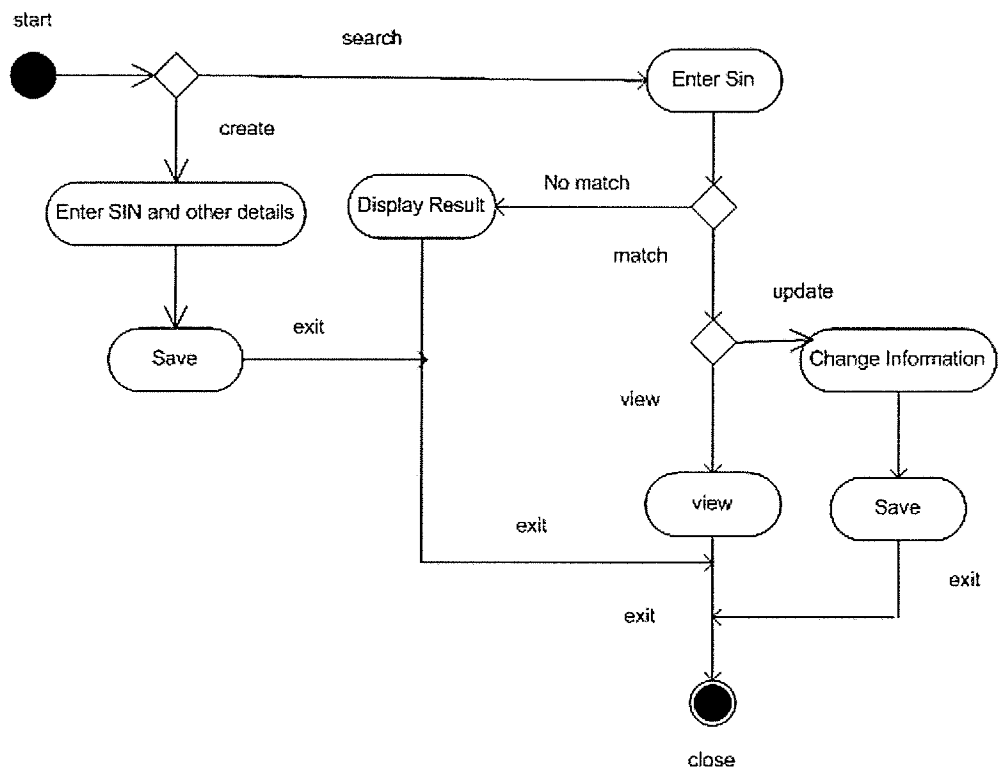


Figure 4.17: Two Operations in the Front Desk : Activity Diagram

Figure 4.17 represents two frontdesk operations: searching for patients and adding patients to the system. A patient's record is searched based on the sin number, if there is a match, the record is updated other wise error message is displayed and the user can start again by making a new search.

Frontdesk personnel also search for any available resources. The resources could be anything/anyone who could offer some health care service to a patient.

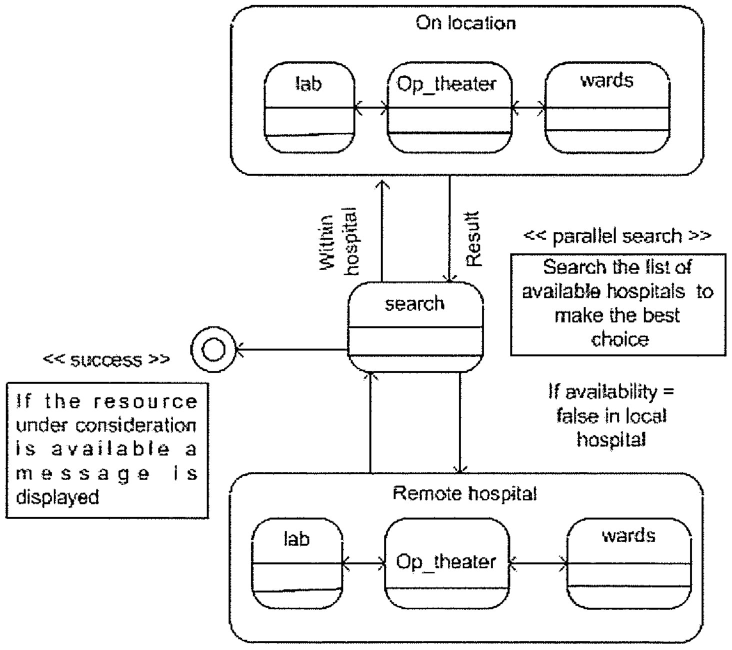

On location
lab
Op_theater
wards
Within hospital
Result
<< parallel search >>
Search the list of available hospitals to make the best choice
search
<< success >>
If the resource under consideration is available a message is displayed
If availability = false in local hospital
Remote hospital
lab
Op_theater
wards

Figure 4.18: Search for a Resource : State Chart

Figure 4.18 shows the idea behind making health care facility distributed. In case of emergency, access to the best available service a patient can get in the vicinity could save the patient's life. Applications should also communicate with other hospitals to retrieve information such as available number of beds in wards and operating theaters. A parallel search is done to retrieve the hospital, which can offer the best service at that moment.

The pharmacy supplies the drugs required by patients in the health care system. Hence, it is necessary to maintain a log of drugs that need to be replenished . Figure 4.19 shows the flow of activities during the update of drug inventory information in the pharmacy database. The requested drug's available stock in the database is retrieved. The process is repeated for the desired number of drugs. All drugs with low stock are recorded for re-ordering.

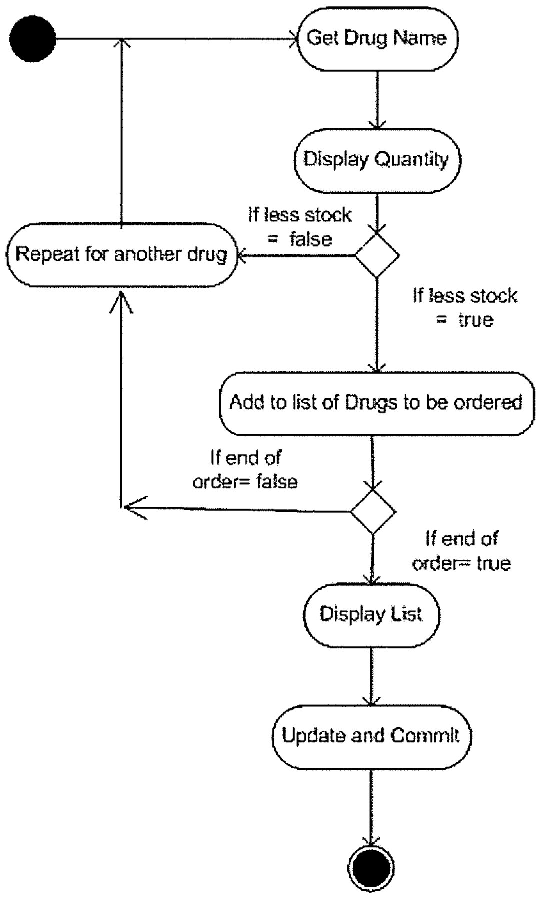


Figure 4.19: Drug Update in Pharmacy Inventory

Patients visit doctors for treatment. A doctor, depending upon the diagnosis, could advise the patient to have certain tests performed in the laboratory. The activities in the laboratory should also be captured.

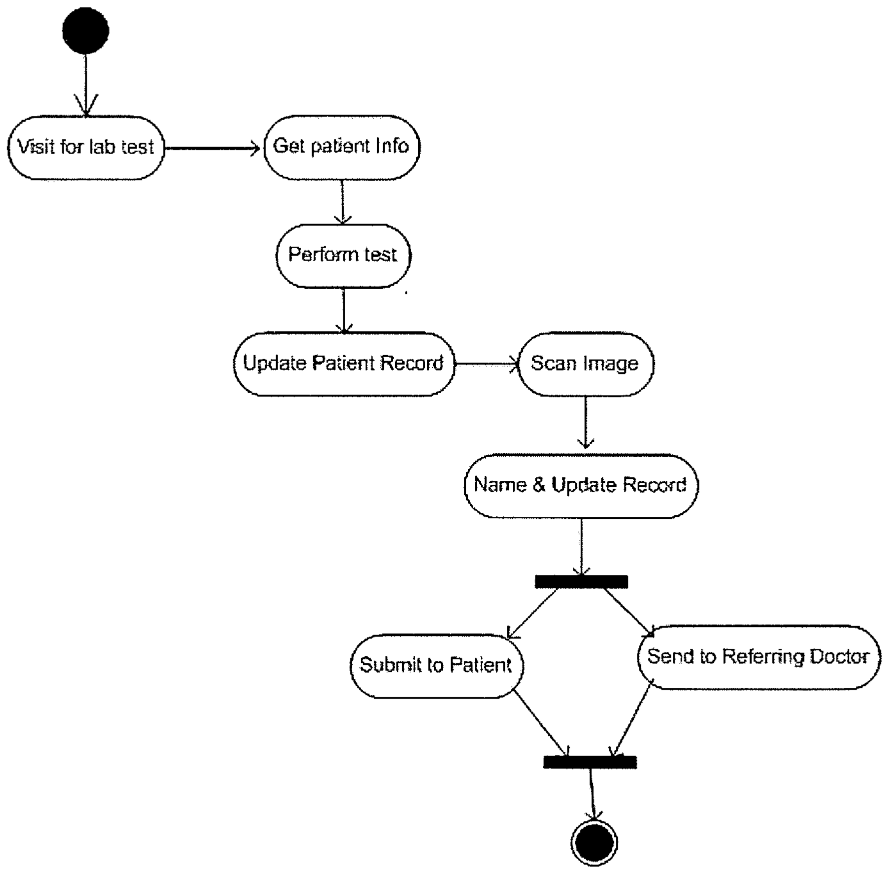


Figure 4.20: Laboratory Activity Diagram

Figure 4.20 shows the activities (e.g., blood samples, etc) in a laboratory. A patient visits the laboratory to perform any requested test. The laboratory technician may scan results and images, if any, updates the patient's record.

Figures 4.21 and 4.22 capture ways patients can view their medical record.

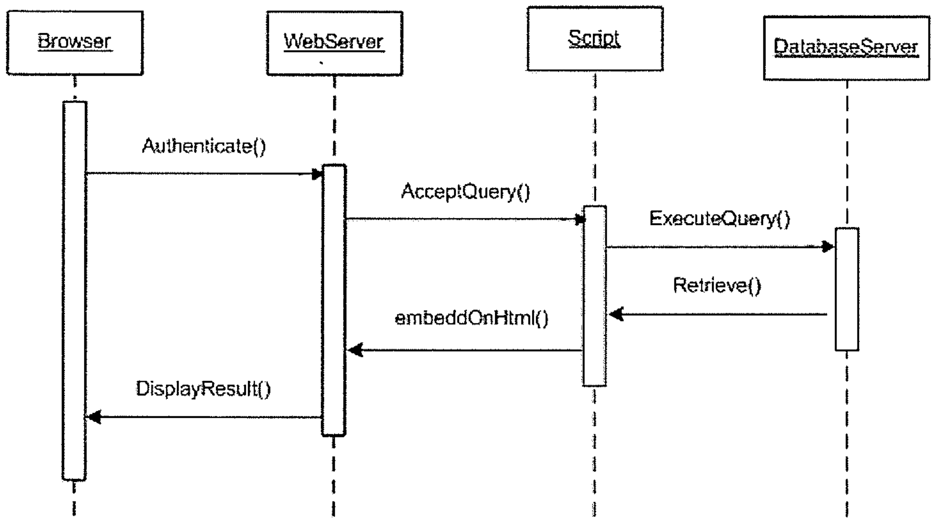


Figure 4.21: Patients Accessing Medical Records : Sequence Diagram

The application should enable patients to view their medical status online. Like all other users, patients are authenticated and a patient's query is executed to retrieve the patient's information from a particular hospital. Figure 4.21 represents this use case scenario as a sequence diagram.

In a distributed environment, a patient should have access to not only information from one health care provider where the patient regularly receives care, but also should be able to retrieve/view the records from the participating hospitals where he/she received care.

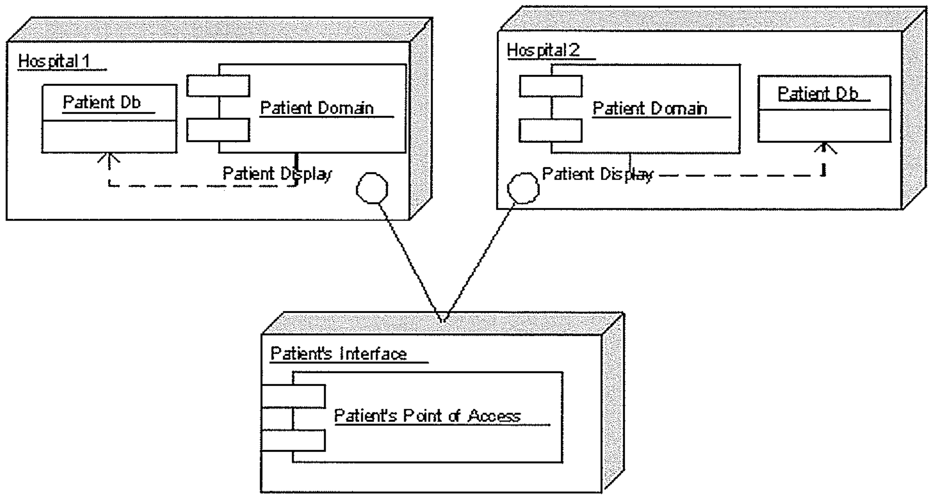


Figure 4.22: Medical Record Retrieval from Participating Hospitals

Figure 4.22 shows the implementation diagram for the patient's access of a medical record. A patient can have access to any number of health care providers'.

Chapter 5

Implementation

" There are two ways of constructing a software design; one way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult." - C. A. R. Hoare .

This chapter explains the implementation strategies and the various screen shots of the implementation. The screen shots shown are generated when various kinds of users (e.g. as a doctor, pharmacist, laboratory technician and the front desk personnel) interact with the system.

5.1 Implementation Environment

The prototype was developed and tested on a heterogeneous database environment (Access and DB2) distributed across a cluster of windows workstations (running Windows NT). The implementation of the prototype is in Microsoft Visual Basic 6.0, as this software supports object oriented programming and aids in the rapid development of prototypes including GUI-based interactive systems.

5.2 Implementation Strategies

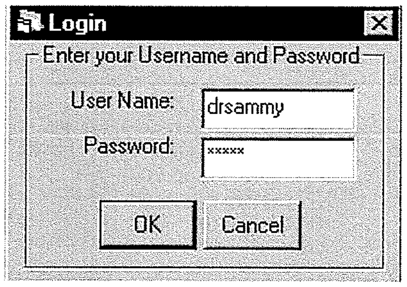
When developing the software one important consideration was the end-users. Since the end-users are the people who will use the software, they should be involved during the entire implementation (and the customization of the software for a particular user profile). A thorough research on user inputs for similar applications was made to study the various advantages that users would want to get from such systems. The implementation was developed progressively because this helps in designing the interoperability between the various user profiles . A medical record system integrates the various autonomous departments, otherwise the overall benefit cannot be realized; hence we provide a wrapper integrator to encapsulate the various modules.

5.3 Run-Time Behaviour

The following screen shots illustrates user interactions relating a patient's visit to a hospital for treatment. As described, a medical record is generated based on the various encounters the patient has had with the medical personnel in the hospital.

The developed implementation when executed prompts the user for his/her user name and password to authenticate the user. All user names are assigned with respect to the user's role in the health care domain. For instance, a doctor in the domain has a username starting with "dr" and a front desk personnel has his/her user name beginning with "fr". The prefix (dr, fr) determines the access rights for the various user profile.

Figure 5.1 shows the login screen for the application to validate the user. The form allows users to submit their user name and password and/or cancel the operation. If the password supplied is incorrect then an error is reported and the application prompts the user to re-try with the correct password. Also, if the user name does not exist, an error message is generated informing the user to change the user name entered.



User Name: drsammy

Password: xxxxx

Figure 5.1: Form for user authentication

The following figures (Figure 5.1 - 5.18) illustrate some of the activities a front desk personnel performs in a health care system.

The front desk personnel are primarily responsible for scheduling patients with doctors. They are also responsible for generating bills for using health care resources.

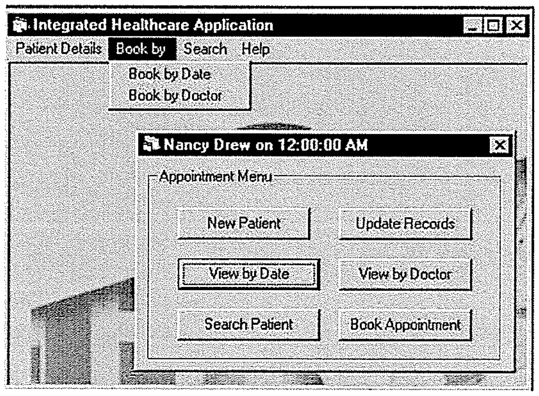
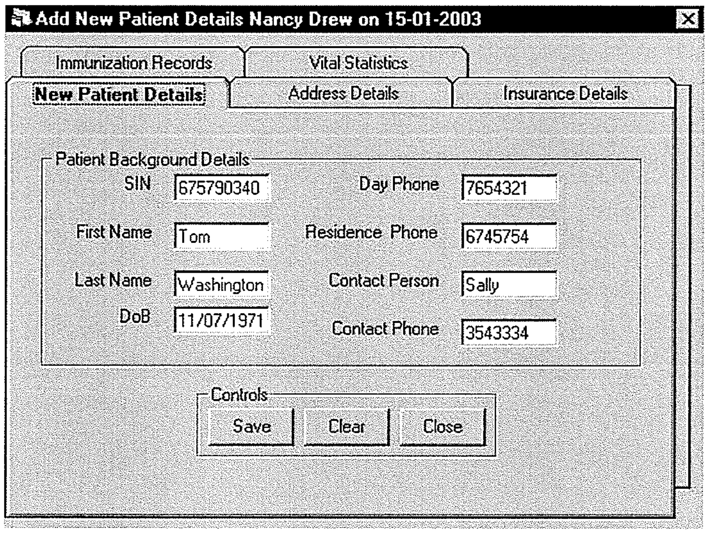


Figure 5.2: Form showing the front desk menu

Figure 5.2 shows the start up screen with the menu details for a front desk user.

For booking appointments, the patient details are collected. This includes the demographic, insurance and the immunization details as shown in Figures 5.3 to 5.7.


Add New Patient Details Nancy Drew on 15-01-2003

Patient Background Details

SIN	675790340	Day Phone	7654321
First Name	Tom	Residence Phone	6745754
Last Name	Washington	Contact Person	Sally
DoB	11/07/1971	Contact Phone	3543334

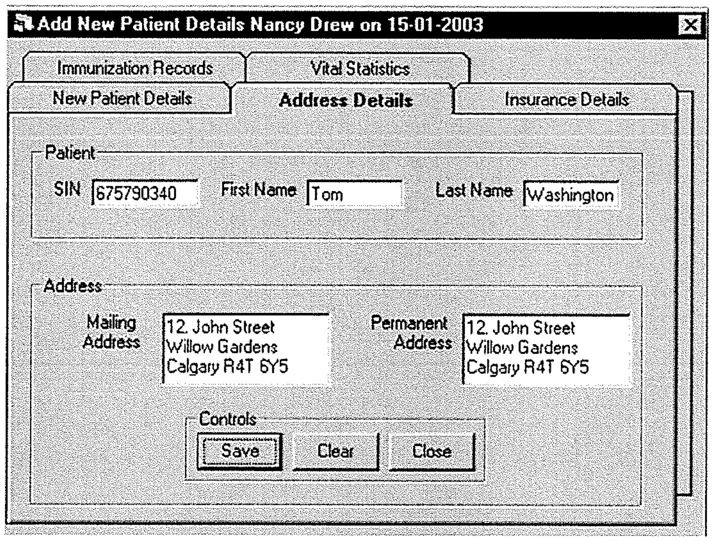
Figure 5.3: Form showing patient details

Figure 5.3 shows the form that is used by the front desk personnel to create new patient records. The user inputs the patient's personal information, this includes the patient's identification number (I use patient's SIN to gurantee uniqueness of the patient's medical records), address, phone number, contact person's name and phone number. This form and the subsequent forms are designed with user optons to save, clear and close the form while certain forms having more user controls for customized operations.

Figure 5.4 shows the form front desk personnel use to capture the address details of a patient. The patient's name and medical number automatically are copied from the form shown in Figure 5.3. The form allows patients to specify both their temporary and permanent address for future communication.

Figure 5.5 shows the form front desk personnel uses to capture the insurance details of a new patient.

Figure 5.6 shows the form where a patient's immunization records are updated. Front


Add New Patient Details Nancy Drew on 15-01-2003

Patient

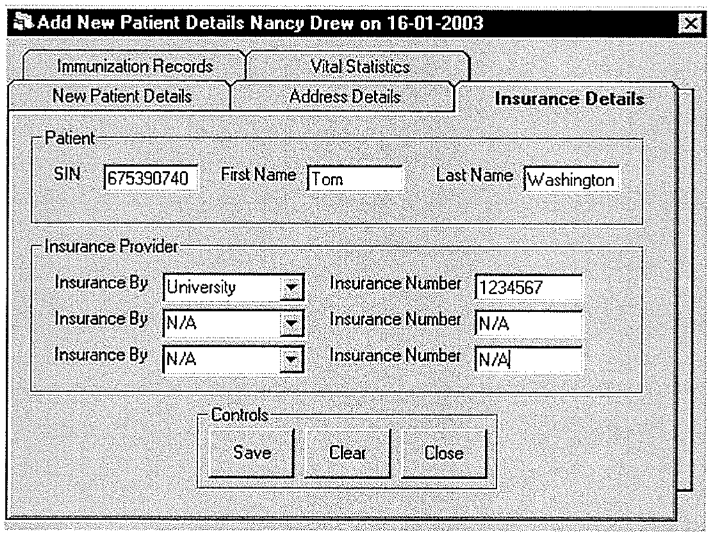
SIN 675790340 First Name Tom Last Name Washington

Address

Mailing Address: 12. John Street / Willow Gardens / Calgary R4T 6Y5

Permanent Address: 12. John Street / Willow Gardens / Calgary R4T 6Y5

Figure 5.4: Form showing patient's address details


Add New Patient Details Nancy Drew on 16-01-2003

Patient

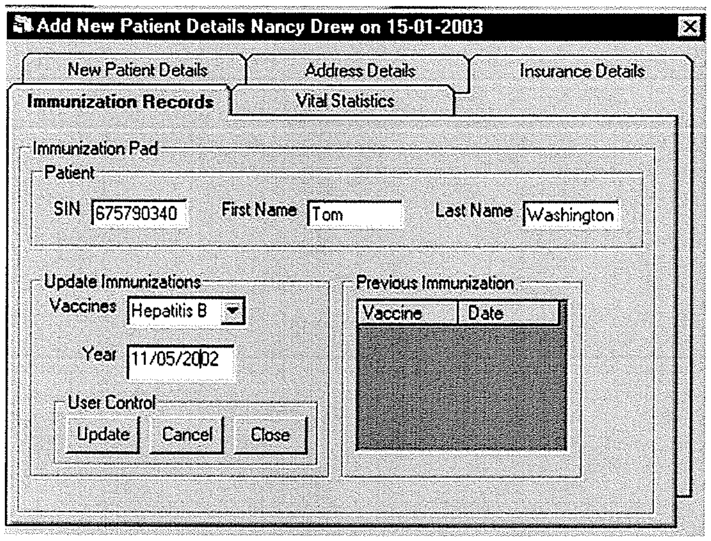
SIN 675390740 First Name Tom Last Name Washington

Insurance Provider

Insurance By	University	Insurance Number	1234567
Insurance By	N/A	Insurance Number	N/A
Insurance By	N/A	Insurance Number	N/A

Figure 5.5: Form showing patient insurance details

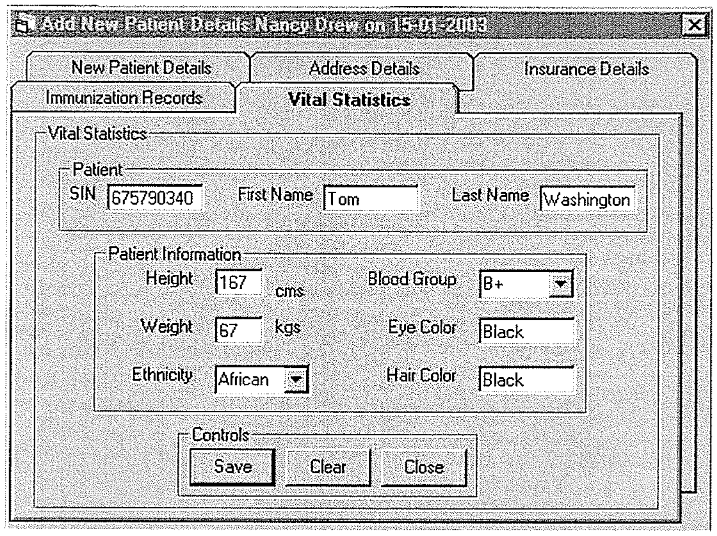
desk personnel are also gather vital statistics information on the patient as shown in Figure 5.7.



SIN	675790340
First Name	Tom
Last Name	Washington
Vaccines	Hepatitis B
Year	11/05/2002

Vaccine	Date

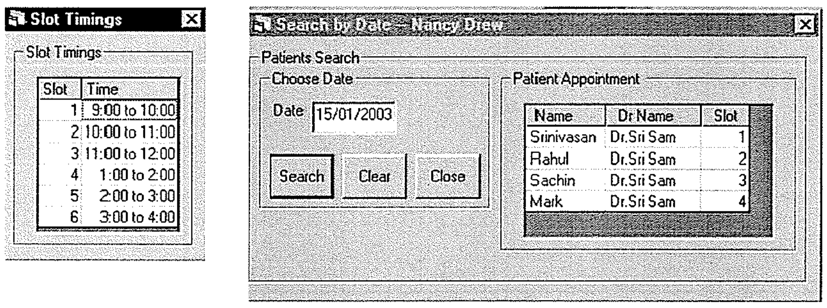
Figure 5.6: Form showing immunization details



SIN	675790340
First Name	Tom
Last Name	Washington
Height	167 cms
Weight	67 kgs
Ethnicity	African
Blood Group	B+
Eye Color	Black
Hair Color	Black

Figure 5.7: Vital statistics of patients

Appointment times are modelled as slots over the day and a slot represents one finite period of time. While scheduling patients with doctors, a number of customized queries have to be modeled. The following form (Figure 5.8) captures the scenario when a patient requests an appointment on a particular day.



Slot Timings

Slot Timings

Slot	Time
1	9:00 to 10:00
2	10:00 to 11:00
3	11:00 to 12:00
4	1:00 to 2:00
5	2:00 to 3:00
6	3:00 to 4:00

Search by Date -- Nancy Drew

Patients Search

Choose Date

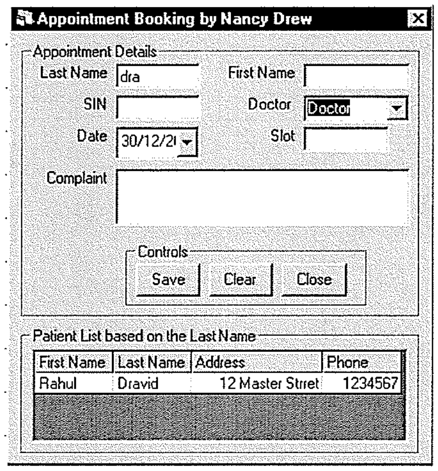
Date: 15/01/2003

Patient Appointment

Name	Dr Name	Slot
Srinivasan	Dr.Sri Sam	1
Rahul	Dr.Sri Sam	2
Sachin	Dr.Sri Sam	3
Mark	Dr.Sri Sam	4

Figure 5.8: Form showing appointment search - date based

When date is the constraint to a patient, the front desk personnel performs a search to view all doctors that may have available appointment times on the date requested (as shown in Figure 5.8). Users can clear the result of the query and view a different date or they can close the form. If the patient is willing to be scheduled in one of the available times then the user points to the preferred time and Figure 5.9 is displayed.



Appointment Booking by Nancy Drew

Appointment Details

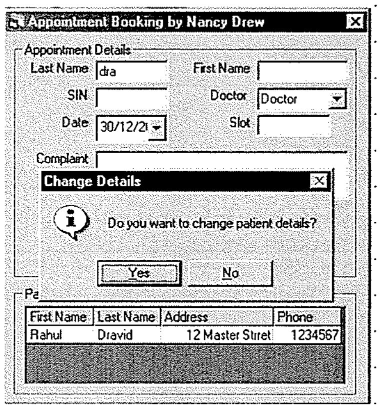
Last Name	dra	First Name	
SIN		Doctor	Doctor
Date	30/12/2(	Slot	
Complaint			

Patient List based on the Last Name

First Name	Last Name	Address	Phone
Rahul	Dravid	12 Master Stret	1234567

Figure 5.9: Form showing appointment booking - flexible search

In Figure 5.9, a "wild card" search is performed on last name, returning all the patient details whose name starts "dra". The corresponding name is chosen and the user makes sure that correct patient is considered for an appointment by verifying the address and phone number of the patient.


Appointment Booking by Nancy Drew

Appointment Details

Last Name	dra	First Name	
SIN		Doctor	Doctor
Date	30/12/2	Slot	

Complaint

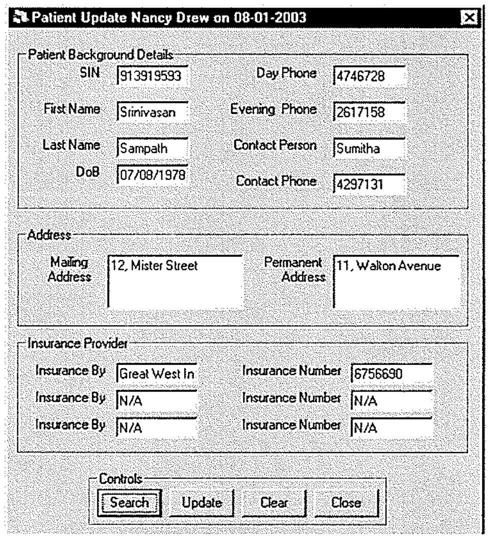
Change Details

Do you want to change patient details?

First Name	Last Name	Address	Phone
Rahul	Dravid	12 Master Stret	1234567

Figure 5.10: Form showing appointment booking - changing patient details

If there is a need to change the patient details, the user clicks on the "Yes" option of the dialogue box (shown in Figure 5.10) to modify the patient details.


Patient Update Nancy Drew on 08-01-2003

Patient Background Details

SIN	913919593	Day Phone	4746728
First Name	Srinivasan	Evening Phone	2617158
Last Name	Sampath	Contact Person	Sumitha
DoB	07/08/1978	Contact Phone	4297131

Address

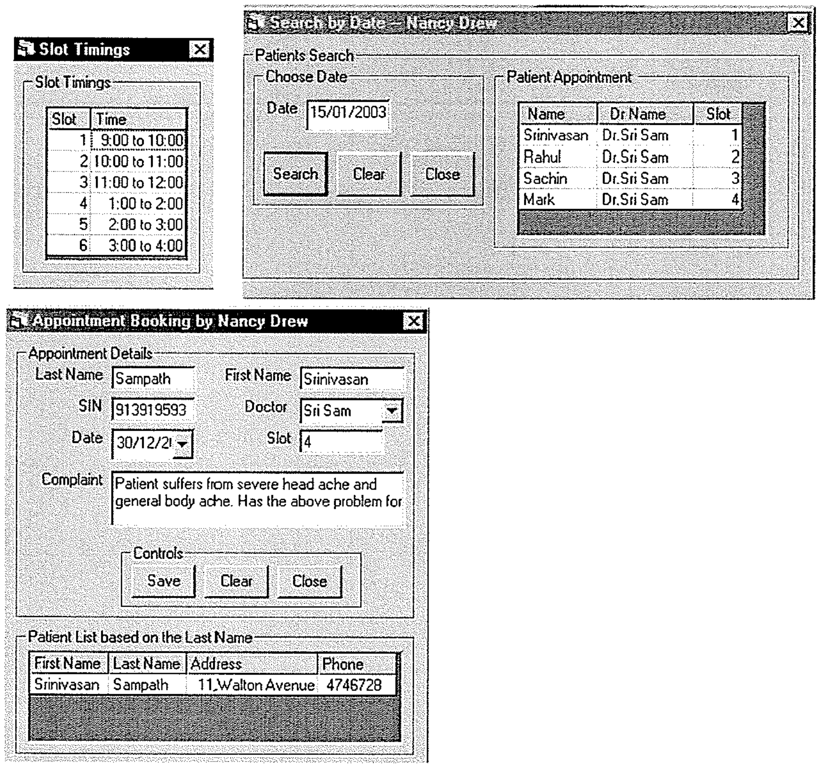
Mailing Address	12, Mister Street	Permanent Address	11, Walton Avenue

Insurance Provider

Insurance By	Great West In	Insurance Number	6756690
Insurance By	N/A	Insurance Number	N/A
Insurance By	N/A	Insurance Number	N/A

Figure 5.11: Form showing an update on patient details

The patient's details are populated in Figure 5.11 from the unique record of the patient chosen using the form shown in Figure 5.10. The user can then change the applicable details and closes the form. If no update of patient data is required, the user clicks the "No" option of the dialogue box (as shown in Figure 5.10) to continue and book the patient appointment.Again,the patient's details are passed to the corresponding text boxes of the form (originally shown as Figure 5.9 and now revised in Figure 5.12.).



Slot	Time
1	9:00 to 10:00
2	10:00 to 11:00
3	11:00 to 12:00
4	1:00 to 2:00
5	2:00 to 3:00
6	3:00 to 4:00

Date 15/01/2003

Name	Dr Name	Slot
Srinivasan	Dr.Sri Sam	1
Rahul	Dr.Sri Sam	2
Sachin	Dr.Sri Sam	3
Mark	Dr.Sri Sam	4

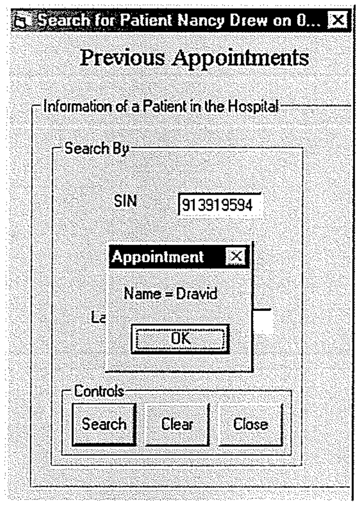
Last Name Sampath First Name Srinivasan
SIN 913919593 Doctor Sri Sam
Date 30/12/2 Slot 4
Complaint Patient suffers from severe head ache and general body ache. Has the above problem for

First Name	Last Name	Address	Phone
Srinivasan	Sampath	11,Walton Avenue	4746728

Figure 5.12: Form showing appointment booking

Often, patients' forget the name of the doctor that last treated them but might want to be treated (or not treated) by the same doctor. Hence, earlier instances of patient's visits should be able to be referred to when booking an appointment.

Figure 5.13 shows a screen that supports two kinds of searches for obtaining patient information. A patient's details can be searched for either by their medical number or their name. When the medical number is submitted, the application outputs the corresponding patient's name and displays the patient's previous appointment details


Search for Patient Nancy Drew on 0...

Previous Appointments

Information of a Patient in the Hospital

Search By

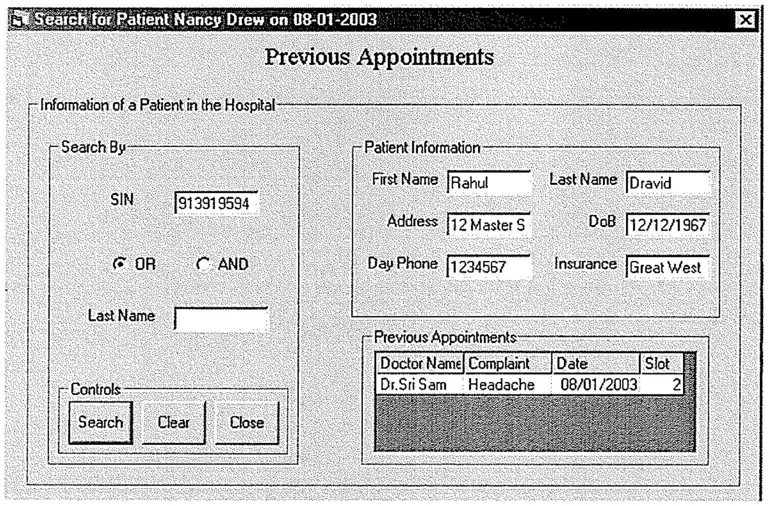
SIN: 913919594

Appointment

Name = Dravid

Figure 5.13: Form showing patient search - 1

as shown in Figure 5.14.


Search for Patient Nancy Drew on 08-01-2003

Previous Appointments

Information of a Patient in the Hospital

Search By

SIN: 913919594

(● OR) ○ AND

Last Name:

Patient Information

First Name	Rahul	Last Name	Dravid
Address	12 Master S	DoB	12/12/1967
Day Phone	1234567	Insurance	Great West

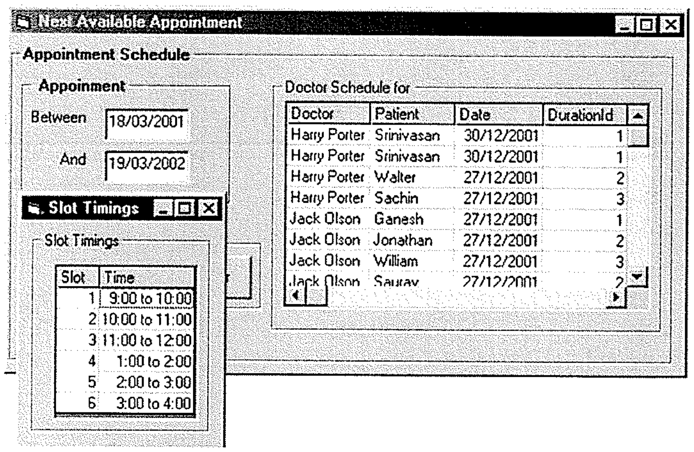
Previous Appointments

Doctor Name	Complaint	Date	Slot
Dr.Sri Sam	Headache	08/01/2003	2

Figure 5.14: Form showing patient search - 2

Figure 5.14 displays the details like the demographic details and the previous appointment dates and times, reason for the patient's visit for encounters with various doctors.

Front desk users should also retrieve queries of users might want to have the first appointment available with a particular doctor between two dates.



Between 18/03/2001

And 19/03/2002

Doctor	Patient	Date	DurationId
Harry Porter	Srinivasan	30/12/2001	1
Harry Porter	Srinivasan	30/12/2001	1
Harry Porter	Walter	27/12/2001	2
Harry Porter	Sachin	27/12/2001	3
Jack Olson	Ganesh	27/12/2001	1
Jack Olson	Jonathan	27/12/2001	2
Jack Olson	William	27/12/2001	3
Jack Olson	Saurav	27/12/2001	2

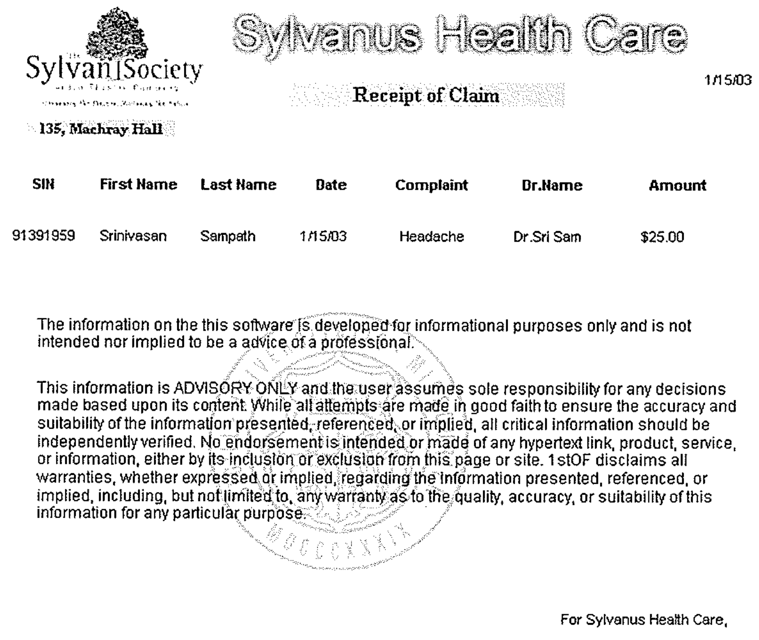
Slot	Time
1	9:00 to 10:00
2	10:00 to 11:00
3	11:00 to 12:00
4	1:00 to 2:00
5	2:00 to 3:00
6	3:00 to 4:00

Figure 5.15: Form showing appointment search - Doctor-based

Figure 5.15 shows the form that displays a list of unavailable (i.e. booked) appointment times with a doctor between two requested dates.

Following a patient's visit, the front desk personnel also generate bills to update the insurance companies. A patient can also receive a hard copy of the bill, hence the application supports ways for generating such bills. Figure 5.16 shows one such sample bill.

The front desk users also check for the availability of resources in the local hospital and resource availability in other health care institutions. Figure 5.17 allows the user to fill in details of the date requested and the resources requested. The person and the hospital requesting the resource are populated automatically. On clicking the "Submit" button information is retreived from the database of the participating hospital (as shown in Figure 5.18).



Sylvan Society

135, Machray Hall

Sylvanus Health Care

Receipt of Claim

1/15/03

SIN	First Name	Last Name	Date	Complaint	Dr.Name	Amount
91391959	Srinivasan	Sampath	1/15/03	Headache	Dr.Sri Sam	$25.00

The information on the this software is developed for informational purposes only and is not intended nor implied to be a advice of a professional.

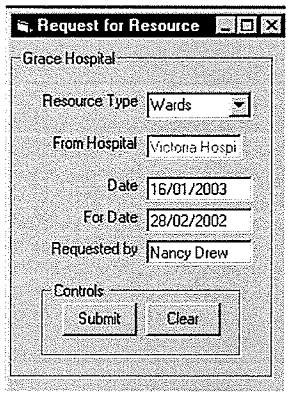
This information is ADVISORY ONLY and the user assumes sole responsibility for any decisions made based upon its content. While all attempts are made in good faith to ensure the accuracy and suitability of the information presented, referenced, or implied, all critical information should be independently verified. No endorsement is intended or made of any hypertext link, product, service, or information, either by its inclusion or exclusion from this page or site. 1stOF disclaims all warranties, whether expressed or implied, regarding the information presented, referenced, or implied, including, but not limited to, any warranty as to the quality, accuracy, or suitability of this information for any particular purpose.

For Sylvanus Health Care,

Figure 5.16: Form showing appointment bill generation

The following figures (Figure 5.19 - 5.25) ilustrate some of the activities done by a doctor once he/she is authenticated in to the system. The doctor's menu is shown in Figure 5.19.

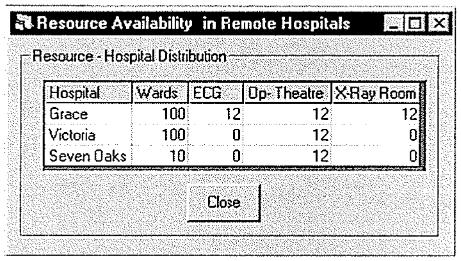
When a doctor is authenticated, the appointments the doctor has for the day are displayed along with a set of menu options. On choosing a particular patient and clicking one of the menu options, the doctor performs the selected action on the chosen patient (e.g., choosing the first record and then choosing the diagnosis button allows doctors to diagnose the first patient (as shown in Figure 5.20).


Request for Resource

Grace Hospital

Resource Type	Wards
From Hospital	Victoria Hospi
Date	16/01/2003
For Date	28/02/2002
Requested by	Nancy Drew

Figure 5.17: Form showing request for resource


Resource Availability in Remote Hospitals

Resource - Hospital Distribution

Hospital	Wards	ECG	Op- Theatre	X-Ray Room
Grace	100	12	12	12
Victoria	100	0	12	0
Seven Oaks	10	0	12	0

Figure 5.18: Form showing resource availability in other health care institutions

After diagnosis a doctor may write a prescription for the patient. Figure 5.21 illlustrates this process.

Doctors may also refer patients to specialists. In my implementation I have captured referral as an internal process, where doctors within the organization refer patients to each other. Figure 5.22 shows the referral form.

Doctors view a patient's medical record to get a better understanding of the patient. Hence, my implementation retrieves information about the patient from different departments of the health care institution to present a consolidated view. Figure 5.23 shows a sample encounter record.

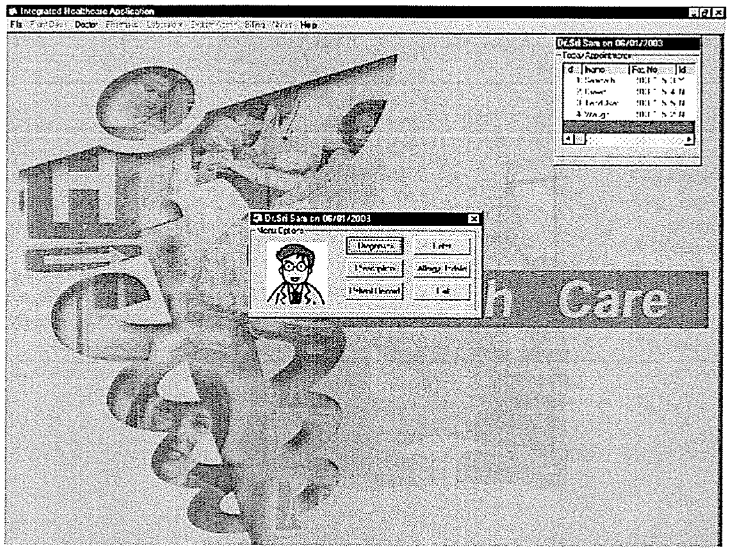
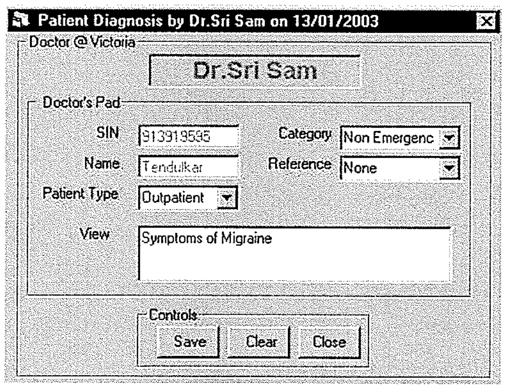


Figure 5.19: Form showing Doctor's Menu



Patient Diagnosis by Dr.Sri Sam on 13/01/2003

Doctor @ Victoria

Dr.Sri Sam

Doctor's Pad

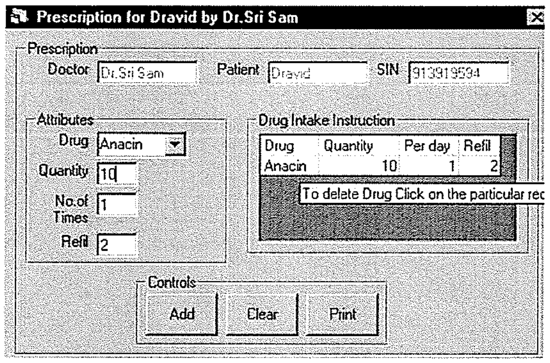
SIN	913919595	Category	Non Emergenc
Name	Tendulkar	Reference	None
Patient Type	Outpatient		

View: Symptoms of Migraine

Figure 5.20: Form showing patient diagnostics

In the encounter record images (e.g., x-ray or ECG) may be included. Doctors have the freedom to zoom in/out images. This allows doctors to view diagnostic images more clearly to make more accurate inferences.

The vertical scroll bar to the right of the image (shown in Figure 5.25) allows users to



Prescription for Dravid by Dr.Sri Sam

Prescription

Doctor	Dr.Sri Sam
Patient	Dravid
SIN	913919534

Attributes

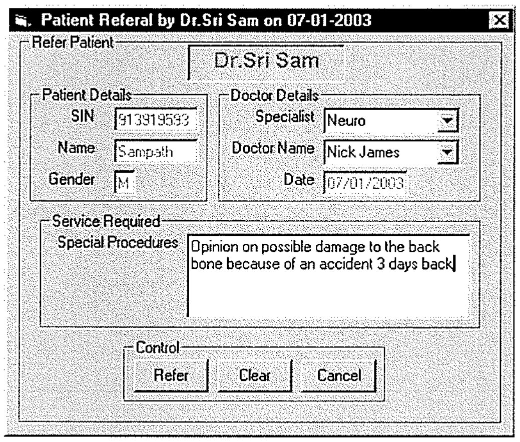
Drug	Anacin
Quantity	10
No.of Times	1
Refil	2

Drug Intake Instruction

Drug	Quantity	Per day	Refil
Anacin	10	1	2

To delete Drug Click on the particular rec

Figure 5.21: Form showing prescription filling



Patient Referal by Dr.Sri Sam on 07-01-2003

Refer Patient

Dr.Sri Sam

Patient Details

SIN	913919593
Name	Sampath
Gender	M

Doctor Details

Specialist	Neuro
Doctor Name	Nick James
Date	07/01/2003

Service Required

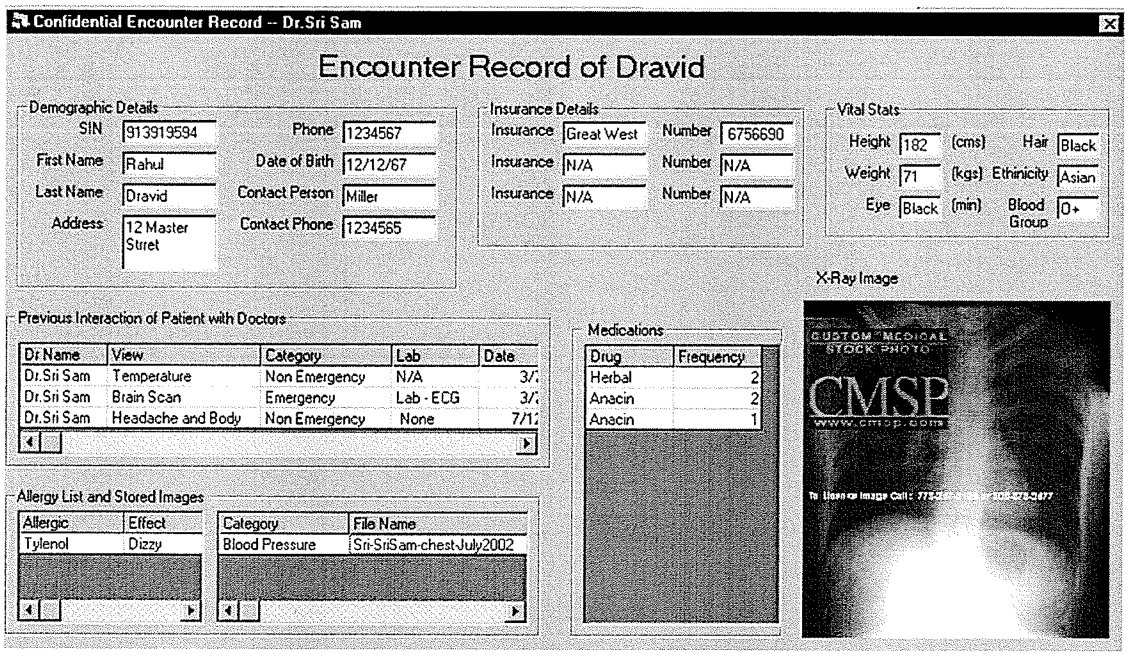
Special Procedures: Opinion on possible damage to the back bone because of an accident 3 days back

Figure 5.22: Form showing patient referring

zoom in or zoom out the image. Figure 5.24 and Figure 5.25 shows the same image before and after being zoomed respectively.

After viewing the image, the form can be closed by clicking the close button, which takes the user back to the form shown in Figure 5.23.

The pharmacy in a health care instituion, dispenses medicine to patients and updates the drug information in the database.



Confidential Encounter Record -- Dr.Sri Sam

Encounter Record of Dravid

Demographic Details

SIN	913919594	Phone	1234567
First Name	Rahul	Date of Birth	12/12/67
Last Name	Dravid	Contact Person	Miller
Address	12 Master Stret	Contact Phone	1234565

Insurance Details

Insurance	Great West	Number	6756690
Insurance	N/A	Number	N/A
Insurance	N/A	Number	N/A

Vital Stats

Height	182	(cms)	Hair	Black
Weight	71	(kgs)	Ethinicity	Asian
Eye	Black	(min)	Blood Group	O+

Previous Interaction of Patient with Doctors

Dr Name	View	Category	Lab	Date
Dr.Sri Sam	Temperature	Non Emergency	N/A	3/
Dr.Sri Sam	Brain Scan	Emergency	Lab - ECG	3/
Dr.Sri Sam	Headache and Body	Non Emergency	None	7/1

Medications

Drug	Frequency
Herbal	2
Anacin	2
Anacin	1

Allergy List and Stored Images

Allergic	Effect
Tylenol	Dizzy

Category	File Name
Blood Pressure	Sri-SriSam-chest-July2002

X-Ray Image

Figure 5.23: Form showing an encounter record

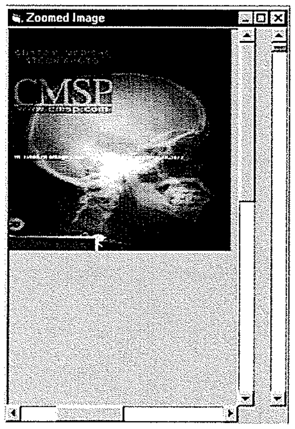


Figure 5.24: Form showing image before zoom

Figure 5.26 shows a typical form where a pharmacist dispenses drugs.

In Figure 5.26 illustrates drug dispensing by a pharmacist. To delete a wrong entry, pharmacist clicks on the corresponding drug as shown in Figure 5.27.

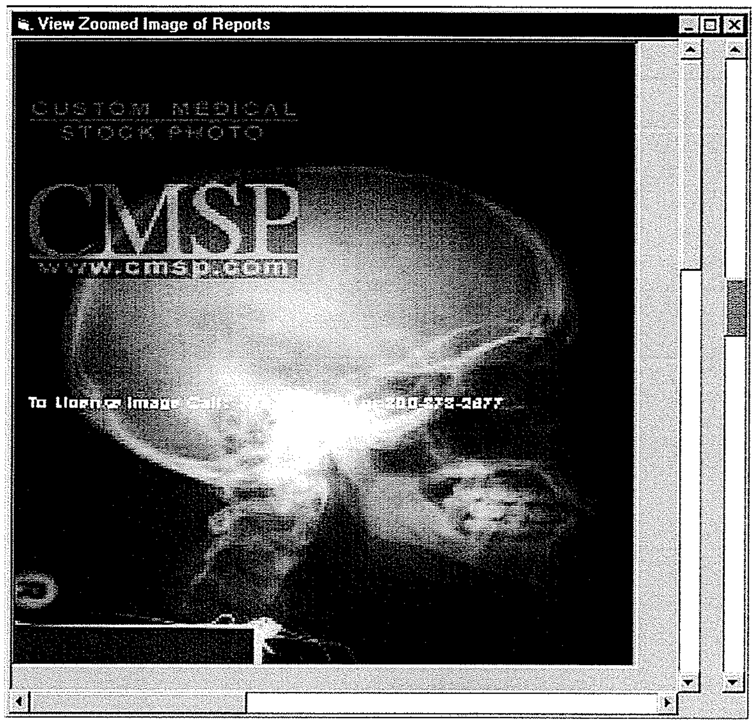
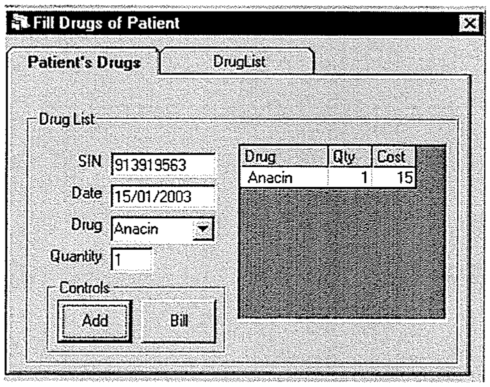


Figure 5.25: After Zoom

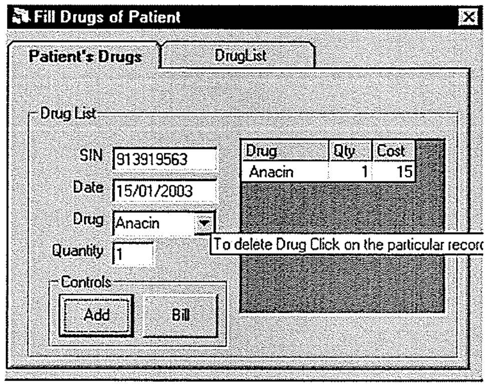


SIN	913919563
Date	15/01/2003
Drug	Anacin
Quantity	1

Drug	Qty	Cost
Anacin	1	15

Figure 5.26: Form showing drug dispense

Pharmacist completes the transaction by clicking on "Bill" on the form shown in Figure 5.26. This forces the application to pop up a window displaying the count and


Fill Drugs of Patient

Patient's Drugs | DrugList

Drug List

SIN 913919563

Date 15/01/2003

Drug Anacin

Quantity 1

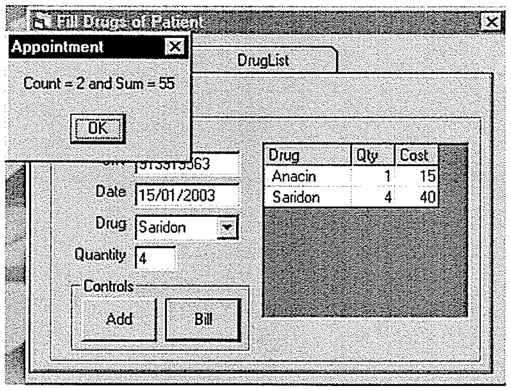
Drug	Qty	Cost
Anacin	1	15

To delete Drug Click on the particular recor

Controls

Figure 5.27: Form showing drug deletion

cost of the drugs sold as shown in Figure 5.28. The final list of drugs purchased is listed on the form shown in Figure 5.29.


Fill Drugs of Patient

Appointment

Count = 2 and Sum = 55

DrugList

SIN 913919563

Date 15/01/2003

Drug Saridon

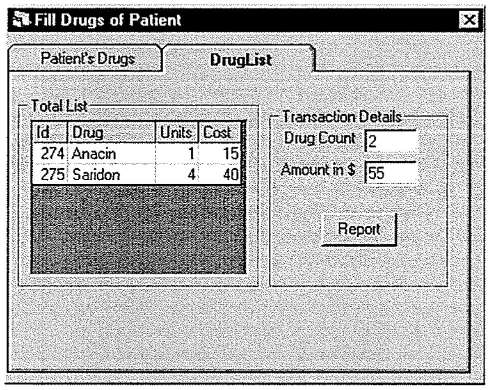
Quantity 4

Drug	Qty	Cost
Anacin	1	15
Saridon	4	40

Controls

Figure 5.28: Form showing the count and cost of drugs

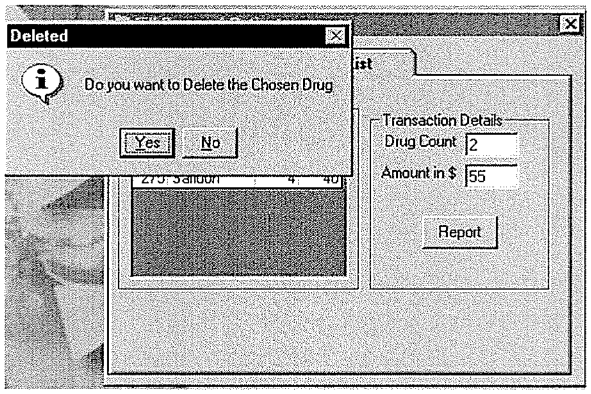
Drugs in the purchased list should have the flexibility to be deleted before the payment is committed. Figure 5.30 captures this process. The application should be interactive to the user instructions. The process of deleting the drug should be acknowledged to the user. Figure 5.31 reflects this. The purchase of drugs should be generated as a bill. This bill is similar to Figure 5.16.



Id	Drug	Units	Cost
274	Anacin	1	15
275	Saridon	4	40

Drug Count 2
Amount in $ 55

Figure 5.29: Form showing purchase list



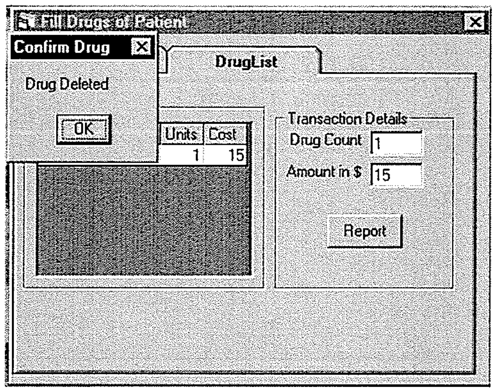
Do you want to Delete the Chosen Drug

Drug Count 2
Amount in $ 55

Figure 5.30: Delete Chosen Drug

In a laboratory, users typically add and update patient records as per the instructions of doctors (i.e., the users add patient records to the laboratory database and update the records after the test results are obtained). Following figures (see Figures 5.32 and 5.33) capture the this scenario.

As with other users, a laboratory user can perform a wild character search to identify patients (e.g., name contains "sam") in the Figure 5.32. This results in another pop up window with all patients names starting "sam". The laboratory user chooses the appropriate patient record and the other details are populated automatically based on the patient selected.

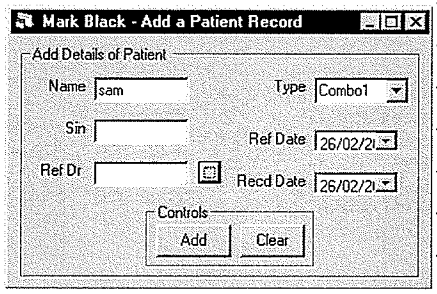


Drug Deleted

Units	Cost
1	15

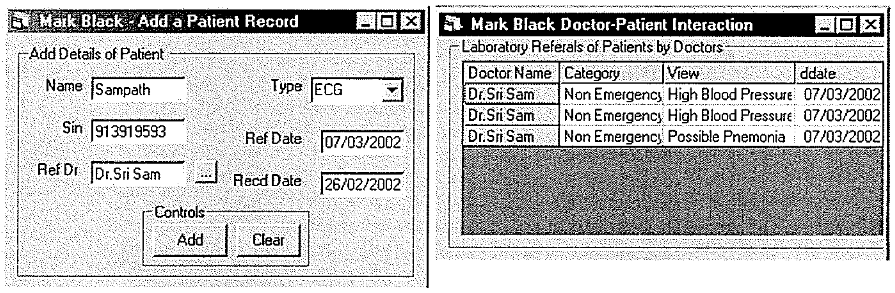
Transaction Details
Drug Count 1
Amount in $ 15

Figure 5.31: Drug Deletion Confirmation



Add Details of Patient
Name sam
Type Combo1
Sin
Ref Date 26/02/2(
Ref Dr
Recd Date 26/02/2(

Figure 5.32: Adding a Patient Record - Laboratory Module


Mark Black - Add a Patient Record
Controls
Add
Clear
Mark Black Doctor-Patient Interaction

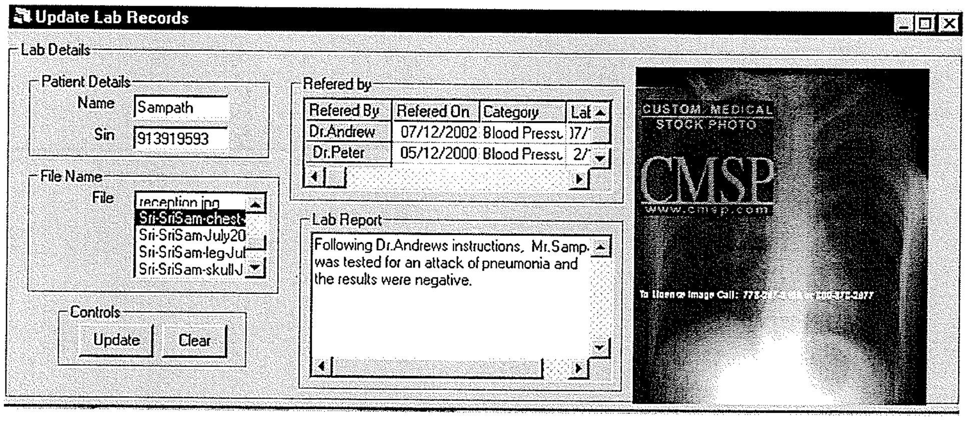
Add Details of Patient
Name Sampath
Type ECG
Sin 913919593
Ref Date 07/03/2002
Ref Dr Dr.Sri Sam
Recd Date 26/02/2002

Laboratory Referals of Patients by Doctors

Doctor Name	Category	View	ddate
Dr.Sri Sam	Non Emergency	High Blood Pressure	07/03/2002
Dr.Sri Sam	Non Emergency	High Blood Pressure	07/03/2002
Dr.Sri Sam	Non Emergency	Possible Pnemonia	07/03/2002

Figure 5.33: Form showing matching patient record - laboratory module

To match a laboratory test with a particular Doctor-Patient encounter, laboratory users choose a particular encounter (various laboratory referrals of patients over a period of time is displayed in another window and the user has the freedom to choose one of the matching interactions).



Name: Sampath
Sin: 913919593

File: reception.jpg
Sri-SriSam-chest-
Sri-SriSam-July20
Sri-SriSam-leg-Jul
Sri-SriSam-skullJ

Refered By	Refered On	Category	Lal
Dr.Andrew	07/12/2002	Blood Pressu	)7/'
Dr.Peter	05/12/2000	Blood Pressu	2/'

Following Dr.Andrews instructions, Mr.Samp. was tested for an attack of pneumonia and the results were negative.

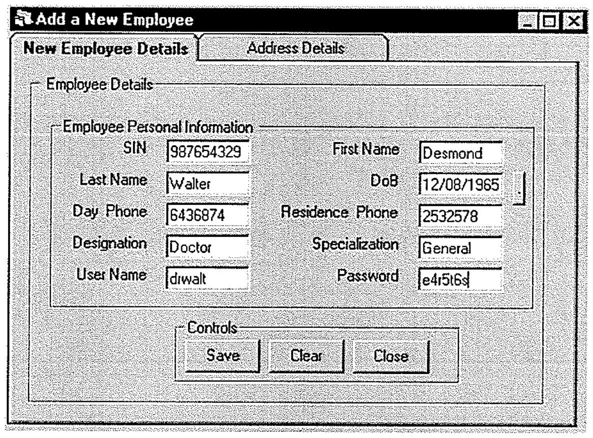
Figure 5.34: Associating a record with the Corresponding Doctor-Patient Encounter

After matching the record, the laboratory user updates the patient record by assigning the matching file name referring to the test results (e.g., X-Ray images,ECG reports, etc.). The user gives his/her view on the report.

Finally, the administrators are responsible for maintaining the database and managing the resources (adding and deleting as needed) in the system. The following forms are displayed when a user is authenticated as a system administrator (i.e., the user has a prefix "ad" on his/her user name). The following forms (Figure 5.35 and Figure 5.36) illustrate the process of adding new users to the system by the administrators.

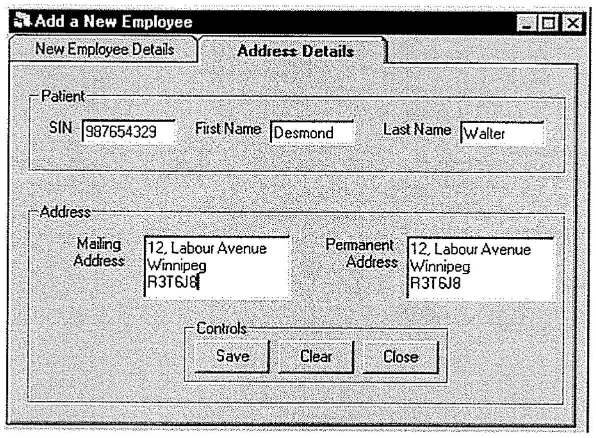
Figure 5.37 shows a form that allows administrators to delete other users from the system. Similarly, other resources (e.g., laboratory machines, etc) are also added to the system. An administrator updates the information on addition of such resources.

In addition to adding and deleting employees, administrators also configure certian key attributes required by the application (e.g., the organization name, tax rates and



SIN	987654329	First Name	Desmond
Last Name	Walter	DoB	12/08/1965
Day Phone	6436874	Residence Phone	2532578
Designation	Doctor	Specialization	General
User Name	drwalt	Password	e4r5t6s

Figure 5.35: Form showing adding new employees



SIN 987654329 First Name Desmond Last Name Walter

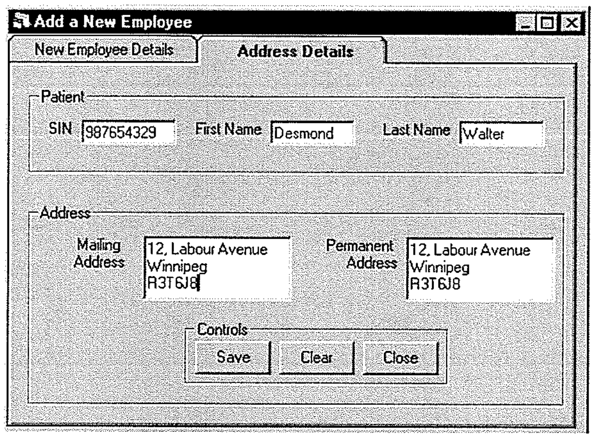
Mailing Address: 12, Labour Avenue Winnipeg R3T6J8

Permanent Address: 12, Labour Avenue Winnipeg R3T6J8

Figure 5.36: Form showing updating address details of employees

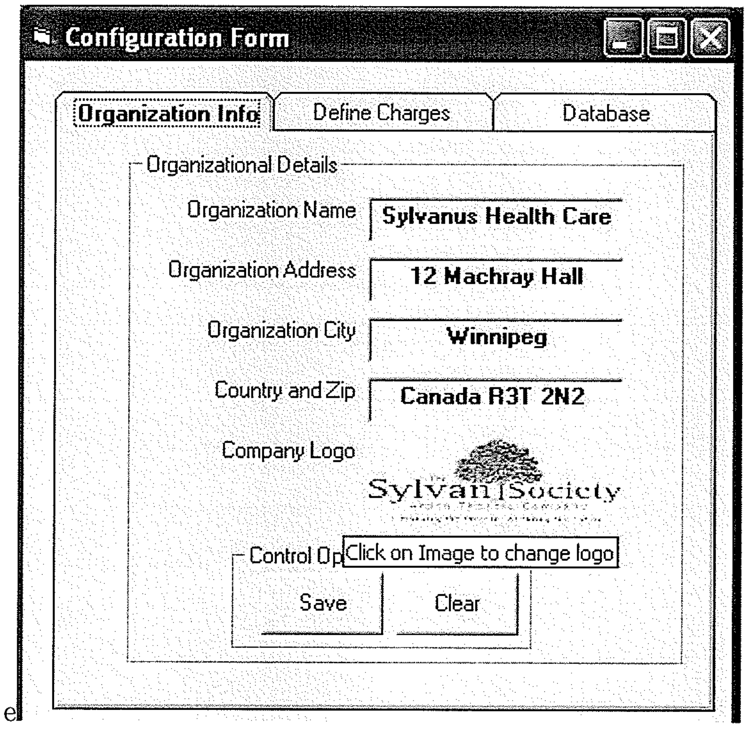
the database this application uses. Figures (5.38 - 5.41) shows the configuration of such fields.

Figure 5.38 shows a screen that allows administrators to change the organization's name.
Figure 5.39 shows a screen that allows, system administrators to update the cost for using any of the services in the health care system. Figure 5.40 shows the form the system admnistrators use to change the ODBC/Database used by the health care



SIN	987654329
First Name	Desmond
Last Name	Walter
Mailing Address	12, Labour Avenue Winnipeg R3T6J8
Permanent Address	12, Labour Avenue Winnipeg R3T6J8

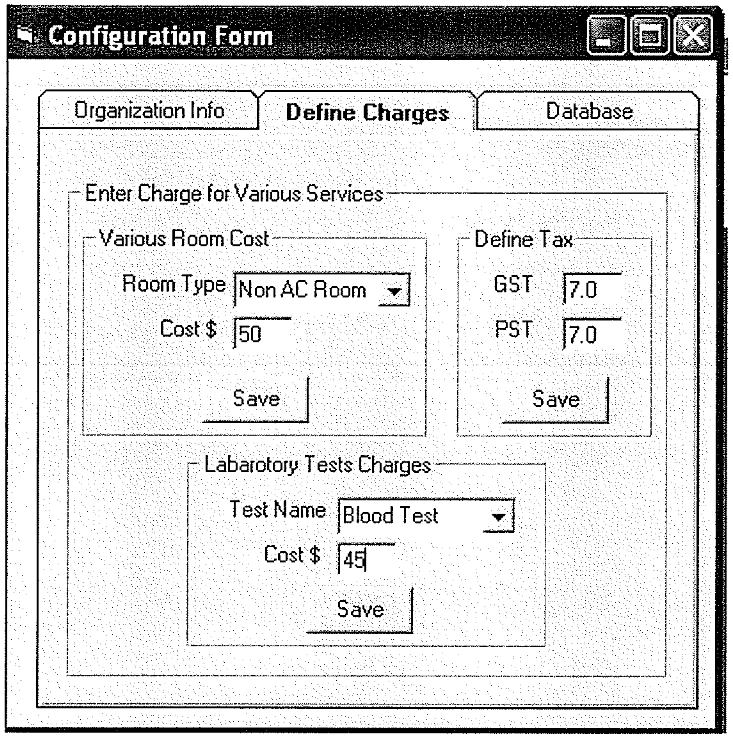
Figure 5.37: Form displaying deleting employees profiles



Organization Name	Sylvanus Health Care
Organization Address	12 Machray Hall
Organization City	Winnipeg
Country and Zip	Canada R3T 2N2

Figure 5.38: Setting the Organization Name and Logo

application.



Enter Charge for Various Services

Various Room Cost
Room Type: Non AC Room
Cost $: 50

Define Tax
GST: 7.0
PST: 7.0

Labarotory Tests Charges
Test Name: Blood Test
Cost $: 45

Figure 5.39: Form showing setting of cost for using resources

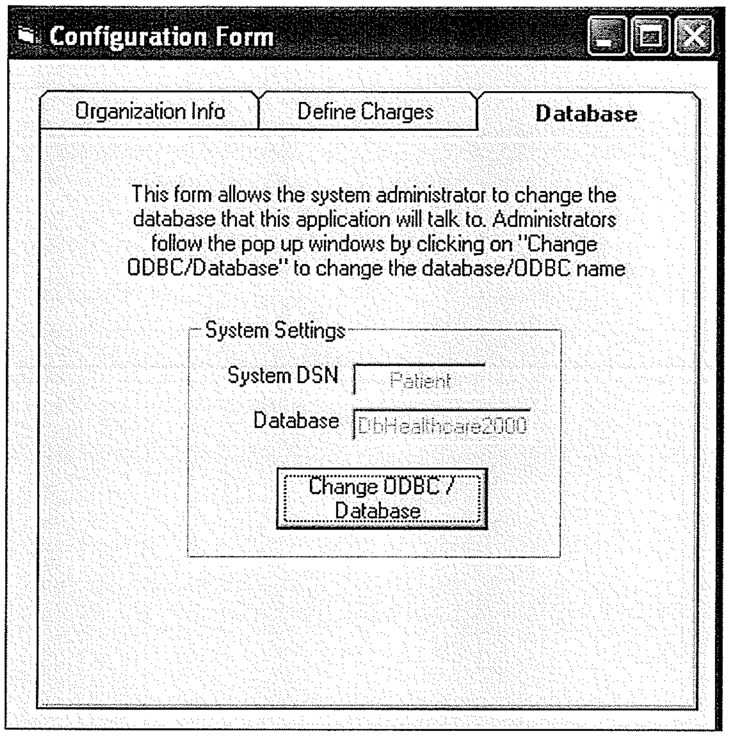
On clicking the " Change ODBC/Database", the apllication opens the Windows Data Source Administrator for changing the ODBC as shown in Figure 5.41.

Users can also generate various reports[1] *and* [2] pertaining to patient's visit, resource usage and doctor's appointments on a particular day or week. Following figures (Figure 5.42 - 5.52) shows some of the reports developed.

Invoices, as described earlier, are generated for the use of health care resources (e.g. rooms and laboratory services). Forms (5.47 and 5.48) show such invoices.

[1]The logo in the header of the report is copyright of The Arden Theatre Company image available at http://www.ardentheatre.org

[2]The logo on the body of the report is copyright of University of Missouri available at www.system.missouri.edu/vpacad/ images/sigill.gif



This form allows the system administrator to change the database that this application will talk to. Administrators follow the pop up windows by clicking on "Change ODBC/Database" to change the database/ODBC name

System DSN: Patient
Database: DbHealthcare2000

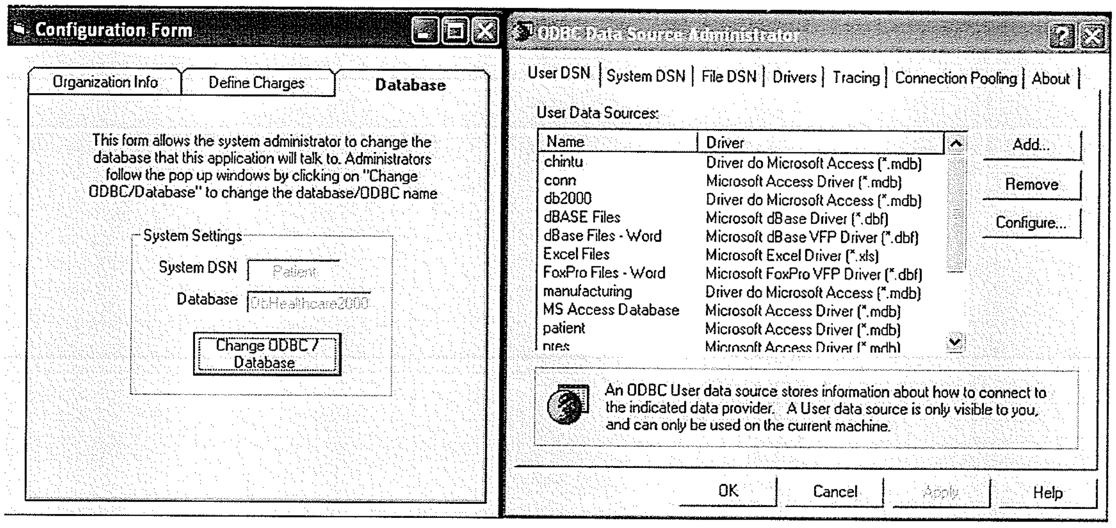
Figure 5.40: Setting the ODBC Database used by the health care application

Reports of drugs ordered in the pharmacy are also generated. Figures 5.49 and 5.50 show reports of drug orders. Reports of doctors interaction with patients have to be generated as well. Figures 5.51 and 5.52 show the interaction reports.

A consolidated bill on the patient's visit is also to be developed as shown in Figure 5.53.

5.4 Deployment and Quality Assurance Strategies

Quality Assurance (QA) is the validation and verification of a system performed by end users rather than by the software developers or the software testers. A number of attributes determine the quality of a software system [So92], however not all of the criteria are applicable to every software developed [BP84]. Some QA attributes,



This form allows the system administrator to change the database that this application will talk to. Administrators follow the pop up windows by clicking on "Change ODBC/Database" to change the database/ODBC name

System Settings

System DSN	Patient
Database	DbHealthcare2000

Change ODBC / Database

User Data Sources:

Name	Driver
chintu	Driver do Microsoft Access (*.mdb)
conn	Microsoft Access Driver (*.mdb)
db2000	Driver do Microsoft Access (*.mdb)
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
manufacturing	Driver do Microsoft Access (*.mdb)
MS Access Database	Microsoft Access Driver (*.mdb)
patient	Microsoft Access Driver (*.mdb)
nres	Microsoft Access Driver (*.mdb)

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

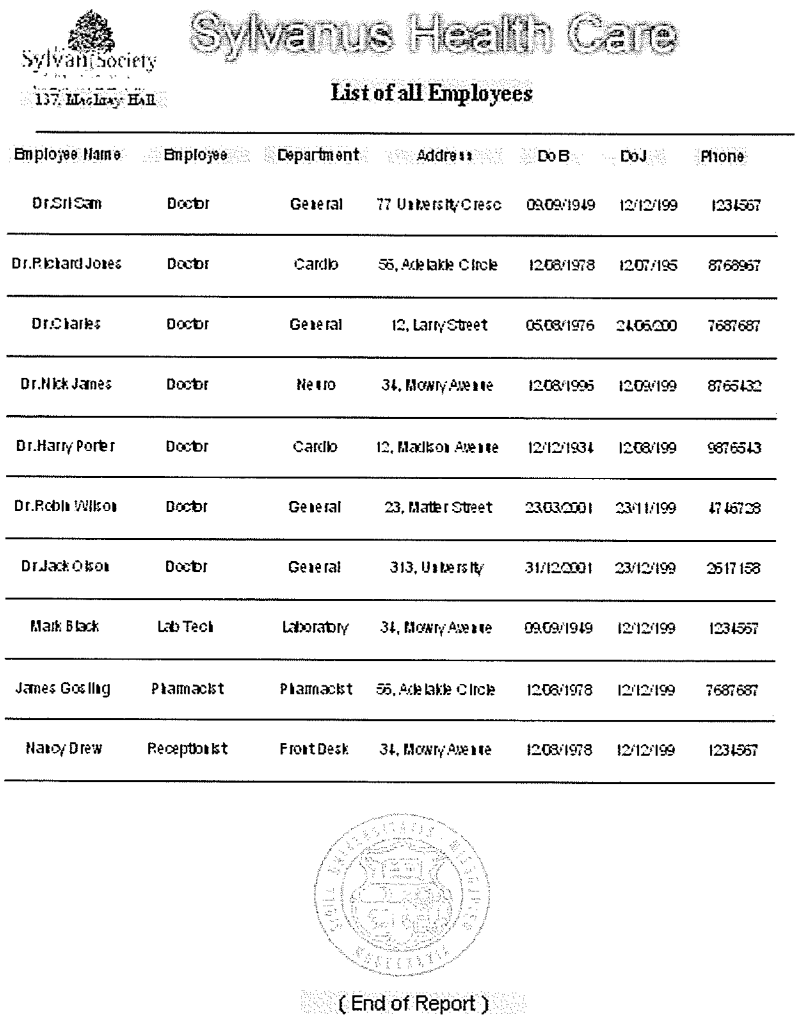
Figure 5.41: Windows Data Source Administrator Window

which relates to this application are explained in the following paragraph.

The application in this thesis is windows-based, hence little or no effort will be required to make the application run on other windows-based system, thus supporting portability. For a good understanding of the system, the design, implementation, and the testing of the system are documented. The application itself includes help files and commented source codes to aid the understanding of the system. Various users access the application for various purposes. The faults for a particular user profile do not halt the proper working of the system (offering reliability). The transactions in the implementation are based on the formal specifications, supporting correctness of the software.

Software quality assurance guarantees the quality of the development process and products. A well defined software development process yields high quality software [So92]. Hence, a great deal of time and effort was devoted in formulating a well-planned software development process.

A constant review of the software developed helped in examining parts and all of



Sylvan Society
137, Macleay Hall

Sylvanus Health Care

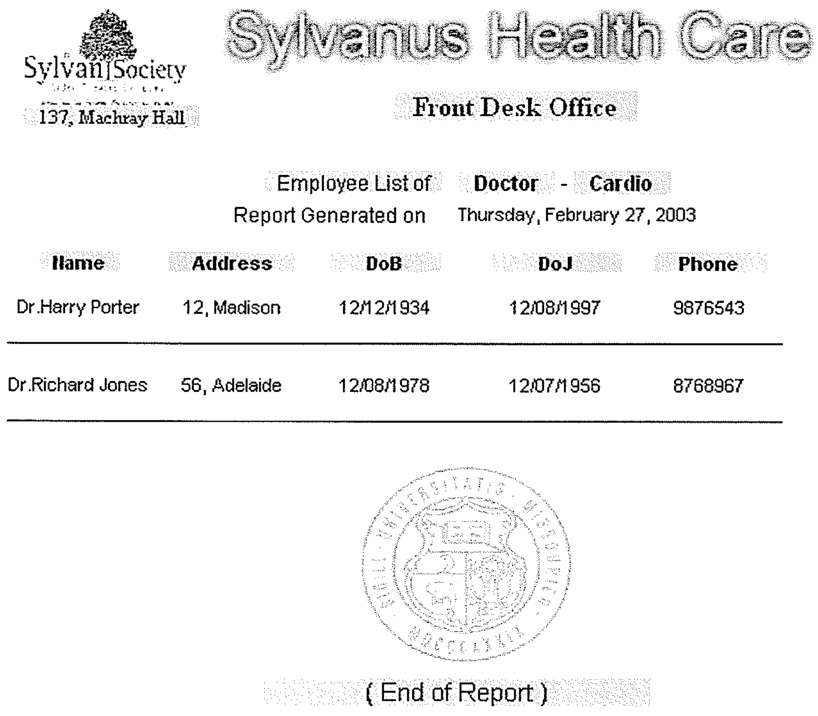
List of all Employees

Employee Name	Employee	Department	Address	DoB	DoJ	Phone
Dr.Sri Sam	Doctor	General	77 University Cresc	09/09/1949	12/12/199	1234567
Dr.Richard Jones	Doctor	Cardio	56, Adelaide Circle	12/08/1978	12/07/195	8768967
Dr.Charles	Doctor	General	12, Larry Street	05/08/1976	24/06/200	7687687
Dr.Nick James	Doctor	Neuro	34, Mowry Avenue	12/08/1996	12/09/199	8765432
Dr.Harry Porter	Doctor	Cardio	12, Madison Avenue	12/12/1934	12/08/199	9876543
Dr.Robin Wilson	Doctor	General	23, Matter Street	23/03/2001	23/11/199	4746728
Dr.Jack Olson	Doctor	General	313, University	31/12/2001	23/12/199	2517158
Mark Black	Lab Tech	Laboratory	34, Mowry Avenue	09/09/1949	12/12/199	1234567
James Gosling	Pharmacist	Pharmacist	56, Adelaide Circle	12/08/1978	12/12/199	7687687
Nancy Drew	Receptionist	Front Desk	34, Mowry Avenue	12/08/1978	12/12/199	1234567

(End of Report)

Figure 5.42: Report showing the list of all employees working in Sylvanus Health Care Center

the system. Notes on the review were recorded and the problems identified were corrected. A thorough review of earlier system designs gave better insight into the various problems encountered in the development of the system and how to resolve them. Verification and validation of a software accompanies the software accompanies product through out its implementation. Verification and validation involves an analysis of the product using both static and dynamic approaches. Formal specification



Sylvanus Health Care

Front Desk Office

Employee List of Doctor - Cardio
Report Generated on Thursday, February 27, 2003

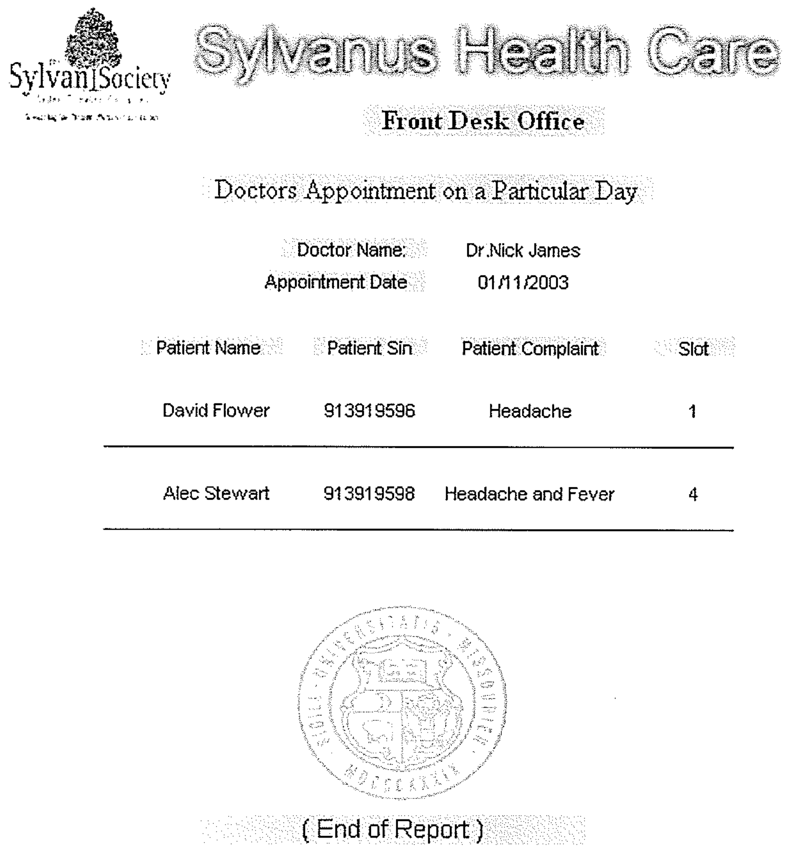
Name	Address	DoB	DoJ	Phone
Dr.Harry Porter	12, Madison	12/12/1934	12/08/1997	9876543
Dr.Richard Jones	56, Adelaide	12/08/1978	12/07/1956	8768967

(End of Report)

Figure 5.43: Report showing doctors belonging to a particular specialization

and verification was done guaranteeing the mapping between the implementation and the specification (the formal specification itself was guaranteed freed of types and logical errors using the Z-Eves tool) and dynamic verification was done using functional testing (*black-box testing*).

Testing any application usually demonstrates the existence of errors. Defects often exists in complex systems even after detailed testing. A successful defect test displays the presence of defects in the software under consideration. Testing the application involved performing unit and module tests using typical test data and testing the code during the implementation of individual components of the software system. Unit testing confirms that the tested units adhered to the purpose of their implementation. After implementing various modules (e.g., doctor modules, front desk modules,


Sylvan Society

Sylvanus Health Care

Front Desk Office

Doctors Appointment on a Particular Day

Doctor Name:	Dr.Nick James
Appointment Date	01/11/2003

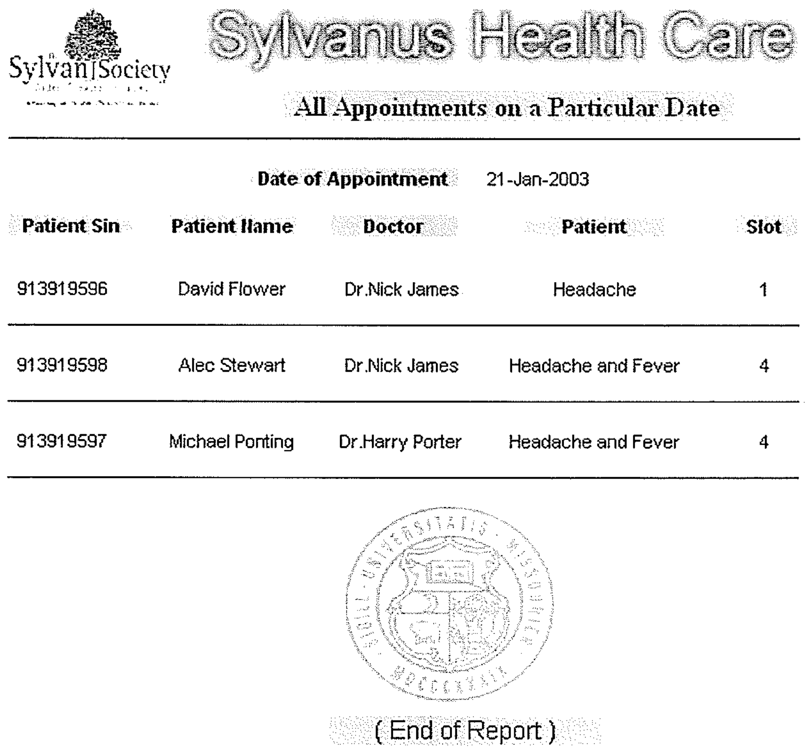
Patient Name	Patient Sin	Patient Complaint	Slot
David Flower	913919596	Headache	1
Alec Stewart	913919598	Headache and Fever	4

(End of Report)

Figure 5.44: Report showing a doctor's appointments for a particular day

etc), the modules were integrated. Sub-system and system testing were performed on the integrated modules to expose interface conflicts and validate the system, respectively. The detected defects were corrected and the above tests repeated (*regression testing*) to account for the modifications.

Since, the application is enterprise-based and will be used across a network, stress tests of the software with increasing load was performed. Stress testing showed that overloading the system does not cause any anomaliesin the software. Poor usability may result from badly designed user interface. With mulitple navigational windows and various number of clicks (Click-Depth) the user could be lost during the naviga-


Sylvan Society

Sylvanus Health Care

All Appointments on a Particular Date

Date of Appointment 21-Jan-2003

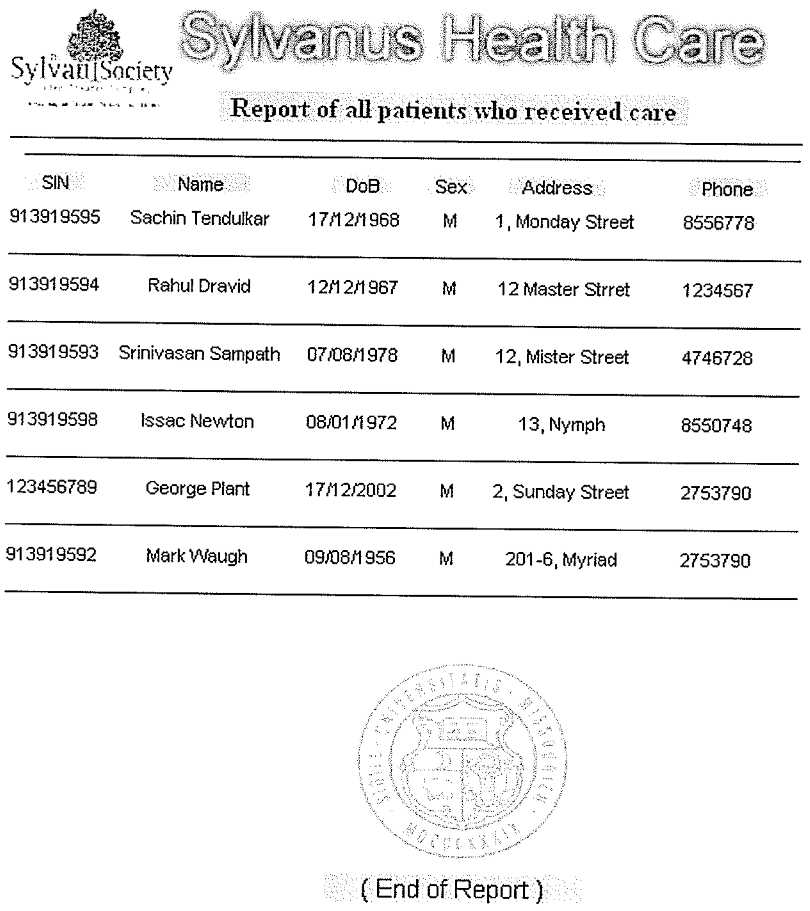
Patient Sin	Patient Name	Doctor	Patient	Slot
913919596	David Flower	Dr.Nick James	Headache	1
913919598	Alec Stewart	Dr.Nick James	Headache and Fever	4
913919597	Michael Ponting	Dr.Harry Porter	Headache and Fever	4

(End of Report)

Figure 5.45: Report showing all appointments for a particular day

tion. Usability results, often reveal poor designing problems (in terms of layout and structure).

The deployment of the application to the end user is the ultimate aim in developing software systems. Hence, adoption of a sound deployment strategy is a prerequisite for the ultimate success of the software. The deployment strategy for this project was based on several phases as described in [Sc99]. For example, in the inception phase of the software, initial deployment strategies were planned for with a focus on identifying the end users and releasing the software at once rather than progressively.



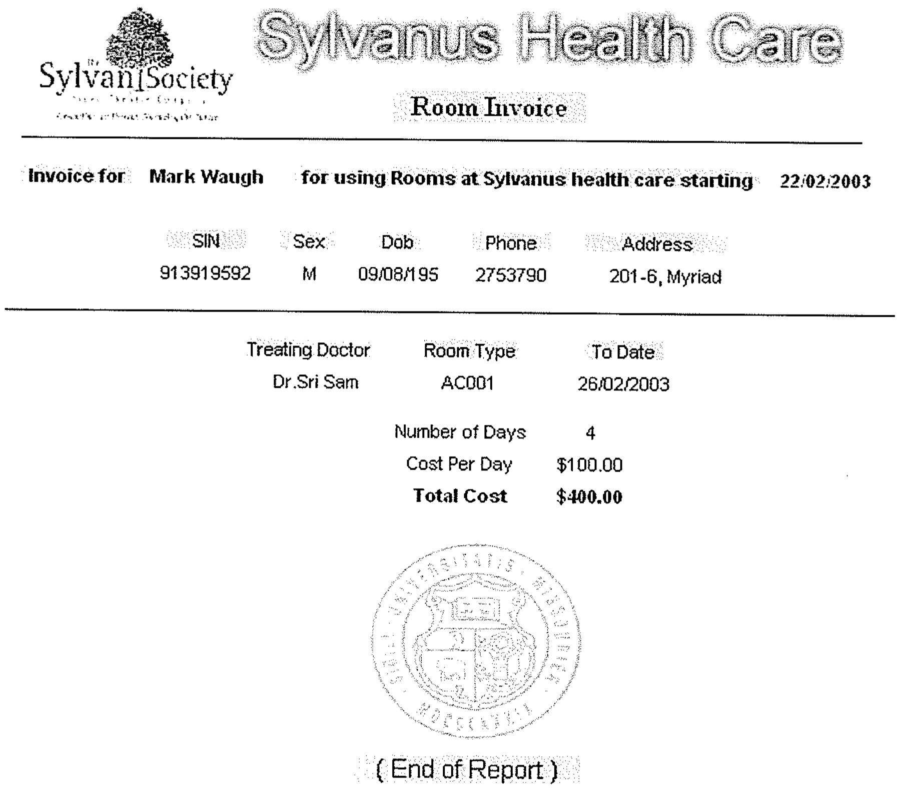
Sylvanus Health Care

Report of all patients who received care

SIN	Name	DoB	Sex	Address	Phone
913919595	Sachin Tendulkar	17/12/1968	M	1, Monday Street	8556778
913919594	Rahul Dravid	12/12/1967	M	12 Master Strret	1234567
913919593	Srinivasan Sampath	07/08/1978	M	12, Mister Street	4746728
913919598	Issac Newton	08/01/1972	M	13, Nymph	8550748
123456789	George Plant	17/12/2002	M	2, Sunday Street	2753790
913919592	Mark Waugh	09/08/1956	M	201-6, Myriad	2753790

(End of Report)

Figure 5.46: Report showing the all patient who have received care at Sylvanus Health care center



Sylvan Society

Sylvanus Health Care

Room Invoice

Invoice for Mark Waugh for using Rooms at Sylvanus health care starting 22/02/2003

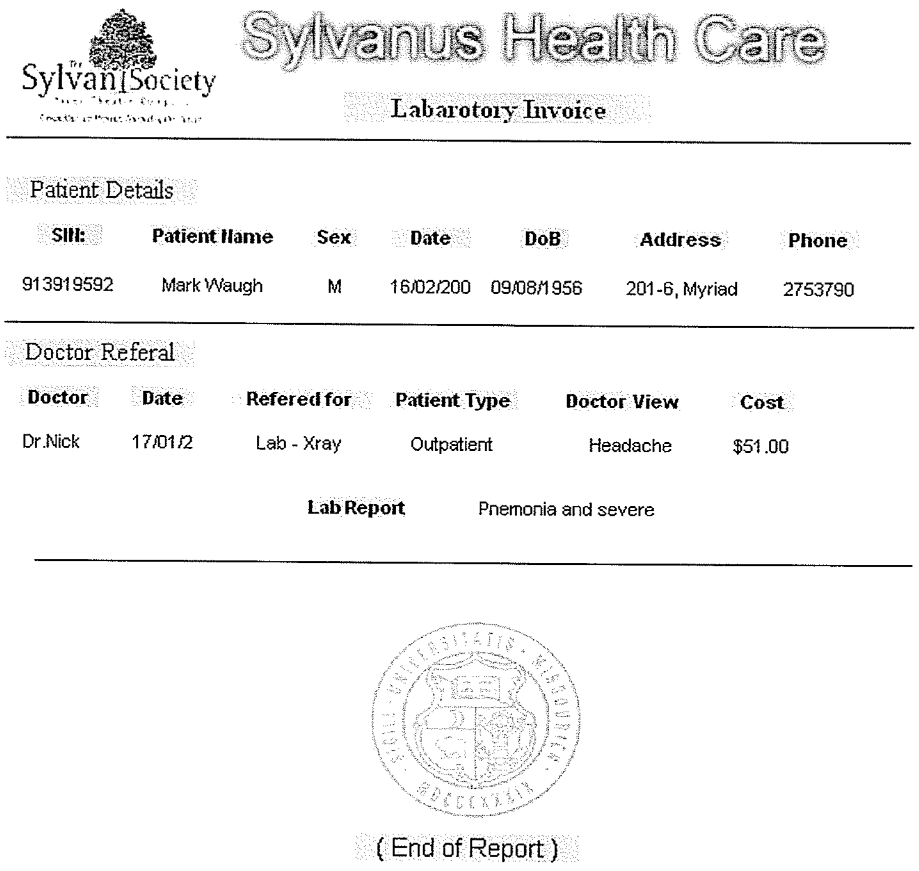
SIN	Sex	Dob	Phone	Address
913919592	M	09/08/195	2753790	201-6, Myriad

Treating Doctor	Room Type	To Date
Dr.Sri Sam	AC001	26/02/2003

Number of Days	4
Cost Per Day	$100.00
Total Cost	**$400.00**

(End of Report)

Figure 5.47: Report showing the room renting charges



Sylvanus Health Care

Labarotory Invoice

Patient Details

SIN:	Patient Name	Sex	Date	DoB	Address	Phone
913919592	Mark Waugh	M	16/02/200	09/08/1956	201-6, Myriad	2753790

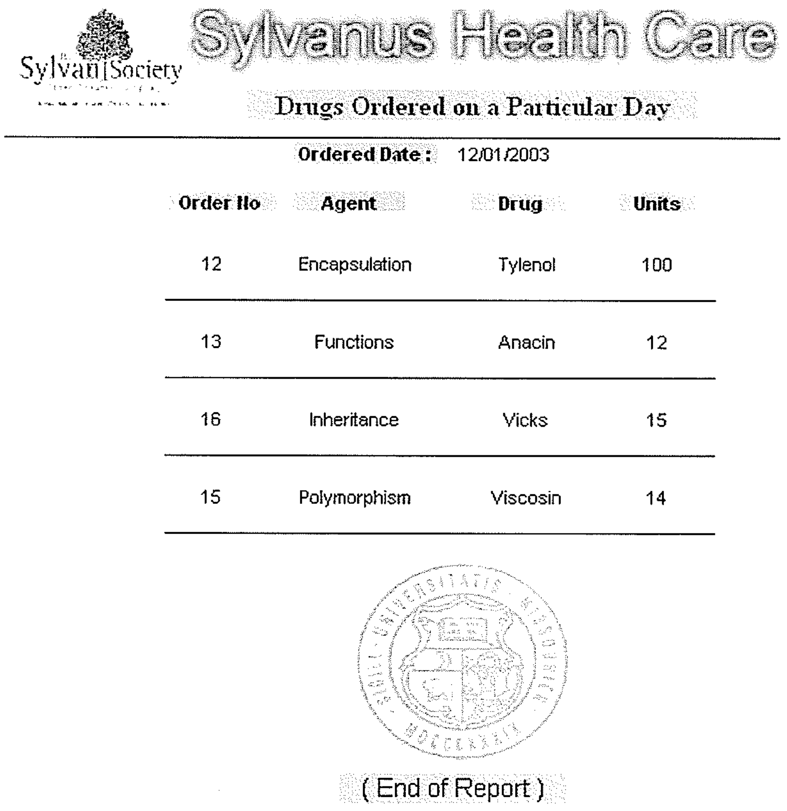
Doctor Referal

Doctor	Date	Refered for	Patient Type	Doctor View	Cost
Dr.Nick	17/01/2	Lab - Xray	Outpatient	Headache	$51.00

Lab Report Pnemonia and severe

(End of Report)

Figure 5.48: Report showing the laboratory resources invoice



Sylvan Society

Sylvanus Health Care

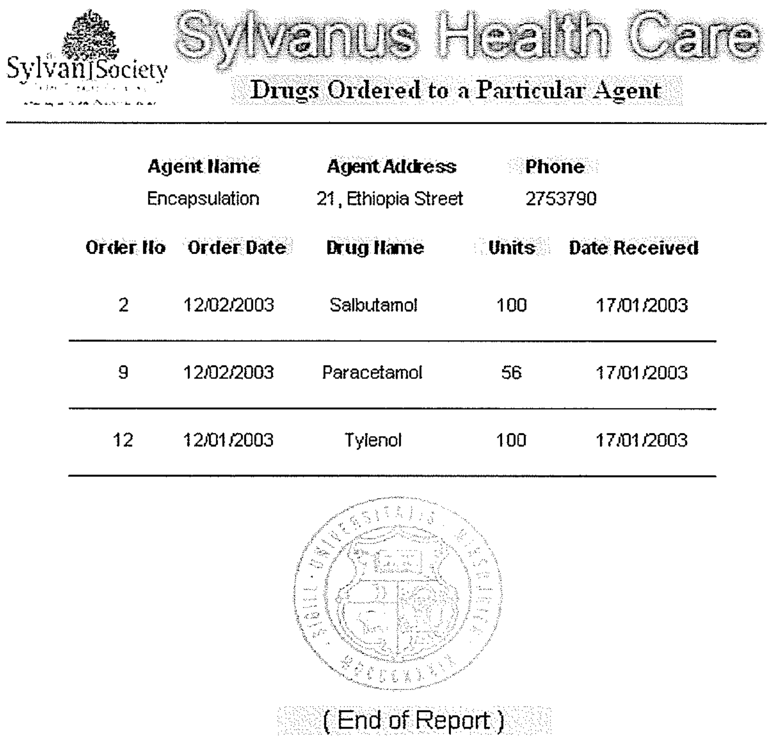
Drugs Ordered on a Particular Day

Ordered Date : 12/01/2003

Order No	Agent	Drug	Units
12	Encapsulation	Tylenol	100
13	Functions	Anacin	12
16	Inheritance	Vicks	15
15	Polymorphism	Viscosin	14

(End of Report)

Figure 5.49: Report showing the drug orders on a day with various drug suppliers


Sylvan Society

Sylvanus Health Care

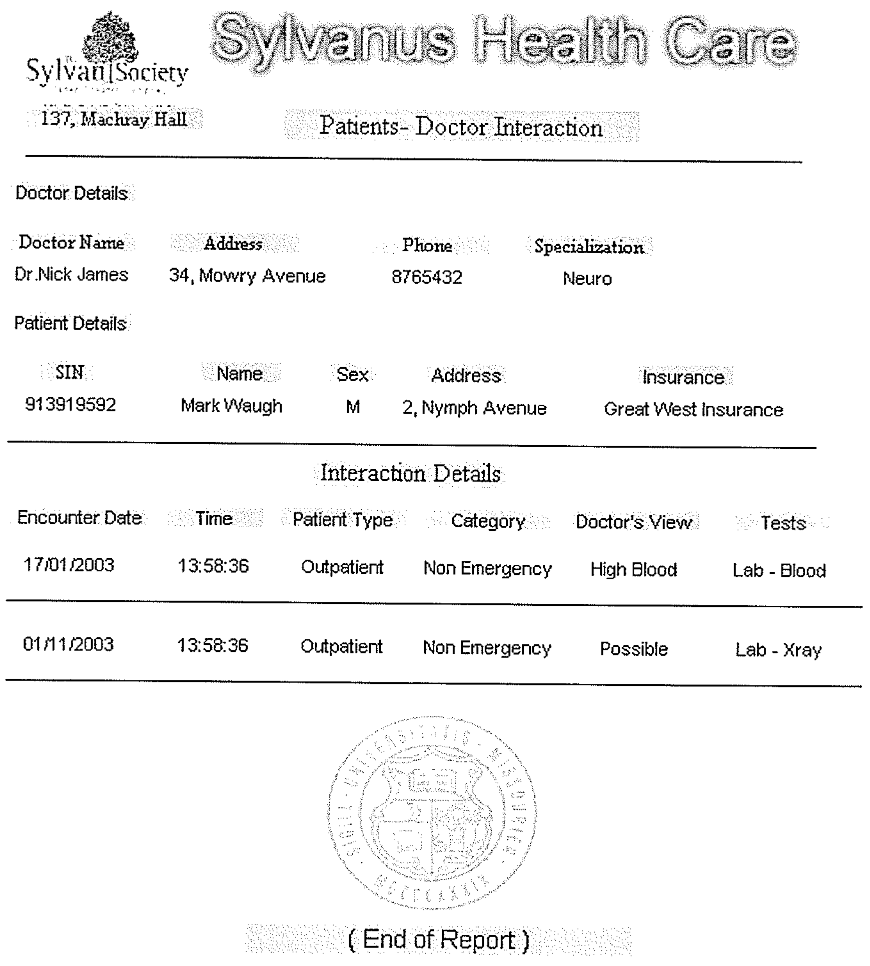
Drugs Ordered to a Particular Agent

Agent Name	Agent Address	Phone
Encapsulation	21, Ethiopia Street	2753790

Order No	Order Date	Drug Name	Units	Date Received
2	12/02/2003	Salbutamol	100	17/01/2003
9	12/02/2003	Paracetamol	56	17/01/2003
12	12/01/2003	Tylenol	100	17/01/2003

(End of Report)

Figure 5.50: Report showing the drug ordered with a particular supplier



Sylvan Society

Sylvanus Health Care

137, Machray Hall

Patients- Doctor Interaction

Doctor Details

Doctor Name	Address	Phone	Specialization
Dr.Nick James	34, Mowry Avenue	8765432	Neuro

Patient Details

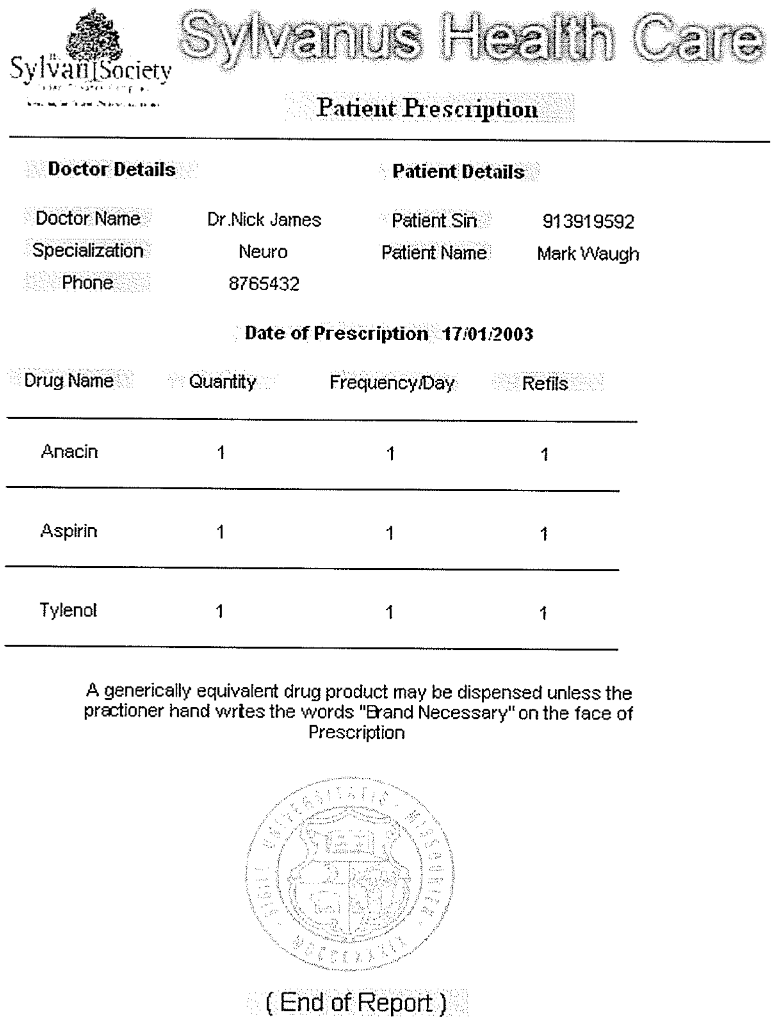
SIN	Name	Sex	Address	Insurance
913919592	Mark Waugh	M	2, Nymph Avenue	Great West Insurance

Interaction Details

Encounter Date	Time	Patient Type	Category	Doctor's View	Tests
17/01/2003	13:58:36	Outpatient	Non Emergency	High Blood	Lab - Blood
01/11/2003	13:58:36	Outpatient	Non Emergency	Possible	Lab - Xray

(End of Report)

Figure 5.51: Report showing the doctor patient interaction


Sylvan Society

Sylvanus Health Care

Patient Prescription

Doctor Details		Patient Details	
Doctor Name	Dr.Nick James	Patient Sin	913919592
Specialization	Neuro	Patient Name	Mark Waugh
Phone	8765432		

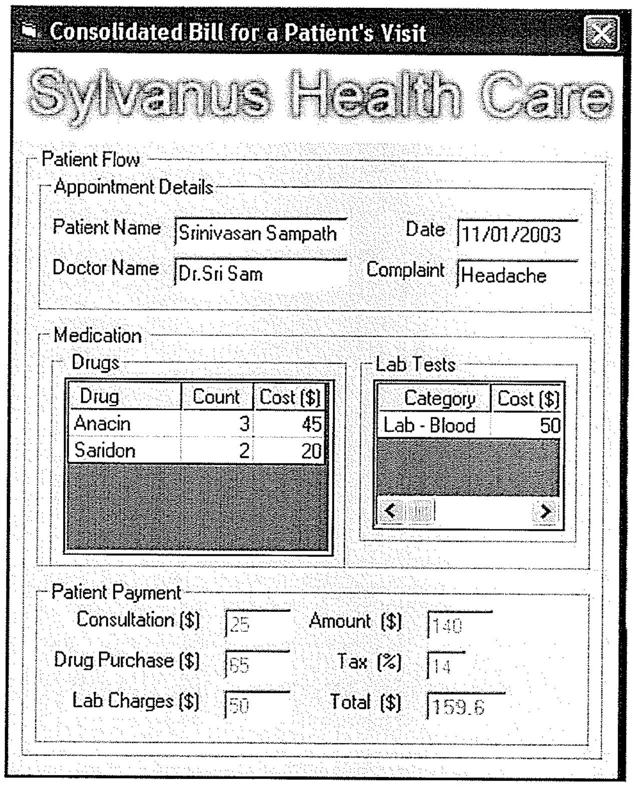
Date of Prescription 17/01/2003

Drug Name	Quantity	Frequency/Day	Refils
Anacin	1	1	1
Aspirin	1	1	1
Tylenol	1	1	1

A generically equivalent drug product may be dispensed unless the practioner hand writes the words "Brand Necessary" on the face of Prescription

(End of Report)

Figure 5.52: Report showing the medicines prescribed for a patient



Patient Name: Srinivasan Sampath
Date: 11/01/2003
Doctor Name: Dr.Sri Sam
Complaint: Headache

Drug	Count	Cost ($)
Anacin	3	45
Saridon	2	20

Category	Cost ($)
Lab - Blood	50

Consultation ($)	25	Amount ($)	140
Drug Purchase ($)	65	Tax (%)	14
Lab Charges ($)	50	Total ($)	159.6

Figure 5.53: Form showing a consolidated bill for a particular visit

Chapter 6

Conclusions

" I find that the harder I work, the more luck I seem to have." - Thomas Jefferson (1743-1826)

This chapter summarizes the contributions of the thesis and explores some areas of the potential future work. This thesis provided a formal specification of transaction processing for a distributed, integrated health care system with focus on electronic patient record.

6.1 Summary of Contributions

The thesis contributes to research of application of computers in the health care domain in the following ways:

1. It formalizes the transactions of a patient with the medical community and formally models the design of those aspects of a health care system that impacts electronic medical record.

2. A formal specification of the health care system was developed and a description of the adopted integrated health care system architecture, including the different object models developed using UML was presented.

3. A system was implemented to reflect parts of formal specifications.

4. This thesis provides a good case study for learning and transferring skills in formal software design.

6.2 Future Work

The implementation of certain functionality of the system was not attempted and was left for future work. Few of them include:

1. Various transactions that are specified need to be refined and customized to tailor the needs of a particular health care provider. The thesis presents a generic model for capturing information and processing the transactions based on the inputs. For real-world applications,which are based on formal specifications, these inputs have to be studied in great details and be customized before they can be implemented.

2. Encryption of data using a uniform technique could be added to enhance the security.

3. A health care informatics domain/application is not just patient centric. There exists several other transactions that the patient use indirectly, hence such transactions have to be recognized, studied and formally modelled before the implementation to make the application complete.

4. Even though agent based system was not the focus of this thesis, the abundant literature and the growing adoption of agent based systems in the health care domain prompted us to mention about agents. The use of agents in this domain is a potential research topic. Since, the medical records can be populated from various database(s) of participating health care organizations, the use of agent based systems is a likely candidate.

To conclude, this thesis offers an excellent platform for studying and specifying the overall transactions in a health care domain.

References

[BH95] J. P. Bowen and M. G. Hinchey, "Seven More Myths of Formal Methods," *IEEE Software*, Vol. 12, No. 3, 1995, pp. 34–41.

[BP84] F. J. Buckley and R. Poston, "Software Quality Assurance," *IEEE Transactions on Software Engineering*, Vol. 10, No. 1, January 1984, pp. 36–41.

[Ba92] R. Barden, S. Stepney, and D. Cooper, "Improving Software Tests using Z Specifications," *Proceedings of the Sixth Annual Z User Workshop* J. E. Nicholls, ed., Workshops in Computing , Springer-Verlag, December 1992, pp. 99–124.

[Bi95] K. Binsted, A. Cawsey, and R. Jones, "Generating Personalised Patient Information Using the Medical Record," *Proceedings of the Fifth Conference on AI in Medicine Europe,* P. Barahona, M. Stefanelli, and J. Wyatt, ed., Vol. 934 of *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1995, pp. 29–41.

[Da96] J. Davies and J. C. P. Woodcock, *Using Z: Specification, Refinement, and Proof*, Prentice Hall, London, July 1996.

[De92] K. Decker and V. Lesser, "Generalizing the Partial Global Planning Algorithm," *International Journal of Cooperative Information Systems*, Vol. 1, No. 2, June 1992, pp. 319–346.

[De98] K. Decker and J. Li, "Coordinated Hospital Scheduling," *Proceedings of the Third International Conference on Multi-Agent Systems*, (Paris, France), 1998, pp. 104–111.

[Di95] C. DiMarco, G. Hirst, L. Wanner, and J. Wilkinson, "Healthdoc: Customizing Patient Information and Health Education by Medical Condition and Personal Characteristics," *Proceedings of the First International Workshop on Artificial Intelligence in Patient Education* , (Glasgow, UK), 1995.

[Di97] R. S. Dick, E. B. Steen and D. E. Detmer, "The Computer-Based Patient Record: An Essential Technology for Healthcare," *Institute of Medicine*, (Washington D.C, USA), 1997.

[Dy98] N. Dykman, *SAGE: Generating Applications with UML and Components*, Master's Thesis, Department of Computer Science, The University of Utah, Utah, USA, December 1998.

[EN94] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, Second Edition, Benjamin Cummings, California, 1994.

[Eh00] S. A. Ehikioya, "A Formal Design Framework for Electronic Commerce Transactions," *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR-2000)*, (L'Aquila, Italy), July 31 - Aug 6 2000.

[Eh01] S. A. Ehikioya, "A Formal Characterization of Electronic Commerce Transactions," *International Journal of Computer and Information Science*, Vol. 2, No. 3, September 2001.

[Eh02] S. A. Ehikioya and A. Mitra, "A Formal Design of Electronic Patient Medical Record Exchange System," *The Benin Journal of Advances in Computer Science*, 2002.

[Eh97] S. A. Ehikioya, *Specification of Transaction Systems Protocols*, Ph.D Thesis, University of Manitoba, Winnipeg, Canada, 1997.

[Eh99] S. A. Ehikioya and K. E. Barker, "Towards a Formal Specification Methodology for Transaction Systems Protocols," *Proceedings of the Third Annual IASTED International Conference on Software Engineering and Applications* , (Scottsdale, USA), October 1999.

[Eh99a] S. A. Ehikioya, "An Agent-based System for Distributed Transactions: A Model for Internet-based Transactions," *IEEE Canadian Conference on Electrical and Computer Engineering*, (Edmonton, Canada), 1999.

[Ev98] A. Evans, R. France, K. Lano, and B. Rumpe, "Developing the UML as a Formal Modelling Notation," *UML 1998 Beyond the notation. International Workshop*, Vol. 1618, (Mulhouse, France), June 1998, pp. 297–307.

[Ev99] J. A. Evans, "Electronic Medical Record System," US Patent Number 5924074, Application Number 721182, July 1999.

[Ga91] A. M. Garcia, "System and Method for Scheduling and Reporting Patient Related Services include Prioritizing Services," US Patent Number 5065315, Application Number 426113, November 1991.

[Go00] C. Goble and P. Crowther, "Schemas for Telling Stories in Medical Records," *Proceedings of the Fourth International Conference on Extending Database Technology* M. Jarke, J. Bubenko, and K. Jeffery, ed., Lecture Notes in Computer Science, (Cambridge, U.K), Springer-Verlag, September 1994, pp. 393–406.

[Ha94] W. E. Hammond, "Hospital Informations Systems: A Review in Perspective," *Year Book of Medical Informatics* J. H.van Bemmel and A. T. McCray, ed., Advanced Communications in Health Care, (Stuttgart, Germany), International Medical Informatics Association, Schattauer, 1994, pp. 95–102.

[Ha99] M. Hannebauer, H. D. Burkhard, J. Wendler, and U. Geske, "Composable Agents for Patient Flow Control — Preliminary Concepts," *Proceedings of the DFG-SPP Workshop*, (Ilmenau, Germany), 1999, pp. 223–231.

[He97] S. Helke, T. Neustupny and T. Santen, "Automating Test Case Generation from Z Specifications with Isabelle," *Z User Meeting 1997 (ZUM'97): The Z Formal Specification Notation* J.P. Bowen and M.G. Hinchey and D. Till, ed., Vol. 1212 of *Lecture Notes in Computer Science*, Springer-Verlag, 1997, pp. 52–71.

[Ho95] H. M. Hörcher, "Improving Software Tests using Z Specifications," *Proceedings of the Nineth Annual Z User Workshop* J. P. Bowen and M. G. Hinchey, ed., Vol. 967 of *Lecture Notes in Computer Science*, (Limerick, Ireland), Springer-Verlag, September 1995, pp. 152–166.

[Ho99] C. Hofmeister, R. L. Nord, and D. Soni, "Describing Software Architecture with UML," *Proceedings of the First Working International Federation for Information Processing*, (San Antonio), February 1999, pp. 145–160.

[IE01] Y. Indratmo and S. A. Ehikioya, "A Formal Framework of Multi-Agent Systems for Electronic Commerce Transactions," *The Journal of Computer Science and Information Management*, Vol. 3, No. 3, (To Appear).

[In01] Y. Indratmo, *A Formal Specification of Web-Based Data Warehouses*, Master's Thesis, Department of Computer Science, University of Manitoba, Winnipeg, Canada, 2001.

[Jo94] G. Jones , "Crosstalk," *The Journal of Defense Software Engineering*, March 1994.

[Ke97] J. C. Kelly, *Formal Methods Specification and Analysis Guidebook for the Verification of Software and Computer Systems Volume II: A Practitioner's Companion*, NASA, California, USA, July 1997.

[Ko96] I. S. Kohane, F. J. van Wingerde, J. C. Fackler, C. Cimino, P. Kilbridge, S. Murphy, H. Chueh, D. Rind, C. Safran, O. Barnett, and P. Szolovits, "Sharing Electronic Medical Records Across Multiple Heterogeneous and Competing Institutions," *Proceedings of American Medical Informatic Association Annual Fall Symposium* , (Washington DC, USA), 1996, pp. 608–612.

[Ku99] E. Kuikka, A. Eerola, J. Porrasmaa, A. Miettinen and, J. Komulainen, "An Object-Oriented Method to Create an SGML DTD of an Electronic Patient Record," Technical Report, Department of Computer Science and Applied Mathematics, University of Kuopio, Kuopio, Finland, 1999.

[La98] M. Lavin, Nathan and Michael, "System and Method for Managing Patient Medical Records," US Patent Number 5,772,585, Application Number 706316, June 1998.

[Le99] P. J. Lees, C. E. Chronaki, E. N Simantirakis, S. G. Kostomanolakis, S. C. Orphanoudakis, and P. E. Vardas, "Remote Access to Medical Records via the Internet: Feasibility, Security and Multilingual Considerations," *Proceedings of Computers in Cardiology*, (Hannover, Germany), September 1999.

[Lu96] M. S. Lundy, "The Computer-Based Patient Record, Managed Care and the Fate of Clinical Outcomes Research," *Florida Family Physician Medical Informatics Issue*, Vol. 46, No. 1, January 1996.

[Mi97] S. Miksh, K. Cheng and B. Haynes-Roth, "An Intelligent Assistant for Patient Health Care," *Proceedings of the First International Conference on Autonomous Agent*, (Marina del Rey, CA, USA), August 1997.

[NA99] Computer Science and Telecommunications Board, *For the Record: Protecting Electronic Health Information*, National Research Council, Washington, D.C, USA, July 1999.

[Oe99] B. Oestereich, *Developing Software with UML*, Addison–Wesley, Reading, UK, March 1999.

[Pl98] C. Plaisant, R. Mushlin, A. Snyder, J. Li, D. Heller, and B. Shneiderman, "Lifeline: Using Visualization to Enhance Navigation and Analysis of Patient Records," *Proceedings of the American Medical Informatic Association Annual Fall Symposium* , November 1998, pp. 76–80.

[Po00] G. Potamias, M. Tsiknakis, D. G. Katehakis, E. Karabela, V. Moustakis, and S. C. Orphanoudakis, "Role-Based Access to Patients Clinical Data: The InterCare Approach in the Region of Crete," *Proceedings of MIE 2000 and GMDS 2000*, (Hannover, Germany), IOS Press, August 2000, pp. 1074–1079.

[Po94] A. D. Poon and L. M. Fagan, "PEN-Ivory: The Design and Evaluation of a Pen-Based Computer System for Structured Data Entry," *Proceedings of the Eighteenth Annual Symposium on Computer Applications in Medical Care*, (Washington D.C, USA), November 1994, pp. 447–451.

[Ro01] Roger S. Pressman, *Software Engineering — A Practitioner's Approach*, 5th Edition, McGraw Hill, New York, USA, 2001.

[Sa94] K. Sandrick, "A 26-Year Quest for the Ultimate Electronic Record," *Health Data Management*, July 1994, pp. 21–23.

[Sa95] M. Saaltnik and I. Meisels, *The Z/EVES Reference Manual (Version 2.1)*, ORA Canada, Ottawa, Canada, December 1995.

[Sc99] "Effective Software Deployment," November 1999. Software Development Online.

[So92] Ian Sommerville, *Software Engineering*, Fourth Edition, Addison-Wesley, Harlow, UK, 1996.

[Sp92] J. M. Spivey, *The Z Notation: A Reference Manual*, Second Edition, Prentice Hall, UK, 1992.

[Wi90] B. T. Williams, J. W. Yoder, and D. F. Schultz, "The MEDIGATE System for Direct Entry of Physical Findings by the Examiner: User Interface Issues," *Proceedings of the International Health Evaluation Association Annual Symposium on The Art and Science of Preventive Medicine*, (La Jolla, CA, USA), 1990, pp. 107–114.

[Yv92] A. L. Yves, M. Maksud, B. Desruisseaux, P. Yale, and St-Arneault, "PureMD: A Computerized Patient Record Software for Direct Data Entry by Physicians using a Keyboardless Pen-Based Portable Computer," *The Sixteenth Symposium on Computer Applications in Medical Care*, (Baltimore, USA), November 1992, pp. 319–346.