INTRODUCTION

1.0 Introduction

Nowadays, with the explosion of Internet, we have seen a rapid advancement in application of information technology in medical sector. Smart card with the mobility advantage, combined with the Internet technology, is presenting a new paradigm for medical information access system.

Ubiquitous Database in Mobile Healthcare System (UDMHC) is developed to provide seamless access interface between a Web browser and Java enabled smart card. The smart card is viewed as mobile repository of web object comprising of medical data object, personal data and the emergency contact list. On the other hand, Internet plays a vital role as a medium to ubiquitously accessing detailed information of each patient record from distributed databases. Hence, this system can support both online and offline information access services. An applet can be dynamically loaded into the browser to perform active browsing and updating of medical information. This applet too, can provide Web links to Internet databases to facilitate wide area access for detailed information. With this system, it is possible for medical-related professionals, such as doctors and pharmacists, to seamlessly access medical records directly from the card using a standard browser interface, either to retrieve information from card, or request for more information online. In addition, as the patient moves between hospitals, clinics or countries, the mobility of the smart card dynamically facilitates a truly ubiquitous access and updating of medical information via a standard Web browser interface.

1.1 Problem Background

As can be seen from most government owned hospitals in Malaysia, they still maintain their patient records in the form of paper charts. This scenario has rendered the almost impossible task of integrating and seamlessly managing patient record across hospitals, clinics and between countries. Thus, delay the process of gaining accurate patient medical information. With UDMHC, we believe that the medical-related professional can quickly access the exact victim's information and give the best treatment to the victim in the shortest time. Moreover, they can get further information such as a video of a recent CT scan, high resolution of X-ray image scan online by inserting the patient's UDMHC card.

A research conducted in United State reported that approximately 140,000 hospital patients die every year from adverse drug reactions because of incomplete or incorrect patient information.[1] For instance, imagine your child is highly allergic to certain medications, but you are both seriously injured in an accident and the hospital is about to give her an injection that could prove deadly. Now imagine you're carrying a smart card encoded with all of her critical medical information. The hospital scans the card and she's out of danger. Smart cards can contribute to a better health care system because of their capacity to securely store a patient's essential personal info, medical history, blood type, allergies, physician contact information.[1] .For emergency, doctors in any hospital can quickly get the patients information by accessing patient medical cards in order to treat the patient with the best and fastest way. This also reduces the possibility of mistreatment while raising the chances of saving the patient's life.

Normally, hospital doesn't share its database and most of the current medical record services are limited to operate within the hospital itself. Recently, with the continue proliferation of Internet technology to home, and office via dial-up, lease line, asymmetric digital subscriber line (ADSL), and cable modem, this UDMHC can access database across hospitals, thus improving the quality of healthcare.

1.2 Project Purpose

The purpose of this project is to develop UDMHC by deploying both Smart Card, Internet infrastructure and XML technology in accessing the medical record of the user and providing a method to retrieve medical information from various databases.

1.3 Project Objectives

This project is developed to achieve several objectives, which are:

i Ubiquitously access of patient basic medical information from smart card and detail information of the patient's records through the Internet.

ii To study and understand technology and specification of smart card, Java Server Page, Java Server, Java Bean, JDBC and distributed database.

Iii To enable the access to various different databases with different database schema.

1.4 Project Scopes

This project involves:

i Accessing and manipulating distributed database, only limited to Ms Accesss and MySQL, with different data schema.

ii User basic personal information, medical record, and relative contact will be embedded into smart card

iii Contact smart card will be used

iv Access Control List will be deployed to different group of users, which are group of admin, doctor, nurse and normal user.

V Updating of smart card and relational database can be done simultaneously

vi Blobs (Binary Large Object) and Clobs (Character Large Object) will be used to manage images and documents.

LITERATURE REVIEW

1.0 Introduction

Smart Card technology is the optimal portable solution for information access, management and improved communication among the various professional involved in the administration of healthcare, while providing strong security measures. Its ability in carrying its record management applet enables medical personnel to quickly gain access of vital patient's medical record at hospital.

With the continue proliferation of Internet technology and faster broadband services, the integrity of smart card and Web based technology improve the quality in accessing databases from distributed places.

Database system have taken us from a paradigm of data processing in which each application defined and maintained its own data, to one in which data is defined and administered centrally. And distributed database technology may change the mode of the working from centralized to decentralized. In distributed database management system (DDBMS), users can not only access the database at their own site but also access data which are stored at remote sites.

1.1 Related Technologies

The subsequent sections will discuss on technologies used in developing the system.

1.1.1 Smart Card

Similar in size to today's plastic payment card, the smart card has a microprocessor or memory chip embedded in it. The chip stores electronic data and programs that are protected by advanced security features. When coupled with a reader, the smart card has the processing power to serve many different applications. As an access-control device, smart cards make personal and medical data available only to the appropriate users. Smart cards provide data portability, security and convenience.

There are two types of smart cards, memory cards and microprocessor cards. Memory cards simply store data and it can be viewed as a small floppy disk with optional security. On the other hand, a microprocessor card, can add, delete and manipulate information in its memory on the card. Similar to a miniature computer, a microprocessor card has an input/output port, operating system and hard disk with built-in security features [3.] In terms of design, the smartcard can either be contact or contactless.

2.1.1.1 Contact Smart Card

Contact smart cards must be inserted into a smart card reader. They have a small gold plate on the front, instead of the magnetic strip on the back like a credit card. When the card is inserted into a smart card reader, it makes contact with electrical connectors that transfer data to and from the chip.

2.1.1.2 **Contactless Smart Cards**

Contactless smart card is passed near an antenna to carry out a transaction. They have an electronic microchip and an antenna embedded inside. These components allow the card to communicate with an antenna / coupler unit without physical contact. Contact less cards are the ideal solution when transactions must be processed very quickly, as in mass-transit or toll collection activities.

The size of the card is determined by the international standard (ISO 7810). The ISO 7816 standard also defines the physical characteristics of the plastic, including the temperature range and flexibility, position of the electrical contacts and how the microchip communicates with the outside world.

All smart cards contain three types of memory: persistent non-mutable memory; persistent mutable memory; and non-persistent mutable memory. ROM, EEPROM, and RAM are the most widely used memory for the three respective types in the current smart cards. Persistent memory is also called non-volatile memory.

ISO 7816 part 1-7, defined by International Standard Organization, contains a set of standards that covers various aspects of smart cards. ISO 7816 consists of:

i. Physical characteristics (part 1) (Appendix C: Figure 1)
ii. Dimensions and location of the contacts (part 2)(Appendix C: Figure 1)
iii. Electronic signals and Transmission protocols (part 3)
iv. Inter-industry commands for interchange (part 4)
v. Application identifiers (Part 5)
vi. Inter-industry data elements (Part 6)
vii. Inter-industry commands for SCQL (Part 7)

Normally, a smart card does not contain a power supply, a display, or a keyboard. It interacts with the outside world using the serial communication interface via its eight contact points. The dimensions and location of the contacts are covered in part 2 of ISO 7816.

A smart card is inserted into a Card Acceptance Device (CAD), which may connect to another computer. Other terms used for the Card Acceptance Device are *terminal*, *reader*, and *IFD* (interface device). They all provide the same basic functions, namely to supply the card with power and to establish a data-carrying connection.

When two computers communicate with each other, they exchange data packages, which are constructed following a set of protocols. Similarly, smart cards speak to the outside world using their own data packages -- called *APDU* (Application Protocol Data Units). APDU contains either a command or a response message. In the context of smart card, the master-slave model is used whereby a smart card always plays the passive role. In other words, a smart card always waits for a command APDU from a terminal. It then executes the action specified in the APDU and replies to the terminal with a response APDU. Command APDUs and response APDUs are exchanged alternatively between a card and a terminal.

Table 2.1: Command And Response APDU Formats

Command APDU						
Mandatory Header				Conditional Body		
CLA	INS	P1	P2	Lc	Data field	Le

The header codes denote the selected command. It consists of four fields: class (***CLA***), instruction (***INS***), and parameters 1 and 2 (***P1*** and ***P2***). Each field contains 1 byte:

i. ***CLA***: Class byte. In many smart cards, this byte is used to identify an application.
ii. ***INS***: Instruction byte. This byte indicates the instruction code.
iii. ***P1-P2***: Parameter bytes. These provide further qualification to the APDU command.

Le denotes the number of bytes in the data field of the command APDU; ***Le*** denotes the maximum number of bytes expected in the data field of the following response APDU.

Table 2.2 : Status Bytes SW1 And SW2 Denote The Processing Status Of The Command APDU In A Card.

Response APDU		
Conditional Body	Mandatory Trailer	
Data field	SW1	SW2

2.1.1.3 Java Card

A Java Card is a smart card that is capable of running Java programs. It contains detailed information for building the Java Card virtual machine and application programming interface (API) in smart cards. The minimum system requirement is 16 kilobytes of read-only memory (ROM), 8 kilobytes of EEPROM, and 256 bytes of random access memory (RAM).

The Java Card VM is built on top of a specific integrated circuit (IC) and native operating system implementation. (Appendix C: Figure 3)The JVM layer hides the manufacturer's proprietary technology with a common language and system interface. The Java Card framework defines a set of Application Programming Interface (API) classes for developing Java Card applications and for providing system services to those applications. A specific industry or business can supply add-on libraries to provide a service or to refine the security and system model. Java Card applications are called *applets*. Multiple applets can reside on one card. Each applet is identified uniquely by its

AID (application identifier), as defined in ISO 7816, part

There are several unique benefits of the Java Card technology, such as:-

i. **Platform Independent** - Java Card technology applets that comply with the Java Card API specification will run on cards developed using the JCAE - allowing developers to use the same Java Card technology-based applet to run on different vendors' cards.

ii. **Multi-Application Capable** - Multiple applications can run on a single card. In the Java programming language, the inherent design around small, downloadable code elements makes it easy to securely run multiple applications on a single card.

iii. **Post-Issuance of Applications** - The installation of applications, after the card has been issued, provides card issuers with the ability to dynamically respond to their customer's changing needs. For example, if a customer decides to change the frequent flyer program associated with the card, the card issuer can make this change, without having to issue a new card.

iv. **Flexible** - The Object-Oriented methodology of the Java Card technology provides flexibility in programming smart cards.

v. **Compatible with Existing Smart Card Standards** - The Java Card API is compatible with formal international standards, such as, ISO7816, and industry-specific standards, such as, Europay/Master Card/Visa (EMV).

2.1.2 Ubiquitous Database

Ubiquitous database places data everywhere. A very small Database Management System (DBMS) implemented on smart card can interact through queries on a mobile

communication. Ubiquitous database attaches such DBMS to real world "objects" physically, and then allows different organizations to share information retrieved directly from physical goods, materials, or persons. Figure 2.1 below depicts the concept of ubiquitous database on the smart card and its interaction with the DBMS.

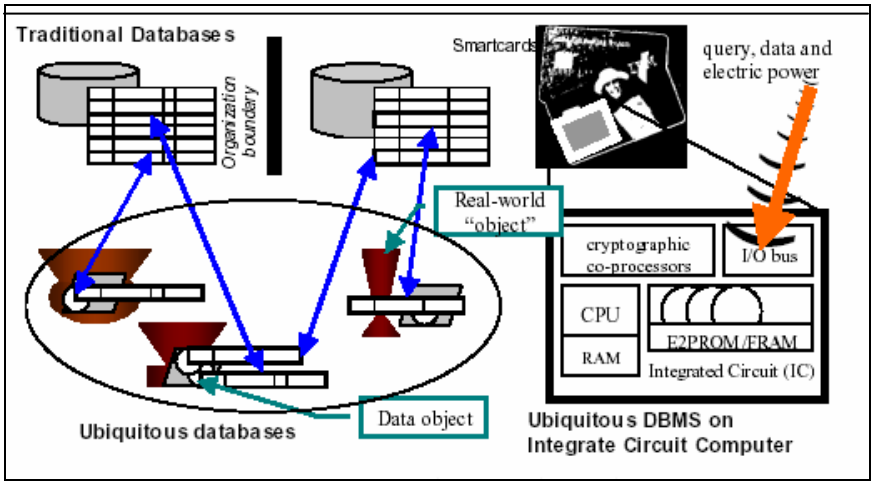


Figure 2.1 : Concept of Ubiquitous Database on the Smart Card

The ultimate goal of *ubiquitous computing* is to place computers everywhere in the real world environment, providing ways for them to interconnect, talk and work together. [5] The concept is now shifting computing paradigm from machines in a room to the augmented contexts in the real world. Similarly, ubiquitous database will make data everywhere possible. Every real-world object originally has information such as properties and its historical changes. Traditional databases collect such information to manage in a central manner. However, the central management, although it is highly efficient, does not necessary meet the demand to establish data applications across different organizations, as electronic commerce and digital libraries today demand.

The ubiquitous database augments "object" that manages information about itself. A database-augmented "object" enables data application integrations through the movement of "object" in the real world. An augmented product moving from one company to another can carry electronic updating records at the same time. An augmented museum piece can play different roles in showing data for visitors, researchers and librarians. A person wearing a small database can autonomously interact

with social information systems under privacy controls. The ubiquitous database provides anyone with the method to retrieve information directly from the real world “objects”. To share information across organizations, the ubiquitous database becomes a database environment.

2.1.2.1 Ubiquitous Database Architecture

Since small processor and limited memory resource, it is very difficult to implement the full functionality of rich DBMS. We therefore divided the function of DBMS into two parts on a host PC and a smart card. The part on the host is a preprocessor that transforms a query language into primitive commands. In addition, it takes charge of the management of schema based view and transaction roll-backing. On the other hand, the part on the smart card is a command processor that executes primitive commands that create, read, write and delete data objects. Also, the permission of the executions is controlled on it. Both parts are connected through an encrypted Internet communication.

2.1.2.2 Concurrency Control

To synchronize with a remote database, two-phases basic commitment schemes have been implemented. Since it is difficult to record the full state of a transaction on the card, the pre-processors are designed to control the commitment and the rollback. Please refer to the diagram below for two-phase commitment scheme adopted.

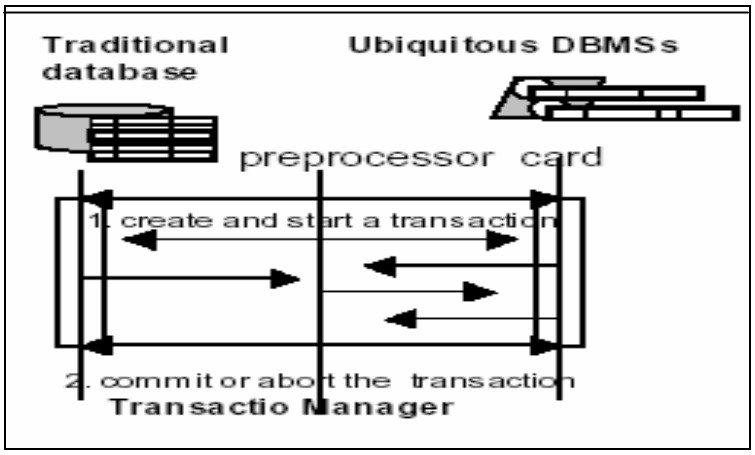


Figure 2.2 : 2-phases commitment

2.1.3 Java Database Connectivity (JDBC)

JDBC is quite similar to Microsoft's Open Database Connectivity (ODBC). Both are based on the X/Open Call Level Interface (CLI) specification . JDBC function as a translation layer between the application and the data source and provide a standard way of connecting with a variety of SQL, relational databases.

The application makes call to JDBC API to open a connection with the database, retrieves and updates data, executes commands on the data source, and closes the connection. At the other end, the database drivers connect either to a specific database, or to another protocol (such as ODBC or middleware product). Since databases vary in their support of SQL, the database driver needs to handle any translation issues between the JDBC commands and the database engine. Databases also vary widely in the protocols used to connect to the engine.

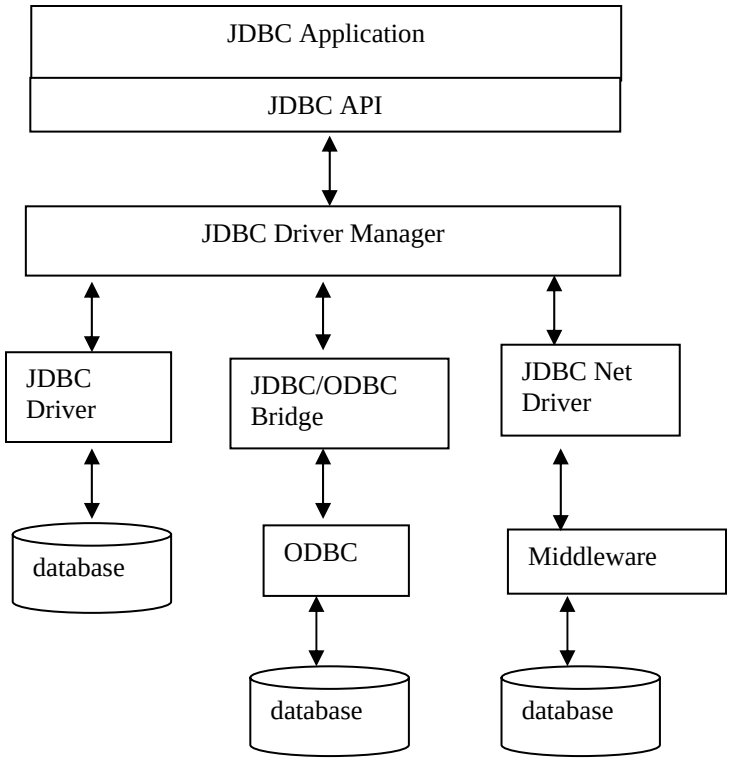


Figure 2.3 : JDBC Library Structure

2.1.4 Java Servlet

Servlet are small units of Java code that execute quickly on the server in response to a browser's request for a web page. Servlet execute on the server side of web connection. Just as applets dynamically extend the functionality of web browser, servlets dynamically extend the functionality of web server.

Servlet offers several advantages:-

i. **Performance is significantly better**. Servlet execute within the address space of a web server. Creating a separate process to handle each client request isn't

necessary.

ii. **Servlet are platform-independent**, because they are written in Java. Several web servers, from vendors such as Sun, Netscape, and Microsoft, offer the Servlet API. Program developed for this API can be moved to any these environments without recompilation.

iii. **The Java Security** Manager on the server enforces a set of restrictions to protect the resources on a machine.

iv. **The full functionality of the Java class libraries** is available to a servlet. It can communicate with applets, database, or other software via the sockets and RMI mechanisms.

2.1.5 Distributed Database

Distributed database is a logically interrelated collection of shared data, physically distributed over a computer network. A Distributed Database Management System (DDBMS) consist of a single database that is split into number of fragments, each of which is stored one or more computers under the control of a separate DBMS. The computers are connected by a communication network. Each site is capable of independently processing user request that require access to local data and also capable of processing data stored on the other computer in the network.

DBMS have the following characteristic:-

i A collection of logically related data is distributed over a number of different computers.
ii The computers are linked by a communication network.
iii The data at each site is under the control of a DBMS.

iv The DBMS at each site can handle local applications, autonomously.

v Each DBMS participates in at least one global application.

2.1.5.1 Homogeneous/Heterogeneous DBMS and XML

A DBMS may be classified as homogeneous or heterogeneous. In a homogeneous system, all sites use the same DBMS product. In a heterogeneous system, sites may run different DBMS products, which need not be based on the same underlying data model and so may be composed of relational, network, hierarchical and object-oriented DMBS.

Homogeneous systems are easier to design and manage. This approach provides incremental growth, making the addition of a new site to the distributed system easy, and allows increased performance by exploiting the parallel processing capability of multiple sites.

Heterogeneous system usually result when individual sites have implemented their own database and integration is considered at a later stage. In a heterogeneous system, translation is required to allow communication between DBMSs. To provide DBMS transparency, users must be able to make request in the language of the DBMS at their local site. The system then has the task of locating the data and performing any necessary translation. Data may be required from another site that may have different hardware, different DBMS products. If the hardware is different but the DBMS products are the same, the translation is straightforward, involving the change of codes and word lengths. If the hardware is the same, but the DBMS products are different, the translation is complicated, involving the mapping of data structures in one data model to the equivalent data structures in another data model. It is also necessary to translate the query language used (for example, SQL SELECT statements are mapped to network FIND and GET statements).

The typical solution used by some relational system that are parts of a heterogeneous DDBMS is to use gateways, which convert the language and model of each different DBMS in to the language and model of the relational system. However, the gateway approach has some serious limitations. First, it does not support transaction management, even for a pair of systems. In other words, the gateway between two systems is merely a query translator. For example, a system may not coordinate concurrency control and recovery of transaction that involve updates to both databases. Second, the gateway approach is concerned only with the problem of translating a query expressed in one language into an equivalent expression in another language. As such, it does not address the issues of homogenizing the structural and representational differences between different schemas.

XML is an Extensible Markup Language, a widely used system for defining data formats. XML provides a very rich system to define complex documents and data structures such as invoices, molecular data, news feeds, glossaries, inventory descriptions, real estate properties, etc. As long as a programmer has the XML definition for a collection of data (often called a "schema") then they can create a program to reliably process any data formatted according to those rules.

2.2 Specific Technologies Applied In The Development of the Project

In developing the system (Ubiquitous Database for Mobile Healthcare), we have deployed these technologies.

2.2.1 The Java Card 2.0 Framework

Smart cards have been in the market for 20 years, and most of them are generally compatible with ISO 7816 Parts 1-7 and/or EMV. The Java Card Framework is designed to easily support smart card systems and applications. It hides the details of the smart

card infrastructure and provides Java Card application developers with a relatively easy and straightforward programming interface.

The Java Card framework contains four packages: -

Table 2.3: Packages of Java Card Frame Work

Package Name	Description
javacard.framework (Appendix C: Figure 12)	This is the core package on the card. It defines classes such as Applet and PIN, which are the fundamental building blocks for Java Card programs and APDU, System and Util, which provide runtime and system service to Java Card programs, such as APDU handling and object sharing
javacardx.framework (Appendix C :Figure 13)	This package provides an object-oriented design for an ISO 7816-4 compatible file system. It supports elementary files (EF), dedicated files (DF) and file-oriented APDUs as specified in ISO7816
javacardx.crypto and javacardx.cryptoEnc	Those two packages support cryptographic functionality required in smart cards

2.2.2 Open Card Framework

OpenCard is an open standard that provides interoperability of smart card applications across NCs, POS terminals, desktops, laptops, set tops, and PDAs. OpenCard can provide 100% pure Java smart card applications. Smart card applications often are not pure because they communicate with an external device or use libraries on the client. (Appendix C: Figure 6)

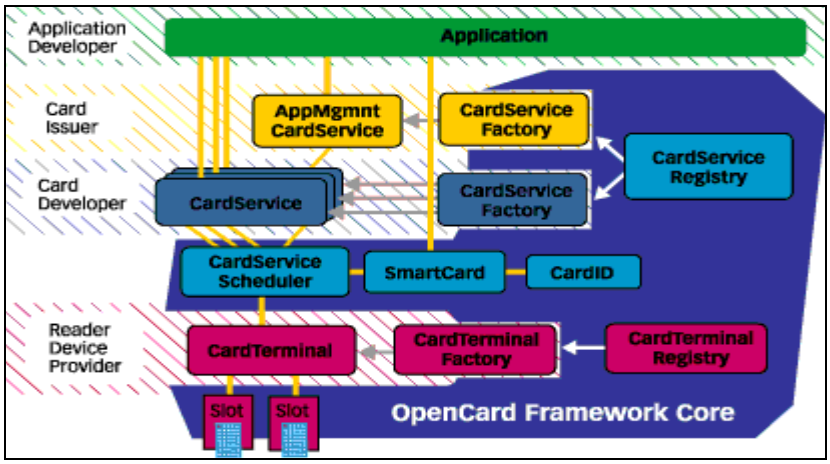


Figure 2.4: The Open Card Framework and the Components that can be Plugged into it

The OpenCard Framework allows us to develop end-to-end solutions using smart cards that are not bound to one platform, card, or application. OpenCard achieves this with an architecture that provides two primary subsystems, one for *card terminals* and one for *card services.* Card terminals are devices you insert smart cards, Java Rings, and the like, into. Card services are used by an application to communicate with the application on the card inserted into a terminal. For inserted cards, OpenCard can automatically select and load the right card service implementation.

The OpenCard Framework integrates CardTerminal classes and CardService classes and offers a standardized, high-level interface to applications. Card terminal manufacturers who want to make their terminals available to OpenCard applications need to provide a CardTerminal class, which encapsulates card terminal behavior, and a CardTerminalFactory class. The card terminal factory has to be registered with the card terminal registry, which keeps track of all card terminals to the OpenCard Framework and will be used by the Framework to create CardTerminal instances when the Framework is initialized. Card services offer smart-card functionality to application developers via high-level interfaces. Smart-card manufacturers have to provide CardService classes encapsulating the behavior of their smart cards and a

CardServiceFactory class. The card service factory must be registered with the OpenCard Framework and is thereafter used by the Framework to instantiate card services.

OpenCard provides an API that allows different card readers, different platforms, and different Java Cards to be used by the same Java code with no change. With OpenCard we can run Java smart card applications in our office, on our set-top, and on our personal data assistant -- and, of course, on Windows platforms as well.

In order to use a smart card, we need to be able to read the card and communicate with it using an application. OpenCard provides a framework for this by defining interfaces that must be implemented. The OpenCard framework defines several of these interfaces. Once these interfaces are implemented, we can use other services in the upper layers of the API. For example, with a properly interfaced reader, OpenCard can start a Java card agent whenever the card is inserted. The card agent can then communicate with applications on the smart card via the card terminal in the context of a session.

OpenCard also provides developers with an interface to PC/SC (a smart card application interface developed by Microsoft and others for communicating with smart cards from Win32-based platforms for PCs) for use of existing devices on Win32 platforms.

OpenCard provides a solution to the problem of interfacing different devices for reading cards to these platforms.

The following are some of the advantages of using OpenCard:

i. OpenCard allows you to write us own services while standardizing on some simple concepts. Ideally, as developers write to OpenCard, we can reuse terminals (smart card readers) and services written by others.
ii. OpenCard was developed with the Web in mind, and it is very easy to add dynamic downloading over the Web to OpenCard. So if the market demands it, billions of cards can be programmed routinely on traditional and non-traditional platforms.

iii. Users can load Java Cards directly. This means custom applications can be added to the card. Many of the current smart card development environments only work on Windows, which is unacceptable to many developers who use Linux or Solaris for software development.

2.2.2.1 Card Services

There may be several instances of card services per card, owned by different threads. A card service offers a certain functionality of a card to the application developer via a high-level interface. Figure 2.5 shows the architecture of OpenCard.
OpenCard defines interfaces for standard functions, such as file system access or generation of digital signatures. Card services for cards that offer such functions should implement these interfaces. For specialized cards, dedicated interfaces may define losing interoperability .

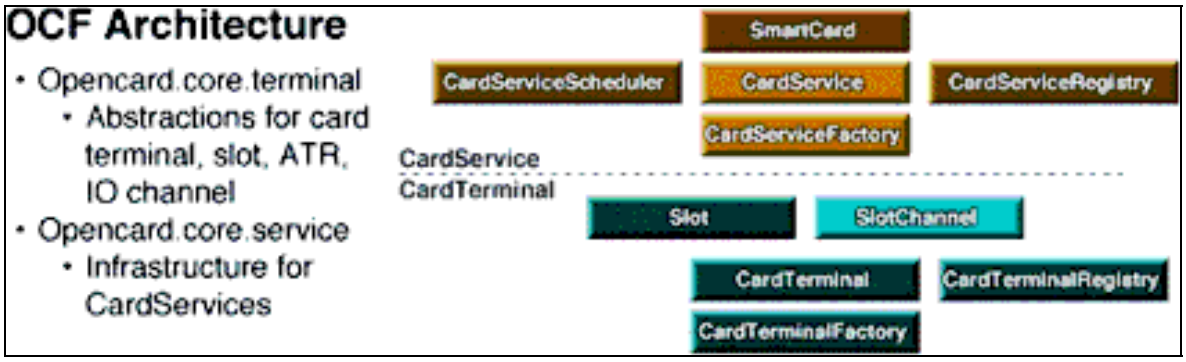


Figure 2.5 : OpenCard Architecture

2.2.2.2 Card Channel

To communicate with a smart card in a card terminal, card services use Card Channel objects, which represent a communication link to the smart card and offer methods for sending commands to smart cards and for receiving the responses. Concurrent access of card services to the card via a card channel is scheduled by the Card Service Scheduler, which serializes the access of different services to the card channel.

2.2.2.3 Card Applet Proxies

For ISO file system cards that have a fixed set of commands for accessing files on the card, OpenCard defines the File System Card Service interface. However, Java Cards are much more flexible than conventional file system-oriented smart cards. They may contain a set of different applets, each supporting a different card applet-specific command set. The only common properties of these applets are that they may be identified by an application identifier (AID), selected, and once selected, can process APDUs (application protocol data units).Because Java Cards are more flexible than other smart cards (for example, file system cards) we need a more flexible concept for accessing and using Java Cards or, to be more exact, the applets on Java Cards. We use card applet proxies representing the applets on the card. Proxies are card applet-specific: each card applet proxy class belongs to a particular Java Card applet. Each proxy class has to know the application identifier of the card applet to communicate with and the protocol for interaction with that card applet.

Proxies are the most suitable method for interacting between applications and applets on Java Cards. An application obtains a proxy card service that represents the card applet on the card. The application uses high-level methods offered by the proxy. Whenever the application invokes a proxy method, the proxy starts communication with the card applet on the card and generates some result, which it returns to the application.

Figure 2.6 : Interaction of application, card applet proxy, and Java Card. The proxy conducts the protocol with the card on behalf of the application.

Card applet proxies may be used in multithreaded programs. This means they may be instantiated several times by different threads so that concurrent access by different instances to the card must be serialized. As there might also be several instances of the same card applet proxy class, it must be possible to share the associated card applet's state so that the different proxy instances properly interact.

As specialized card services before sending APDUs, card applet proxies must allocate a card channel for communication with the Java Card from the card service scheduler -- like any other card service. If the card channel has already been allocated by another card applet proxy, the threads of activity of card applet proxies trying to allocate it are blocked until the current owner of the channel releases it.

A card channel may hold several *state objects* at most one for each card applet on the card. The state object represents the state of a card applet on the card. If, for example, we have a card applet that simulates a file system, the state would consist of the currently selected directory and information about access conditions. There may be applets that are stateless; their associated proxies don't need a state object .As access to the channel is synchronized and states can only be obtained from the channel, there is no need for additional synchronization of access to states.

All proxies need to send APDUs to the card applets they represent. This function is implemented in a common base class of all proxies, which we name Applet Proxy service. In order to send an APDU to a card applet, the card applet must be selected -- except if it is the currently selected card applet. To avoid unnecessary selections, the proxy services have to keep track of the currently selected card applet. There may exist several instances of proxy services for one open platform card simultaneously. In this case, it must be assured that all proxy services accessing the same card also share the representation of the card's state; that is, the currently-selected card applet.

On a Java Card with several card applets, selection of one applet always causes de-selection of another applet, potentially causing the deselected applet to loose its state. This potential state loss requires that applet proxies are notified whenever a card applet --

other than the one they are associated with -- is selected, so that they can update their representation of the associated card applet's state. The notification mechanism works as follows: in the constructor of the base class Applet Proxy, each applet proxy registers with the card state object of its associated card as an *applet selection listener*. As mentioned above, the class Applet Proxy, which is the base class of all applet proxy services, offers methods for sending APDUs to the associated card applet to derive classes. These methods implicitly select the associated card applet if it is not already the current applet.

In this case, these methods modify the card state to indicate that now a different card applet has been selected by calling the method *setSelectedAppletAID* of the card state object. This method updates the selected card applet in the card state and notifies all other card applet proxies associated with card applets on the same card by calling their applet Selection methods. The default implementation of this method in the base class is empty; it must be overwritten in derived card applet proxies if their associated card applets may lose or change their states when other card applets are selected.

Proxy services will usually extend the generic Applet Proxy service and use inherited methods for communication with associated card applets. For obtaining and releasing exclusive communication with the card, inherited methods are used as well.

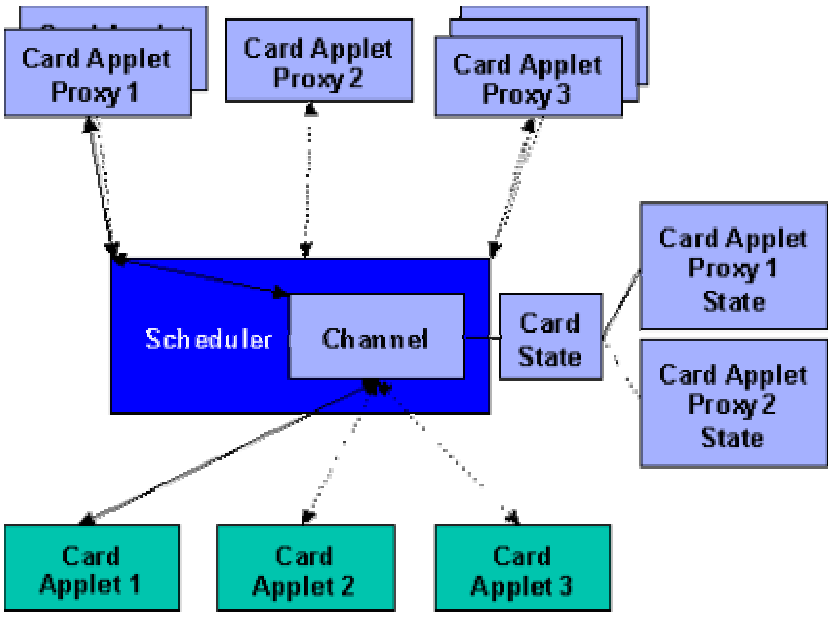


Figure 2.7 : Six Card Applet Proxies Accessing One Java Card. Several Card Applet Proxies May Be Using The Same Card Applet. All Proxies Are Accessing The Java Card Via A Shared Channel. Access To This Channel Is Synchronized By A Card Service Scheduler. For Applets That Have State, Appropriate State Objects Keeping Track Of Those Applets' States Can Be Attached To The Channel.

2.2.4 OpenCard Architecture

OpenCard provides architecture for developing applications in Java that utilize smart cards or other ISO 7816-compliant devices on different target platforms such as Windows, network computers, Unix workstations, Webtops, set tops, and so on. The OpenCard Framework provides an application programming interface (API), which allows you to register cards, look for cards in readers, and optionally have Java agents start up when cards are inserted in the reader. (Appendix C: Figure 11)

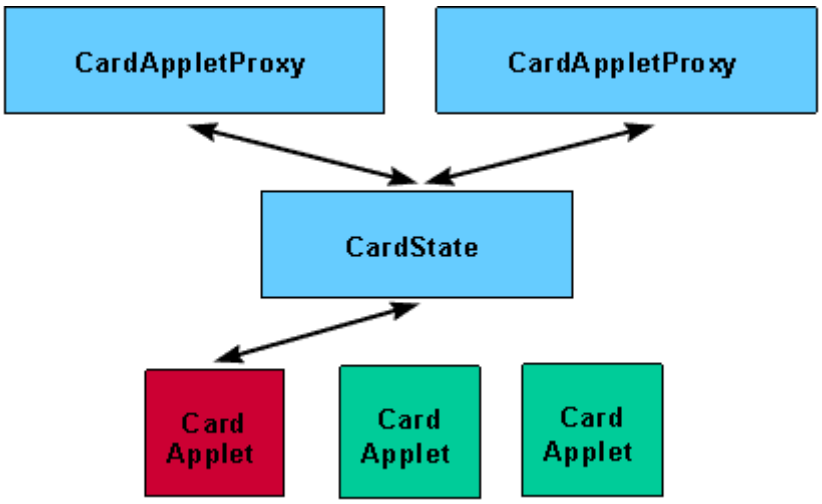


Figure 2.8 : Card applet proxy services accessing the same Java Card. All card applet proxy services accessing the same Java Card share a common card state, which keeps track of the currently-selected card applet. By checking the state for the currently-selected card applet before actually making a selection, card applet proxy services can avoid sending unnecessary select-applet APDUs.

The architecture of the OpenCard Framework is made up of the Card Terminal, the Card Agent, the Agents and/or applications that interact with these components.

OpenCard consists of four Java packages with the prefix *opencard*:

i. *application*
ii. *io*
iii. *agent*
iv. terminal

2.3 Summary

Ubiquitous Database in Mobile Health Care (UDMHC), functions as enhancement and complementary to Ubiquitous Emergency Handling (EHS), as well as provide health care qualify assure to all, is focusing on the marriage of smart card and

internet communication. Communication of these entities is achieved by using standard http request reply via UDMHC system. With the ability of open card technology, UDMHC is capable be operated and used in multi operating system platform, hence prove the ubiquitous database management of the smart card.

UDMHC hopefully will bring an impact on data application integration in the context of real world. Lastly, it is hoped that this system will improve and maintain a good quality of health care as well as provide useful medical information in time of accident and emergency, and present a new paradigm for medical information access system.

METHODOLOGY

2.0 Introduction

Methodology used for developing Ubiquitous Database in Mobile Health Care (UDMHC) in Emergency Handling System is Evolution Prototyping. It includes developing trial system or experiment in short time sequence for evaluation by end users. In other words, prototype was an early version of a system or some of its functional parts that could be examined by end users. Its purpose is to detail and define system model interactively until users' requirements were met. Prototyping methodology's criteria includes:-

i. User evaluation
   If user found any deficiency in the system, suitable and prompt modification could be made.
ii. Development
   Early presentation of the system allowed early feed back from the users. Modification and upgrade could be deployed while developing. Thus, the duration allocated for development can shorten.
iii. Users involvement
   The involvement of users in development will ensures that heterogeneous specifications and requirements of users were fulfilled when system complete.
iv. Reduce risk
   By fulfilling most of the users' requirements, complete system will be commonly accepted by users without much complaint. Besides, eliminating major reconstructing can reduce cost and duration of development.

3.1 Evolution Prototyping

Evolution prototyping is a system development methodology which is similar to requirement prototyping, except evolution prototype would not be thrown away. Prototype developed would be reconstructed to meet users' specifications from time to time until a real system completed. The objective of this method is to produce a functional system for end users. Starting from users' requirements, prototype was built and followed by users' evaluation until users' specifications were met. Phases of evolution prototyping methodology are listed as below:

(i) Determining User's Requirement

Market research and user's interview were carried out to acquire the appropriate users' requirement. Microsoft Access and MySQL databases were chose to be used. Necessary data fields that should be included in the database were determined. Some interviews to doctors were done to acquire necessary medical information that should be stored into a smart card. By the end of this phase, early designs were produced according to each modules and functions.

(ii) Developing Functional Prototype

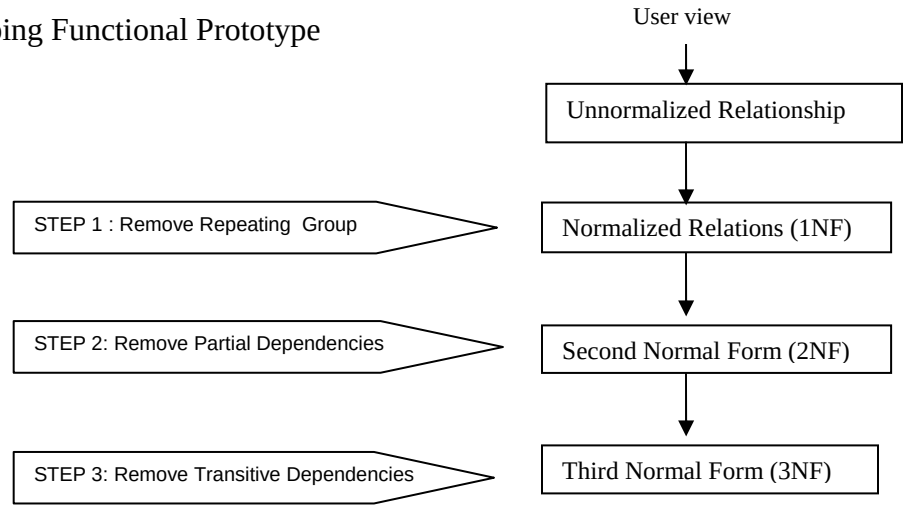


Figure 3.1: Steps In Normalization

At the beginning of this phase, normalization was done to the collection of database. Normalization was the transformation of complex user's views and data stores to a set of smaller, stable data structures. In addition to being simpler and more stable, normalized data structures were more easily maintained. (Appendix C: Figure 15). Normalizes of data structure were done in three steps. Each step involves an important procedure to simplify the data structure.

The relation derived from the user view or data store will most likely be unorganized. The first stage of the process included removing all repeating groups and identifying the primary keys. In order to do this, the relation needed to be broken up into two or more relations. At this point, the relations might already be of the third normal form, but more likely more steps will be needed to transform the relation to the third normal form.

The second step ensured that all none key attributes are fully dependent on the primary keys. All partial dependencies are removed and placed in another relation.

The third step removed any transitive dependencies. A transitive dependency was one in which none key attributes were dependent on other none key attributes.

(iii) Examining and Evaluating

Functional prototypes were thoroughly tested for its reliability. Relational database will be examined concerning the integration and retrieving data via the online smart card. Evaluation will be executed over the working prototype, reviewing its quality and speed as well as other required aspect.

(iv) Checking and Modifying Prototype

The prototype, which still had to be improved during the evaluation phase, would be passed to this step. This prototype would be debugged and be modified to enhance and upgrade its quality to match the requirements. For instance, applet stored in the smart card be upgraded and modified to match the requirement of fastest accessing precise medical records. Then, first phase will go for another round.

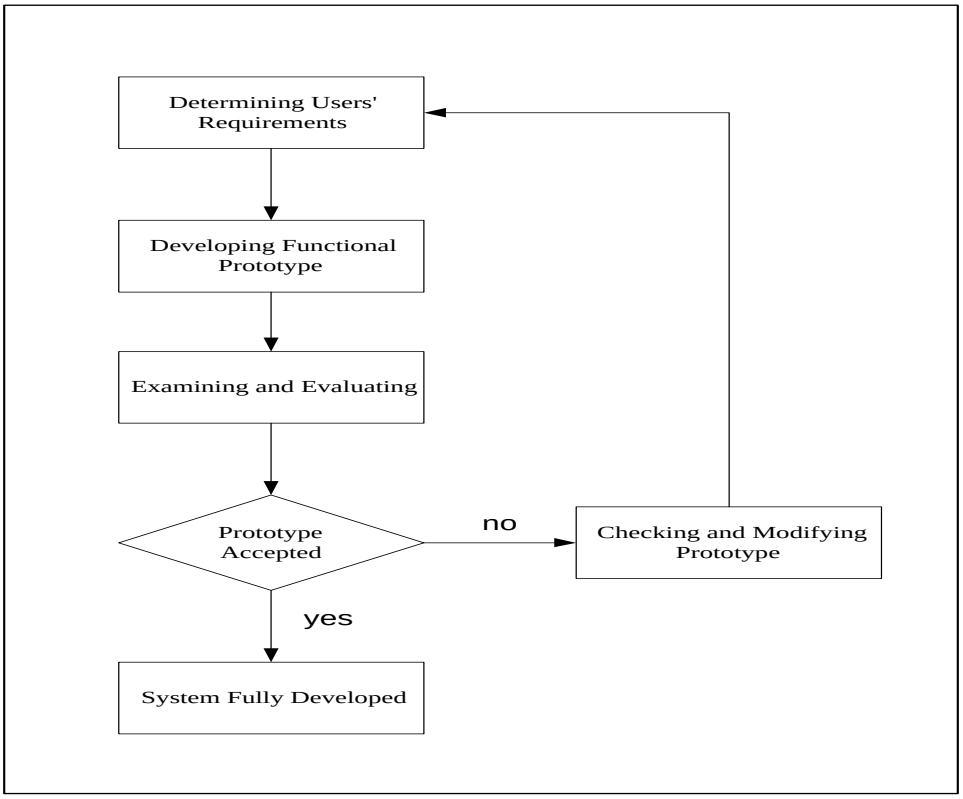


Figure 3.2: Flow of Evolution Prototype Methodology

3.2 Methodology Justifications

There were several justifications to use evolution prototyping: -

i. This methodology was suitable to develop UDMHC because technologies related to UDMHC might keep expanding from time to time.

ii. Eliminating problems caused by insufficient analysis of users' requirements by involving users in development phases.

iii. UDMHC was an interactive and real-time application with complex interfaces. Prototyping was much suitable for system development since users' specifications will be determined thoroughly.

iv. Error in the system can be verified promptly by examining the completed modules while others in progress.

v. Cost and duration used for prototyping methodology is sensible.

3.3 Hardware and Software Requirements

The following hardware and software were used to develop UDMHC system.

3.3.1 Hardware

Hardware was the tools needed to develop UDMHC system. We needed the hardware for accessing client server via HTTP protocol, and smart card for deploying ubiquitous database technologies.

3.3.1.1 Server / Internet Server

Web server served HTML pages to users. UDMHC involved retrieving information online and offline. Therefore, a web page was needed to embed a Java Applet to gaining data. PC or workstation with Pentium III 500 MHz processors or above to be served as Web server, database server and application server. High processing power and memory capacity was needed to process large amount of data thus 128 MB and above RAM is required. Hard disk with at least 20 GB capacities was ideal for storage. More processing power, memory and storage means the system can support more users.

3.3.1.2 Smart Card Reader

This project deployed the usage of smart card. Thus, smart card reader was needed as one of the hardware requirement. Smart card reader would read information from card and send information from host terminal to smart card as well.

3.3.1.3 Smart Card

A smart card was used to store user basic data like name, address, photograph, medical information such as blood types, drug allergies and regular prescribed drugs, user's medical history.

3.3.1.4 Modem

Modem was needed as device to convert a digital signal into an analog signal to be carried by a public access phone line. In the other hand, it's also a device that converts the analog signal received over a phone line into digital signals usable by our computer.

3.3.2 Software

Software was the program used to develop this UDMHC system. These project was written in Java . The following was the software used to develop this project:

3.3.2.1 Java 2 Development Kit (JDK) v1.4.0 and Java Card API 2.0

Java programming provides portabilty and security. Since Java is object-oriented and platform independent, it is easier to develop Java application. Java is popular and evolving as a powerful programming language covering every requirements of developers. Free JDK can be reviewed and downloaded at http://www.sun.com . With the invention of Java Card 2.0 API, Java has become one of the language that supports smart card programming. Therefore, it's suitable for the implementation of Ubiquitous Database in Mobile Health Care.

3.3.2.2 Java Web Service All In One

The Java™ Web Services Developer Pack ("Java™ WSDP") is an all-in-one download containing key technologies to simplify building of web services using the Java™ 2 Platform. The Java Web Services Developer Pack, includes the following:

i. Java™ XML Pack which includes the following:
   - Java™ API for XML Messaging ("JAXM") 1.0 EA 1
   - Java™ API for XML Processing ("JAXP") 1.2 EA 1 (with XML Schema support
   - Java™ API for XML Registries ("JAXR") 1.0 EA 1
   - Java™ API for XML-based RPC ("JAX-RPC") 1.0 EA 1

ii. JavaServer Pages™ Standard Tag Library ("JSTL") 1.0 EA 3

iii. Ant Build Tool 1.4.1

iv. Java WSDP Registry Server 1.0 EA 1

v. Tomcat Java™ Servlet & JavaServer Pages™ container 4.1-dev

This release of the Java Web Services Developer Pack has been tested with various configurations with the Java™ 2 SDK, Standard Edition version 1.3.1_01, 1.3.1_02, and 1.4 on the following platforms:

i. Solaris™ 2.8

ii. Windows 2000, Professional Edition

iii. Windows XP, Professional Edition

iv. RedHat Linux 7.2

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

3.3.2.3 Internet Explorer 4.0 or Above (Web Browser)

Web browser was required to direct user to the appropriate web site.

3.3.2.4 Windows 98, 2000

Windows 98 or 2000 was used as the main platform for project development. Since most software and emulators are supported by Windows OS, project development is much easier by eliminating complicated OS reconfiguration.

3.3.2.5 Microsoft Access 2000

Microsoft Access 2000 is a complete database and data analysis package for keeping user information.

3.3.2.6 MySQL

MySQL is popular database and is one of the product of MySQL AB (www.mysql.com). Recently, MySQL released as open source under General Public Licence. MySQL is popular with ISPs and Web developers because of its speed and reliability and the flexibility of its access control system. MySQL can support in multi-platform, such as Windows 98/2000, Mac OS, Linux, Solaris.

3.4 Assumptions

In order to fully implement this UDMHC system, some assumptions must be made:-

i. Basic medical record stored in the smart card is enough to provide treatment in emergency case.
ii. All hospitals, and medical centers are equipped with the smart card reader.
iii. Internet transmits data in high bandwidth and low delay
iv. The medical records of the card user are not fraud
v. English language is used for the standard language for the medical record
vi. All the professionals subscriber is formally registered as legal professionals

DESIGN AND ANALYSIS

4.0 Introduction

Basically, UDMHC deploys smart card technology, access control level authentication and online distributed databases. Java smart card contained user basic information, relative contact and medical information in conjunction to achieve ubiquitous database. With the deployment of Access Control Level, each group of user will have different priorities to access certain pages and program functionalities. In addition, the function of password error tries limit count is deployed. The user just have three changes to key in correct password, else, the smart card will be terminated. Client Server approach via HTTP enable user's to get detail information from distributed database

4.1 Access Control Level Design

There are four categories of users to use this system. Each of the user have different access control level and authories to access some pages and program functions. The categories can be divided to admin, doctor, nurse and normal user. The following tables show the access level of each categories.

Table 4.1 Access Control Level Design

Functions	Admin	Doctor	Nurse	Normal User
Login	Yes	Yes	Yes	Yes
change password	Yes	Yes	Yes	Yes
card lost setting	Yes	No	No	No
Unblock setting	Yes	No	No	No
create new card	Yes	No	No	No
add user	Yes	No	No	No
modify data	Yes	Yes	No	No
save data	Yes	Yes	No	No
refresh data	Yes	Yes	Yes	Yes
access detail information online	Yes	Yes	Yes	Yes
change card	Yes	Yes	Yes	No
add/modify website database	Yes	Yes	No	No

4.2 Smart Card Database Design

Smart card database consist of 5 tables, which are status table, link_info table, personal_info table, medical_info table and contact table. Please refer Appendix A for details.

4.3 Hospital Database Design

Every hospital has the same data schema of the database design, which consist of 15 tables.

4.3.1 Database Relationship Design

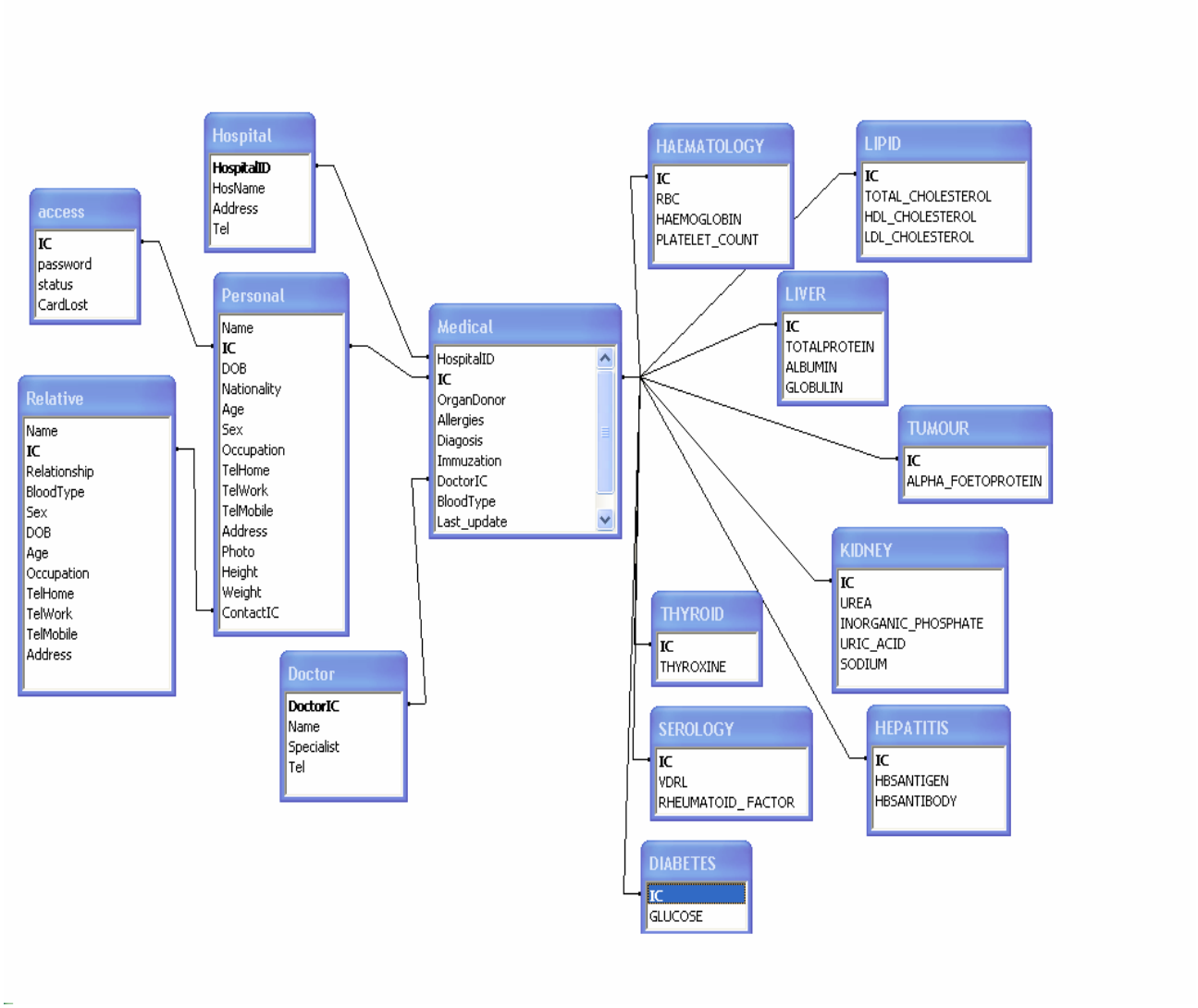


Figure 4.1: Hospital Database Relationship Design

All of the tables are linked by the foreign key as below:

Table 4.7: Primary Key And Foreign Key of Each Table

Table Name	**Primary Key**	**Foreign Key**
Access	Ic	Ic
Relative	Ic	Ic
Personal	Ic	contactIC, ic
Doctor	Ic	Ic
Medical	Ic	Ic, HospitalID
Hospital	HospitalID	HospitalID
Diabetes	Ic	Ic
Hepatities	Ic	Ic
Haematology	Ic	Ic
Lipid	Ic	Ic
Thyroid	Ic	Ic
Liver	Ic	Ic
Kidney	Ic	Ic
Serology	Ic	Ic
Tumour	Ic	Ic

4.4 Architecture Of UDMHC in Emergency Handling System

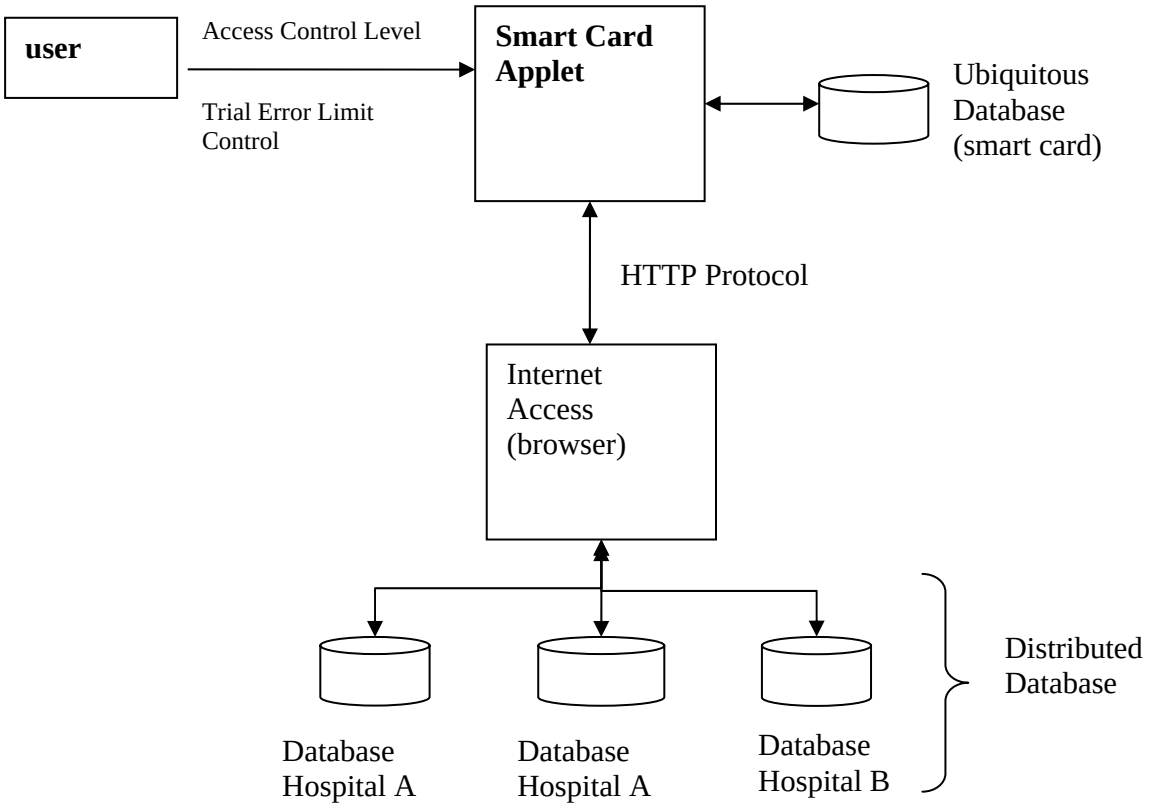


Figure 4.2 Architecture Of UDMHC In Emergency Handling System

Architecture of UDMHC consist of 4 main components, which are the smart card applet, ubiquitous database, browser, and distributed database. Applet works as an interface to the user to access either ubiquitois database or distributed database. Smart card is functioned as a repository for basic personal information, medical information and emergency contact information. In order to achieve data from ubiquitous database, user is prompted to enter pin. Tries error limit control is deploy to prevent Bruce Force attack. User just allow to try maximum 3 times of the password, if can't get the valid password, the card will be blocked. Access Control Level is enforced to make sure that each status

of user have different prioritues to access the program functionality. User can access distributed database from different hospital based on HTTP protocol.

4.5 Use Case / Sequence Diagram

Use case and sequence diagram show how the flow and the process of the Ubiquitous Database in Mobile Health Care in Emergency Handling System.

4.5.1 Use Case and Sequence Diagram for Ubiquitous Database Access

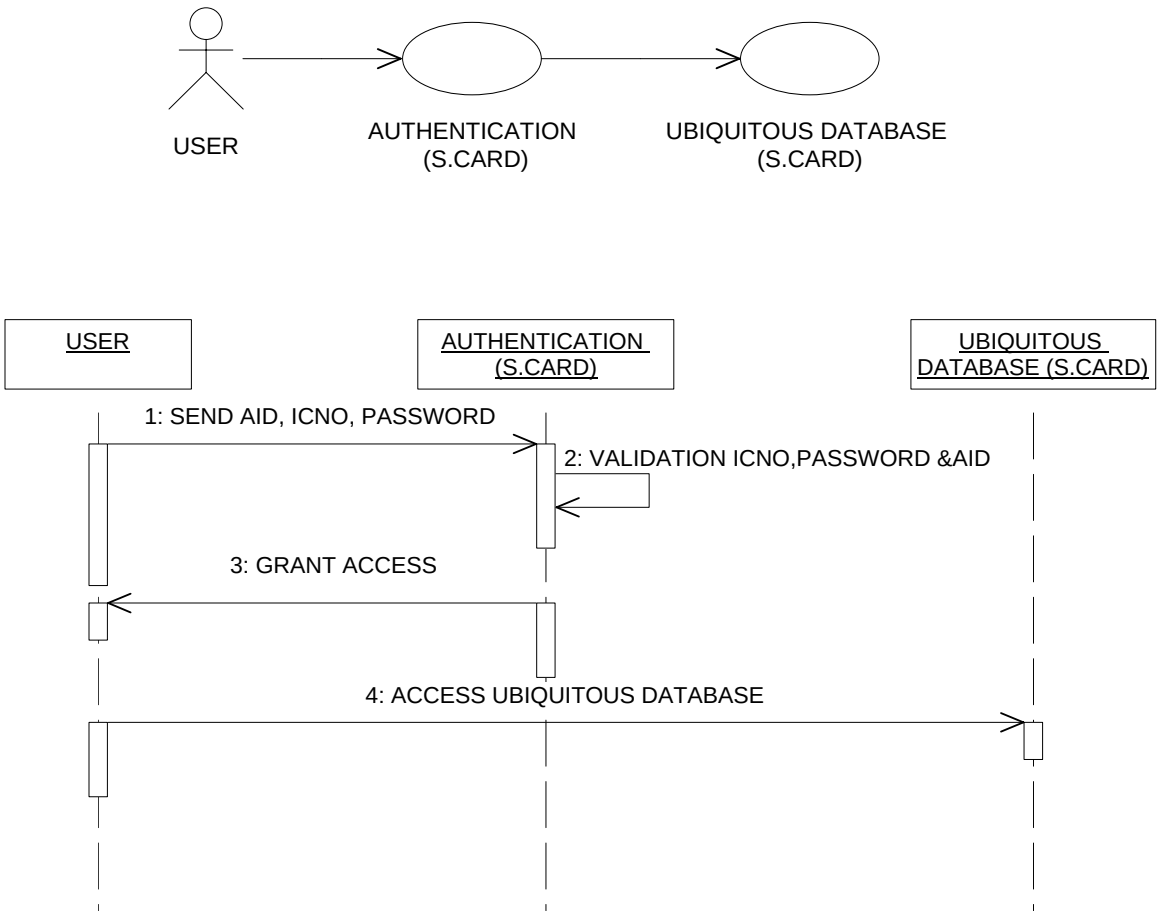
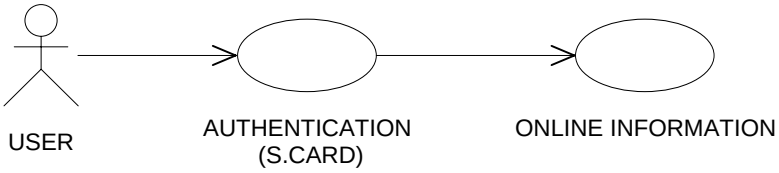
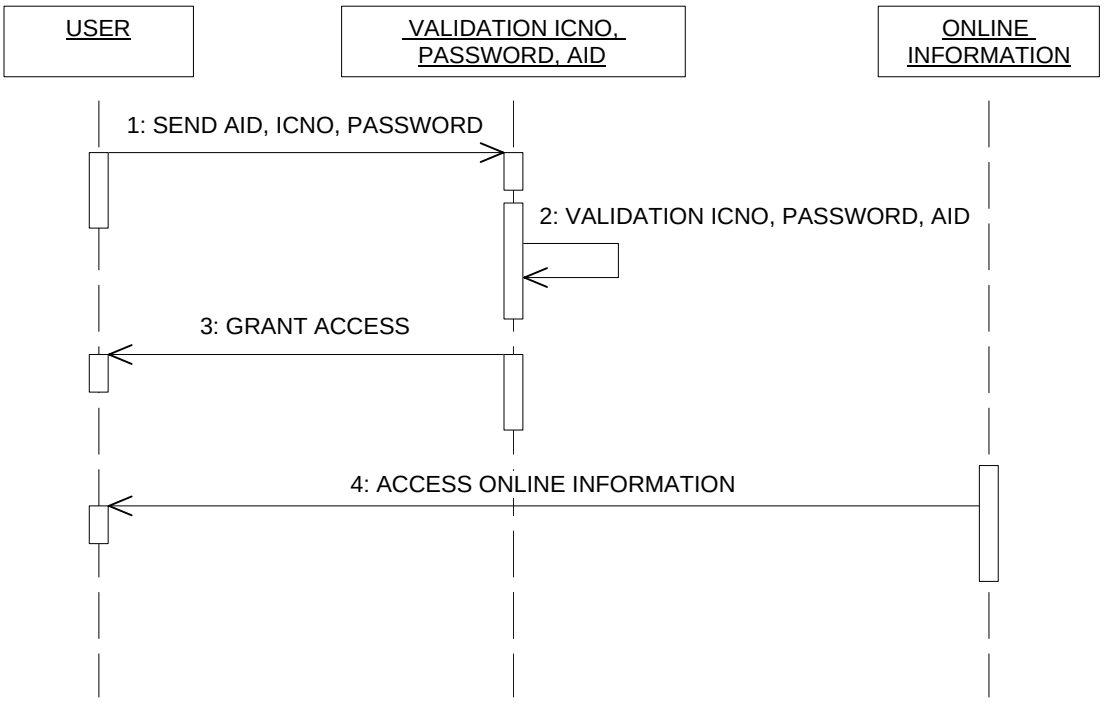


Figure 4.3 : Use Case And Sequence Diagram For Ubiquitous Database Access

User is prompted to choose a correct card and input password. Password is sent for validation. User just have three times to enter correct password, else, the card will be blocked to prevent Bruce Force attack. Once the correct password is validated, user is granted the access to retrieve data from the ubiquitous database base on access control level. Admin have all the authorities to add, modify, save, view, change card, and access to distributed database. Doctor can't add user, anyway, doctor can modify the database, as well as access all the functions. Nurse just can view patient record and her record. Nurse can't modify the data. Normal user is limited to view his record only.

4.5.2 Use Case and Sequence Diagram for Online Distributed Database Access





User is prompted to enter password and choose correct card. Tries Error Limit Control and Access Control Level are deployed for security protection. Different status of user will have different priority in access web site. For instance, admin and doctor are able to modify data on web, but not for nurse and normal user.

4.5.3 Use Case and Sequence Diagram for Card Termination

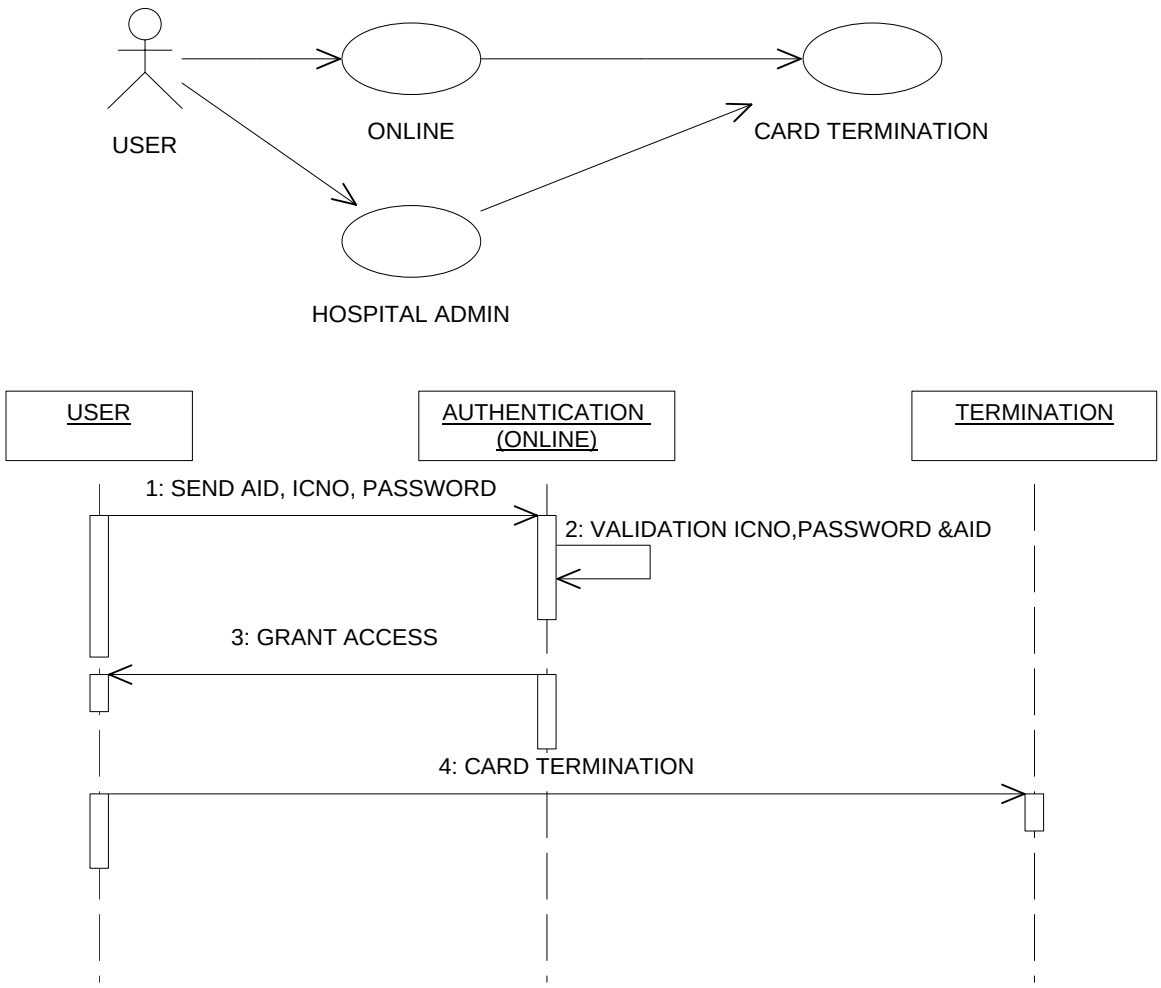


Figure 4.5: Use Case and Sequence Diagram For Card Termination

User has two options to terminate the card. First option is terminating the card online. user can terminate his card anytime by entering user Identity number and

password. This is convenient for the user especially when Admin is out of duty . Otherwise, user can choose second option to inform Admin, then admin will terminate the card. Once the card is terminated, lost count in the database is added by one. So, during the validation to access online data, the lost count number of the smart card and the database is not matched. Thus, user is denied to access the online database.

4.5.4 Use Case and Sequence Diagram to Change Password

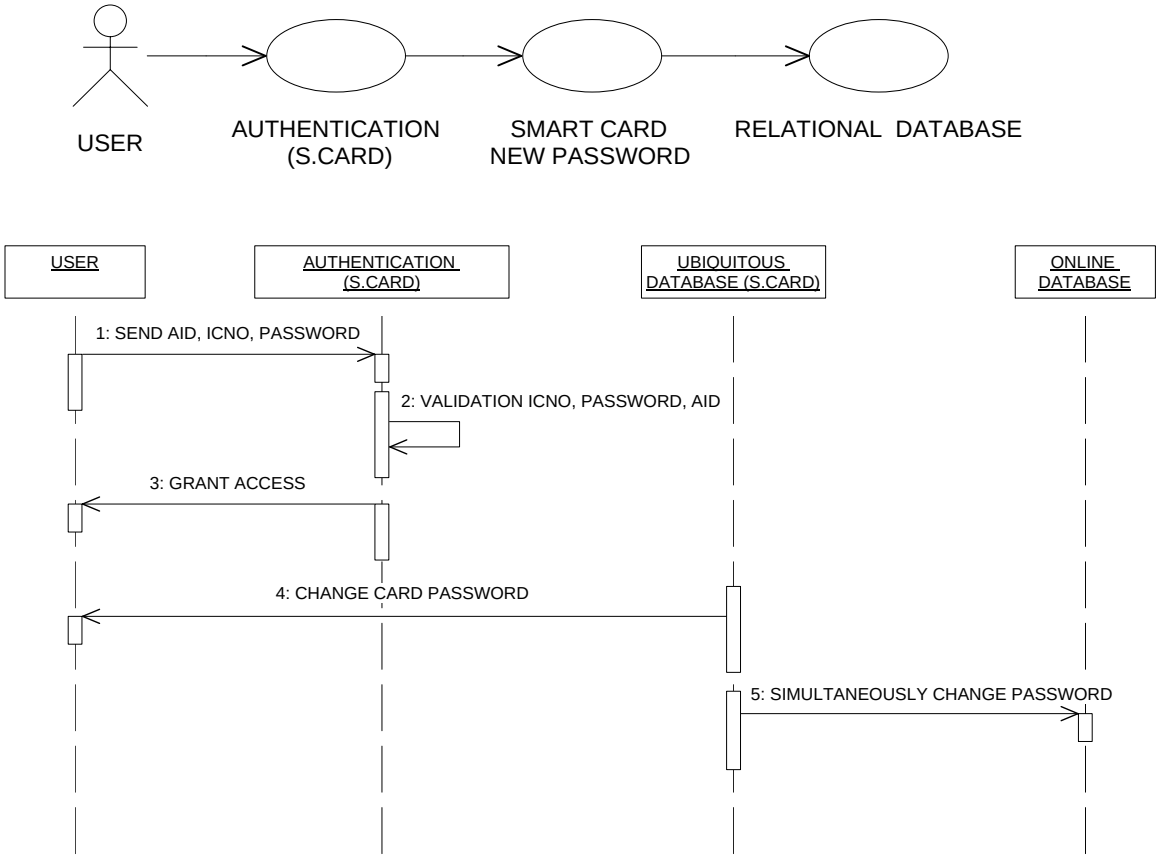


Figure 4.6 : Use Case And Sequence Diagram To Change Password

Every status of user have the ability to change their password once they successful login to the ubiquitous database system. User is prompted to enter new password and re-confirmed the password before the password is updated.

IMPLEMENTATION

5.0 Introduction

This chapter will discuss the detailed implementation of the UDMHC system, the specification of the module, and some technologies used to develop this system. The implementation of this system involves two phases; they are implementing smart card as ubiquitous database and accessing distributed database via HTTP protocol.

5.1 Phase 1 – Smart Card Implementation

This phase discusses on functions and features of UDMHC.

5.1.1 Card Grouping

Generally, the card can be grouped into 4 categories, which are admin, doctor, nurse and normal user. Admin has authority to access all the functions, including add user, change card, unblock card, modify data and create new member. Doctor has privilege to modify user's information, but has no authority to do the admin part, such as unblock card and create new member. Nurse is allowed to view patient's information, but can't modify the data. Normal user, or the patient, can only view his own information and have no right to change it. Please refer to Table 4.1 for details of access level for each card category. Each category of card holders has different Application Identifier (AID). Normal user is assigned with AID = {0xCC, 0x11,0x22,0x33,0x44,0x55},

Doctor with AID= {0xDD,0x11,0x22,0x33,0x44,0x55};

Nurse with AID = {0xBB,0x11,0x22,0x33,0x44,0x55};

Admin with AID = {0xAD,0x11,0x22,0x33,0x44,0x55}. So, once the user chooses his card, as shown on Figure 5, the card validation checking is done to prevent user from using wrong card.

```
/*-------------------------------------------------------------------------*/
private boolean bConnected = false;
private short UserAID[]= {0xCC, 0x11,0x22,0x33,0x44,0x55};
private short DoctorAID[] = {0xDD,0x11,0x22,0x33,0x44,0x55};
private short NurseAID[] = {0xBB,0x11,0x22,0x33,0x44,0x55};
private short ADMINAID[] = {0xAD,0x11,0x22,0x33,0x44,0x55};
/*-------------------------------------------------------------------------*/
```

Figure 5.1: Code of Assign AID to Each Group Of User

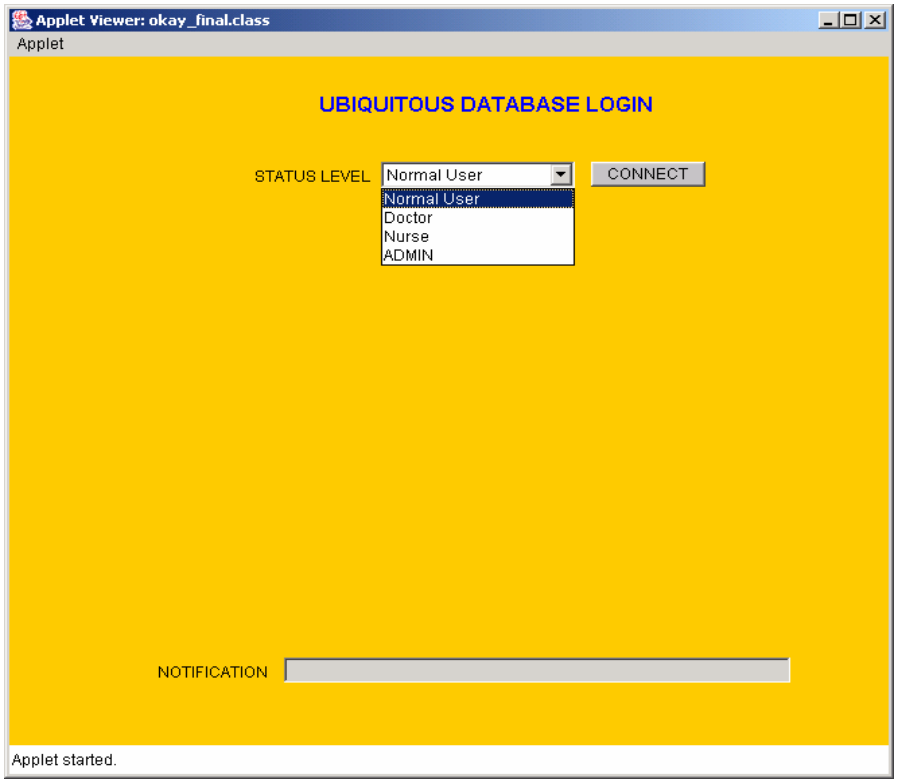


Figure 5.2: Each Group Of User Have Different AID

5.1.2 Validation and Password

In order to access the database, user needs to input password. If the input matches the key in the smart card, then the permission is granted. Else, he will be prompted to re-insert the password. The user only has three times to key in valid password, or the card will be blocked. Refer to the program below; the count of getTriesRemaining is reduced by 1 each time user key in wrong password. Once the card is blocked, it has to be sent to admin to unblock it; else, the card can't be used anymore. As shown in Figure 5.3, error is prompted for invalid password.

```
/*-------------------------------------------------------------*/

   // maximum number of incorrect tries before the PIN is blocked
   final static byte PinTryLimit =(byte)0x03;

   //validate the password
   private void validate(APDU apdu) {
         byte buffer[] = apdu.getBuffer();

         byte byteRead = (byte)(apdu.setIncomingAndReceive());

         if (!passwd.check(buffer, ISO7816.OFFSET_CDATA, byteRead))
               if (passwd.getTriesRemaining() > 0)
                           ISOException.throwIt(SW_WRONG_PIN);
               else
               {
                     ISOException.throwIt(SW_KEY_BLOCKED);
               }
   }

   /*-------------------------------------------------------------*/
```

Figure 5.3: Code of Password Validation

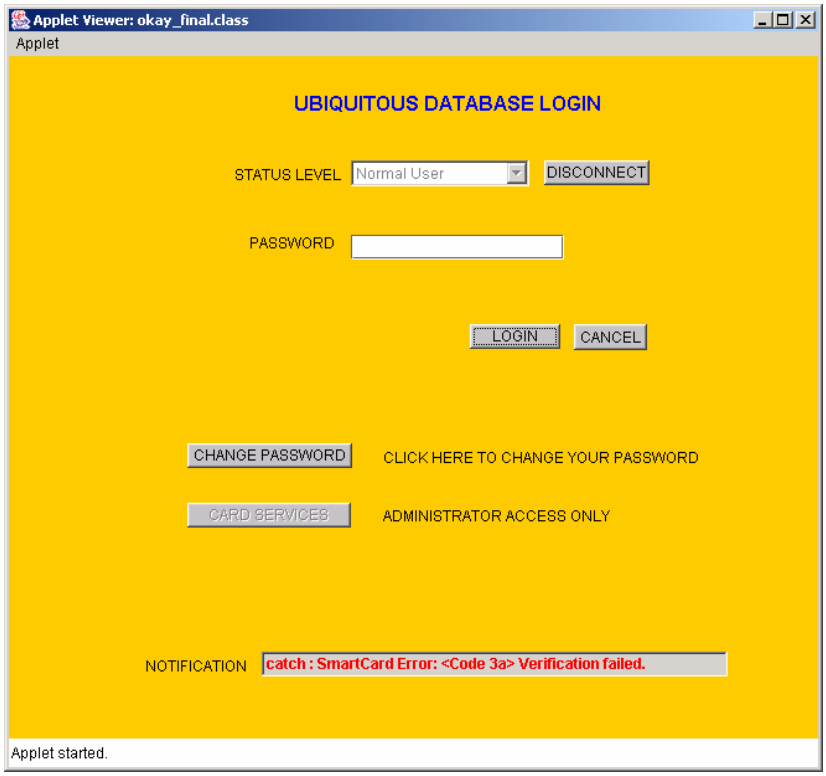


Figure 5.4: Error is Prompted for Wrong Password

5.1.3 Changing Password

User can change his password once he has successfully login to the system. User is prompted to enter new password and re-confirm the password. Once user clicks ok, the old password will be replaced. If the new password and the re-confirm password do not match, user needs to re-enter the password.

```
/*-------------------------------------------------------------*/
change the password
      private void ChangePasswd(APDU apdu) {
      byte buffer[] = apdu.getBuffer();
      apdu.setIncomingAndReceive();
      passwd.update(buffer, (short)(ISO7816.OFFSET_CDATA), MaxPinSize);
      }
/*-------------------------------------------------------------*/
```

Figure 5.5: Code of Password Changing

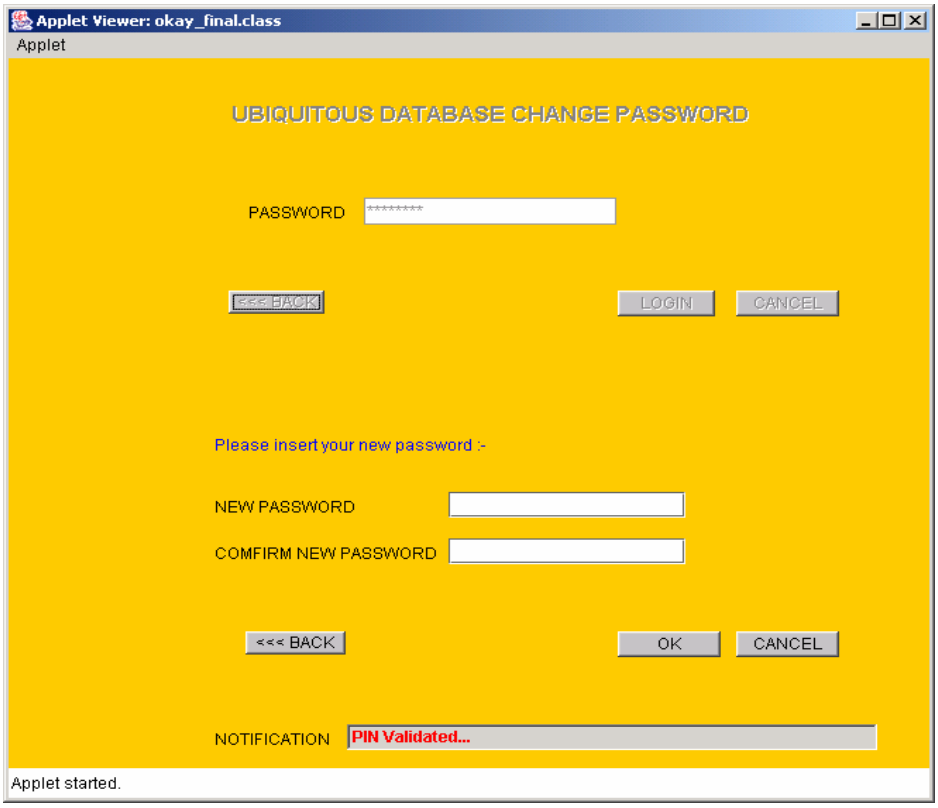


Figure 5.6: Page Change Password

5.1.4 Card Lost Setting

This function is valid for admin only. Once the card is lost, admin will assign a new card to the user and add 1 to the previous lost count number. This lost count number is used when user want to access online information. Lost count number from the smart card will be checked whether it is matched with lost count number of the database. If matched, the card is valid, else, the card is terminated to prevent from being used again. In addition, user can set the lost count in the online database once his card is lost.

```
/*-------------------------------------------------------------------------*/
      //save lost count information to card
      private void setLostCount(APDU apdu) {

            //if(!passwd.isValidated())
            //   ISOException.throwIt(SW_PIN_VERIFICATION_REQUIRED);

            byte buffer[] = apdu.getBuffer();
            byte size = (byte)(apdu.setIncomingAndReceive());
            byte index;
            //  Store the length of the string and the string itself
            lostCount[0] = size;

            for (index = 0; index < size; index++)
            lostCount[(byte)(index + 1)] =
            buffer[(byte)(ISO7816.OFFSET_CDATA + index)];
            return;
      }
/*-------------------------------------------------------------------------*/
```

Figure 5.7: Code To Set Card Lost Number

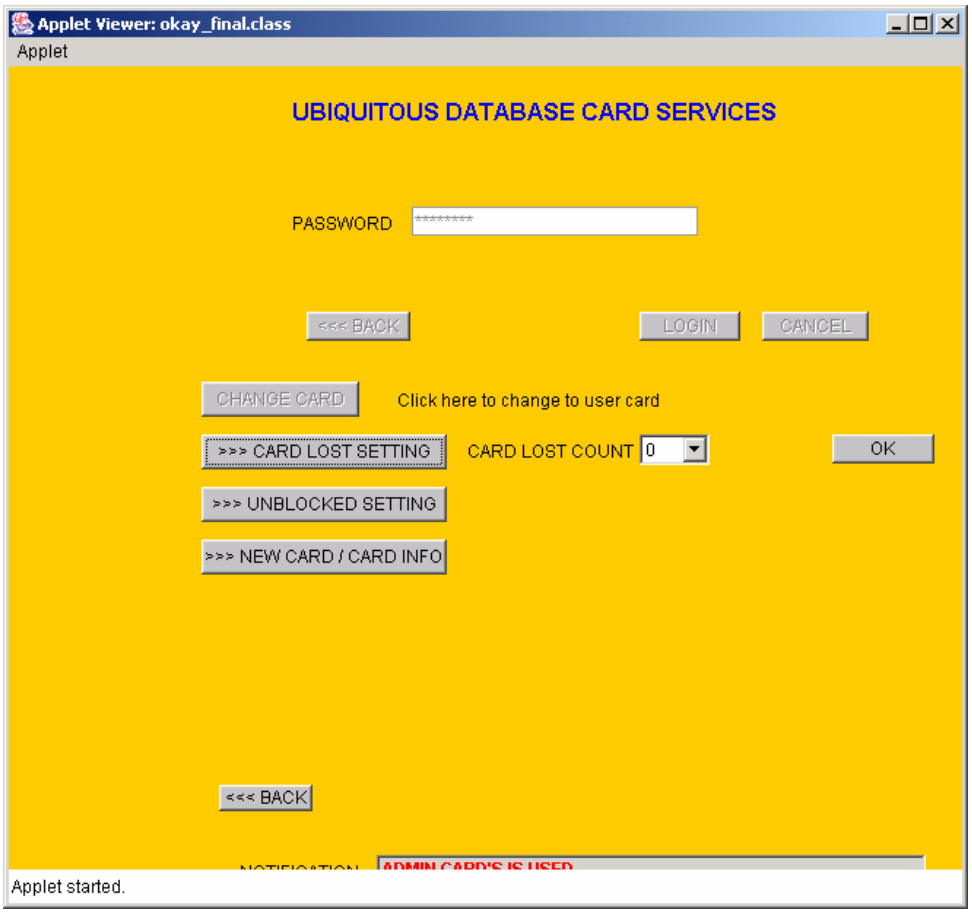


Figure 5.8: Lost Count Setting

5.1.5 Unblocked Setting

This function is valid for Admin only. Admin can unblock the card and set new password to the card user. The code resetAndUnblock() will reset the blocked pin. The blocked card will be replaced by new password.

```
/*-------------------------------------------------------------*/
 //unblock password
private void Unblock(APDU apdu) {
byte buffer[] = apdu.getBuffer();
passwd.resetAndUnblock();
apdu.setIncomingAndReceive();
passwd.update(buffer, (short)(ISO7816.OFFSET_CDATA),MaxPinSize );
}
/*-------------------------------------------------------------*/
```

Figure 5.9: Unblock Card Function

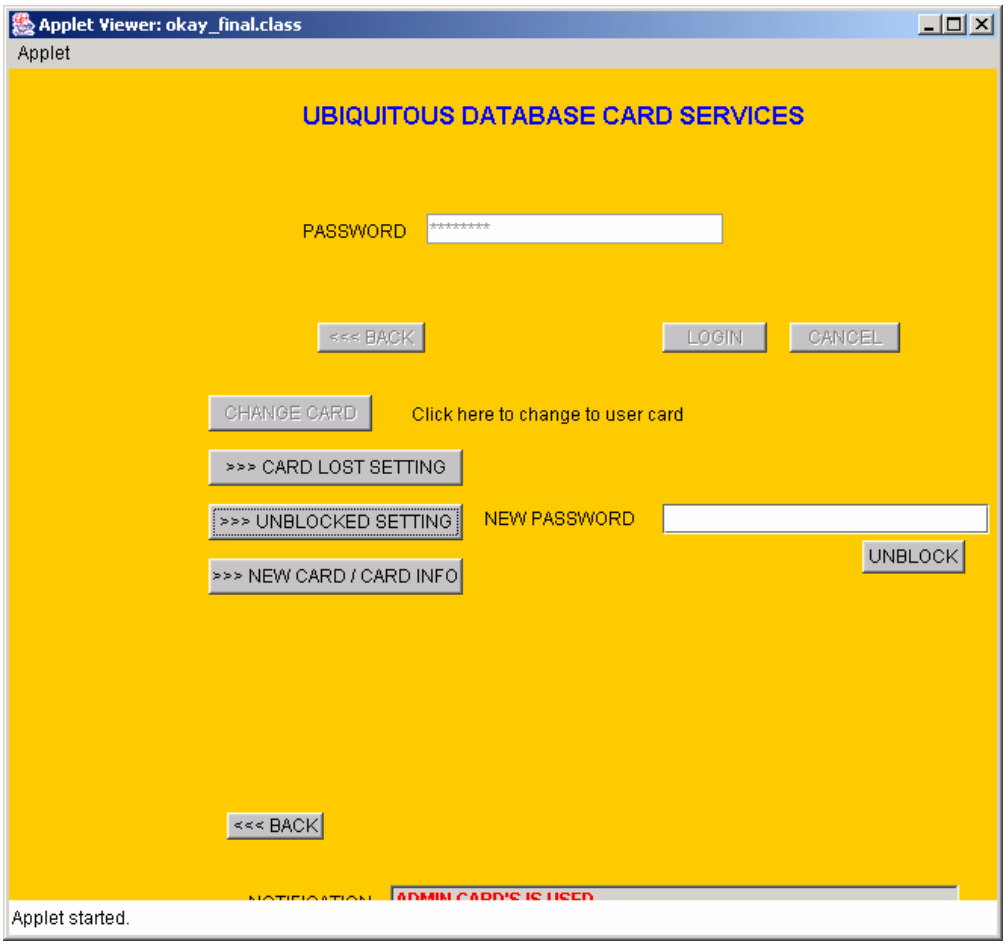


Figure 5.10: Page Unblock Card

5.1.6 Create New Card

This function is valid for admin only. Admin will assign identity number to the new user and choose which hospital to be registered at. The domain and the database type are automatically assigned when the hospital is chosen. The hospital ID and domain represent which database the user record is saved. Besides that, admin can get the info of the card by clicking the get info button.

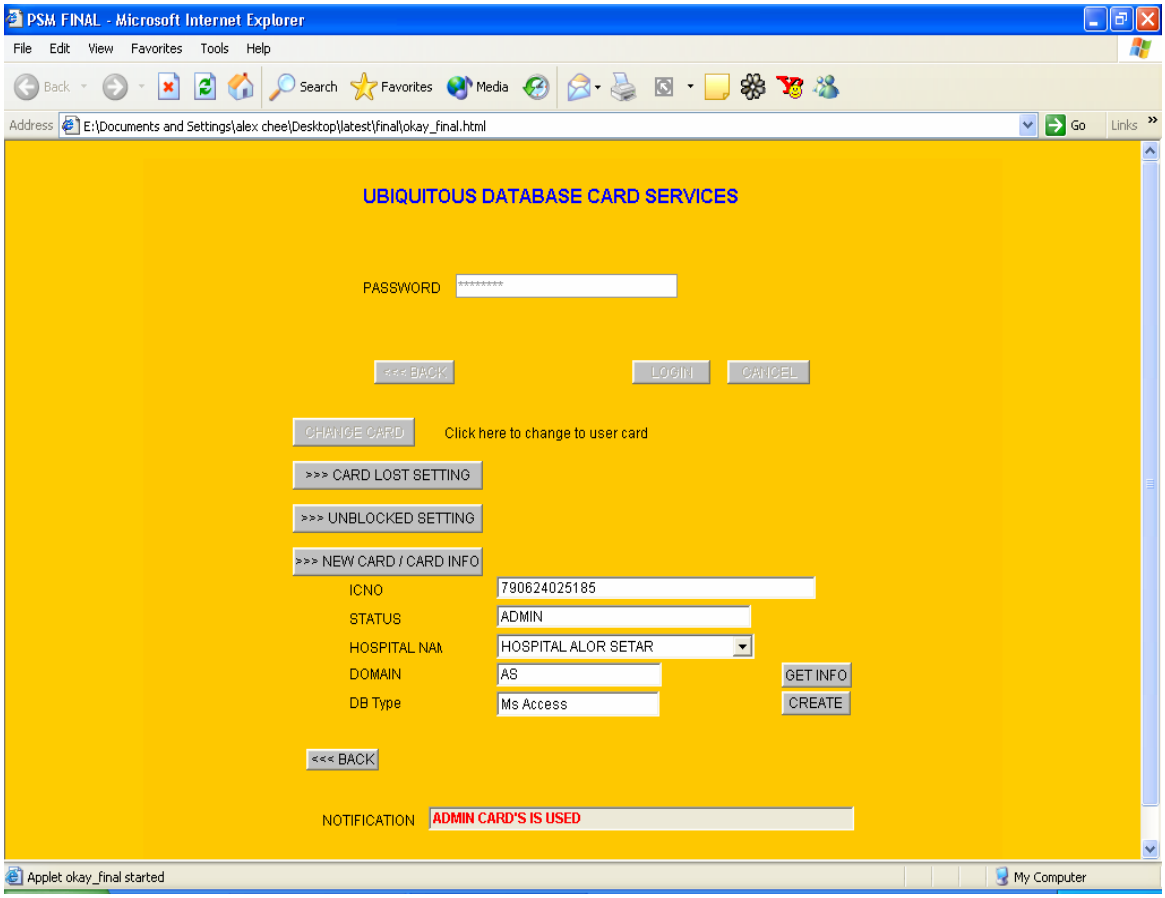


Figure 5.11: Page New Card Creation

5.1.7 Change Card

This function is valid for admin, doctor and nurse. In order to retrieve the patient information without knowing the password of the patient, these authorized users can click on change card button; follow by inserting cards belonging to other parties. For example, doctor can view patient's information by inserting the patient's card. Admin can access the card from all categories. A message box is prompted to let admin choose which categories of card to be retrieved.

```
/*-------------------------------------------------------------------*/
//change medical button
void btnChange_ActionPerformed(java.awt.event.ActionEvent event)
{
      int choice = JOptionPane.showConfirmDialog(null, "CHANGE CARD ?",
      "CHANGE CARD", JOptionPane.OK_CANCEL_OPTION);
            switch(choice) {
                  case JOptionPane.OK_OPTION:
                   short AID[]= {0xCC, 0x11,0x22,0x33,0x44,0x55};
                    if(connect_new(AID)){
                         txtstatusLevel.setText("NORMAL USER");
                         getPersonal();
                         getMedical();
                         getContact();
                         txtInfoNotify.setText("NEW CARD HAS BEEN
CHANGED");
                    }
                    else{
                         txtInfoNotify.setText("NEW CARD FAIL TO BE
CHANGED");
                    }
                    break;
                  case JOptionPane.CANCEL_OPTION:
                   break;
            }
}
/*-------------------------------------------------------------------*/
```

Figure 5.12: Change Card Function

Figure 5.13: Change Card Message Box For Doctor And Nurse

Figure 5.14: Change Card Message Box For Admin

5.1.8 Personal Information

This is the page for retrieving basic personal data from ubiquitous database. Different categories of user have different access authority of the page. For instance, admin can add, modify, change card for this page. Anyway, doctor is not allow to add, doctor is allowed to access function of modify and change card. In other word, doctor can change the information of the user or patient, but can not add new user. Nurse only has authority to retrieve patient data from patient card, but is not allowed to modify the data. Normal user can only see his information, but can't do any amendment. Button advance enable user to retrieve detailed information online. Figure 5.13 shows the page of personal information.

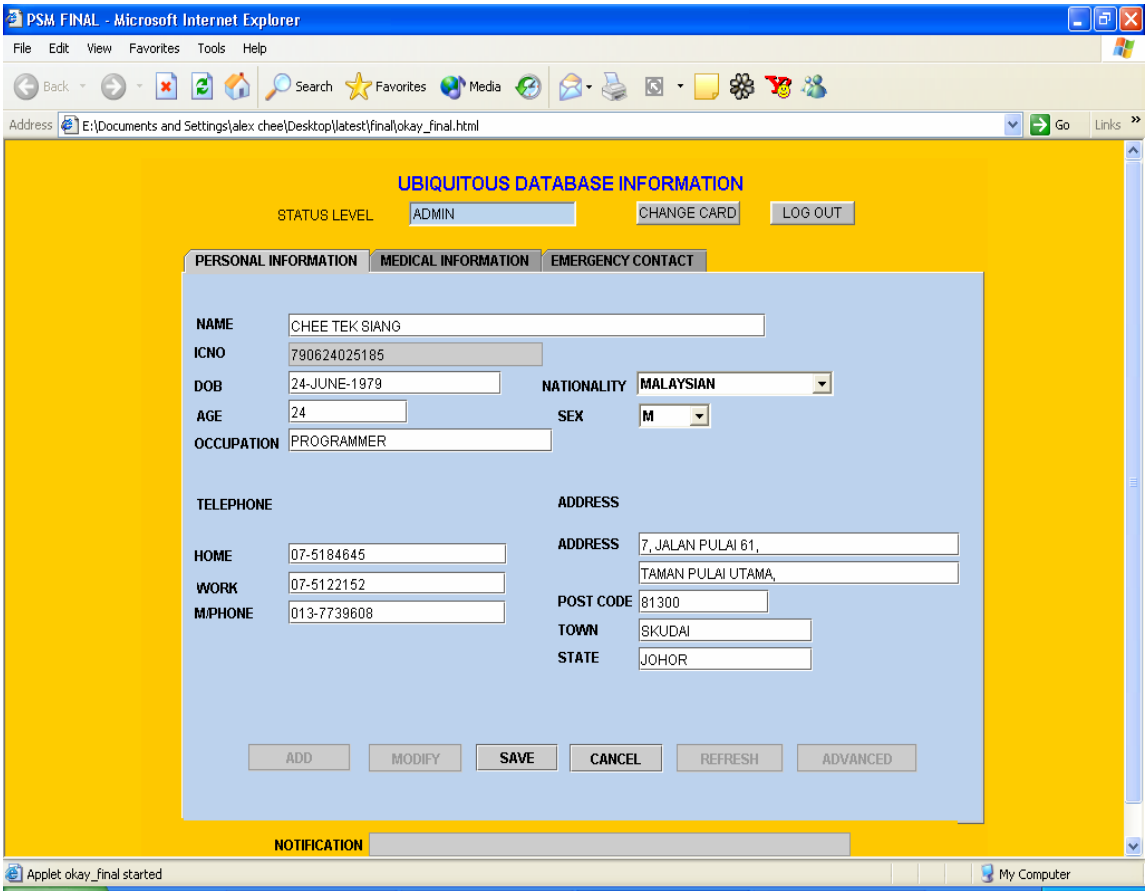


Figure 5.15: Page Of Basic Personal Information

5.1.9 Medical Information

This is the page for retrieving basic medical information from ubiquitous database. Different categories of users have different access authority of the page. For instance, admin can add, modify, change card for this page. Anyway, doctor is not allow to add, doctor is allowed to access function of modify and change card. In other word, doctor can change the information of the user or patient, but can not add new user. Nurse only has authority to retrieve patient data from patient card, but is not allowed to modify the data. Normal user can only see his information, but can't do any amendment. Button advance enable user to retrieve detail information online. Figure 5.14 shows the page of medical information. Hospital ID is automatically assigned to the user during the

registration. Last update time is automatically updated when the information is modified by doctor to indicate the latest treatment date.

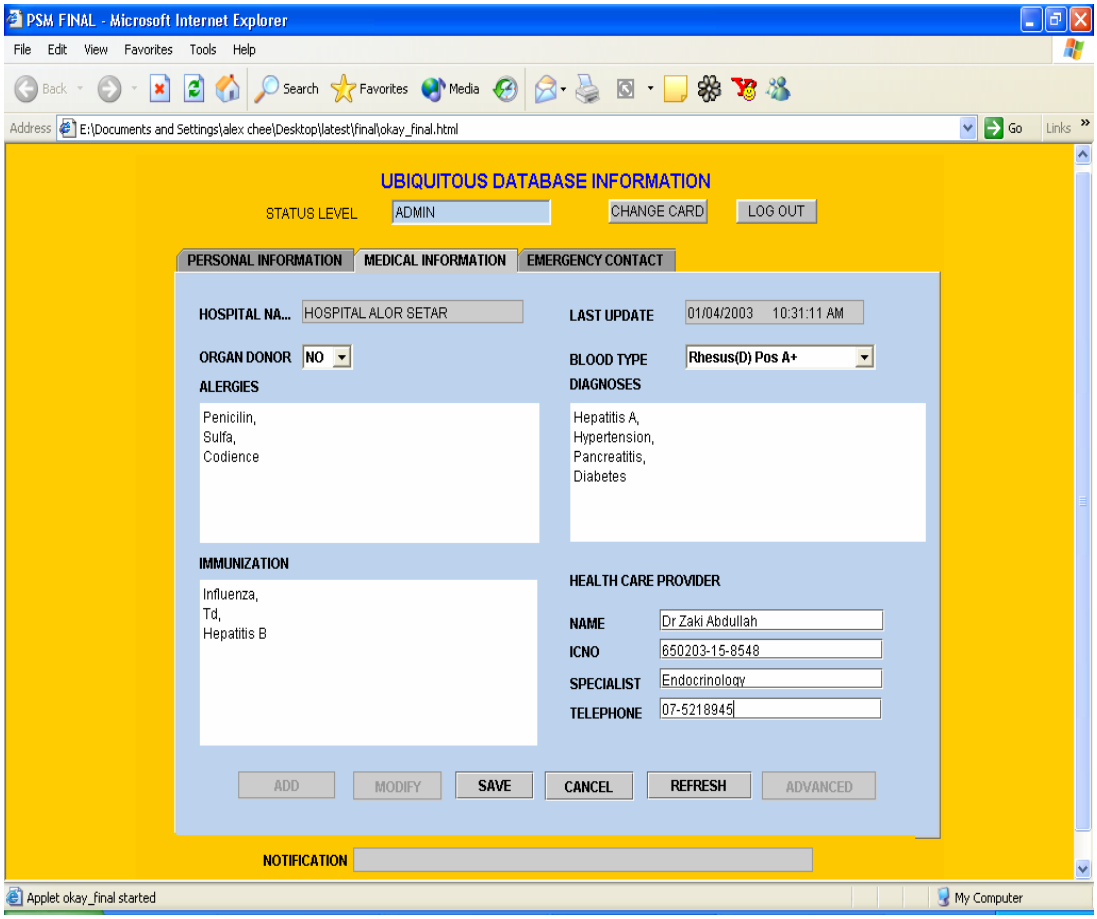


Figure 5.16: Page of Basic Medical Information

5.1.10 Emergency Contact Information

This is the page for retrieving emergency contact person's information from ubiquitous database. Button advance enable user to retrieve detail information online. Figure 5.15 shows the page of emergency contact information.

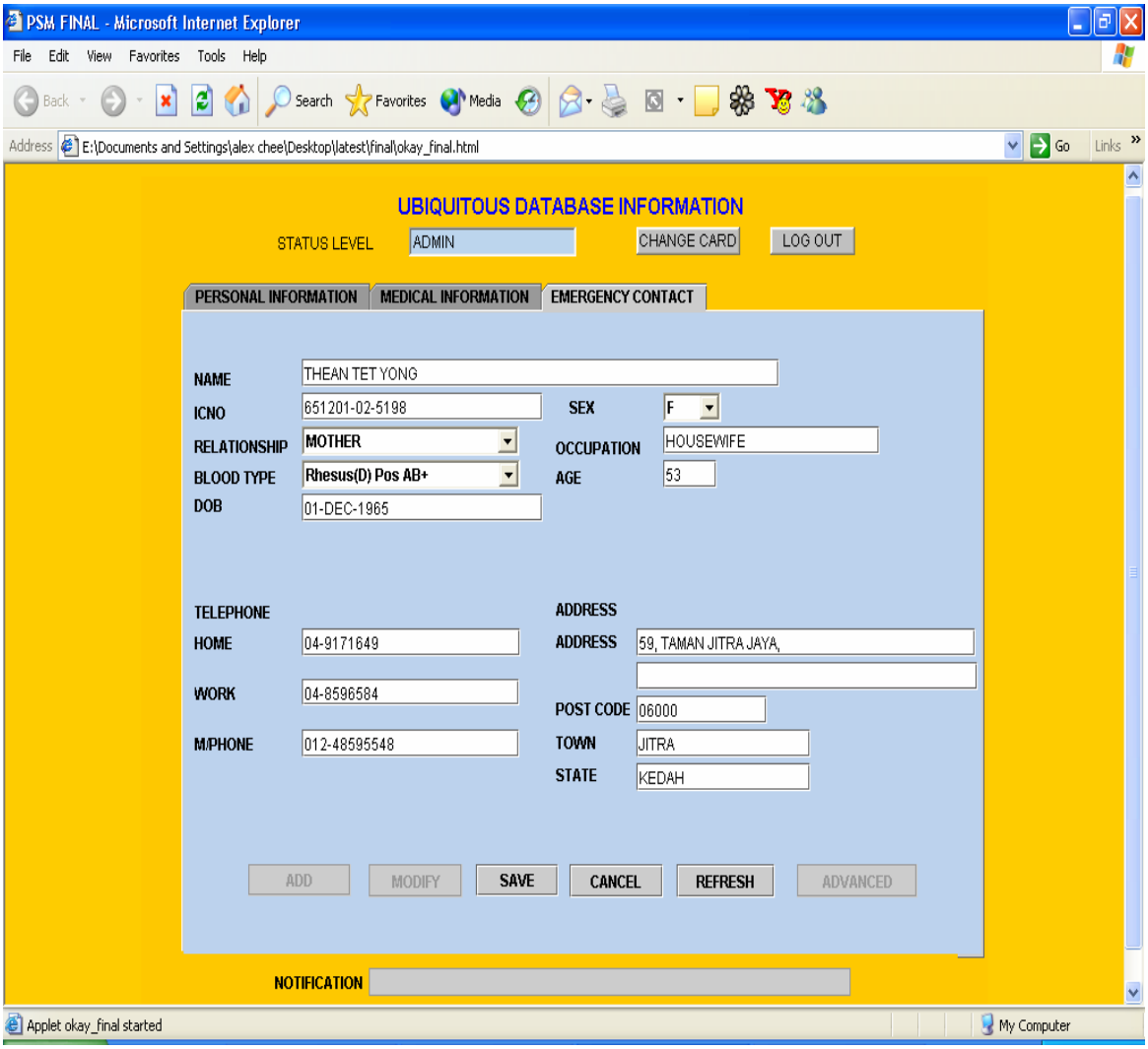


Figure 5.17: Page of Emergency Contact

5.2 Phase II – Web Based Implementation

This phase discuss on implementation of database in web.

5.2.1 Advance Personal Information

User can view detailed information via web. This information is retrieved from distributed hospital where the user registered. Model View Design pattern is used as the solution to remove out.println from the servlet and remove java code for data processing at Java Server Pages.

Figure 5.18: Advance Personal Information

5.2.2 Advance Medical Information

User is able to see the details of his information, such as the X-Ray and full medical check-up information.

5.2.3 Discussion Room

All users are allowed to join the discussion room. It is convenient for the medical-related professionals to share the medical knowledge, as well as for the normal user to seek advice from the medical-related professional.

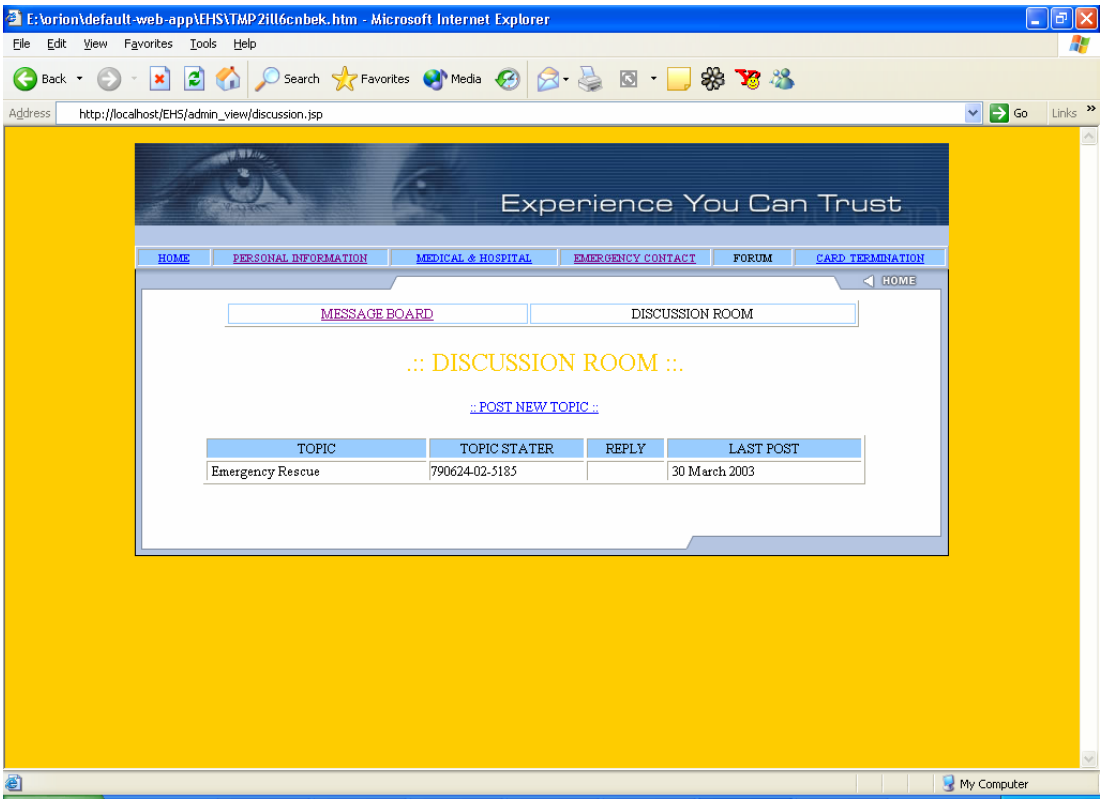


Figure 5.19: Discussion Room

5.3 Test Case

For verification and validation of the whole system's functionalities, some tests were performed. The three stages of testing have the following purposes:

i. Unit test – the smallest testable elements of the system were tested individually, typically at the same time those elements were implemented. Each module was compiled and executed for unit test.

ii. Integration test – the integrated modules are tested, such as add new card subsystem.

iii. System test – The complete application and system (one or more modules) were tested. The whole system was tested for the functionality and integrity.

5.3.1 Invalid Card Test

There are four categories of user for UDMHC system. Each category of users is assigned a unique Application Identifier. So, if the user insert invalid card, error message will be prompted.

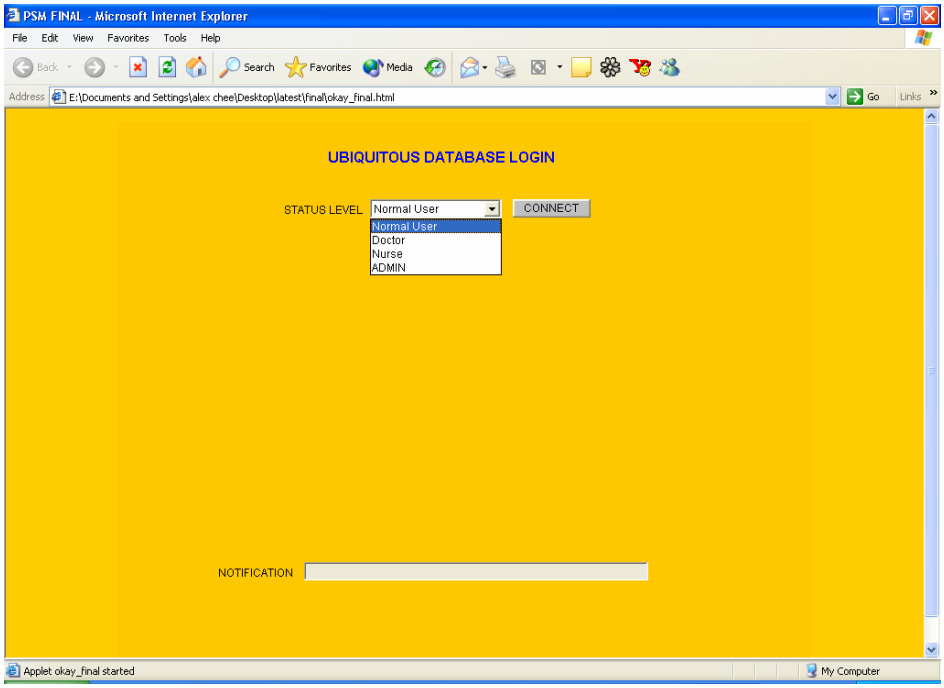


Figure 5.20: Four Categories of Card User

As shown in Figure 5.21, the user insert his card, but he chooses the categories of Doctor's card. So, error prompted and prohibited him from entering password.

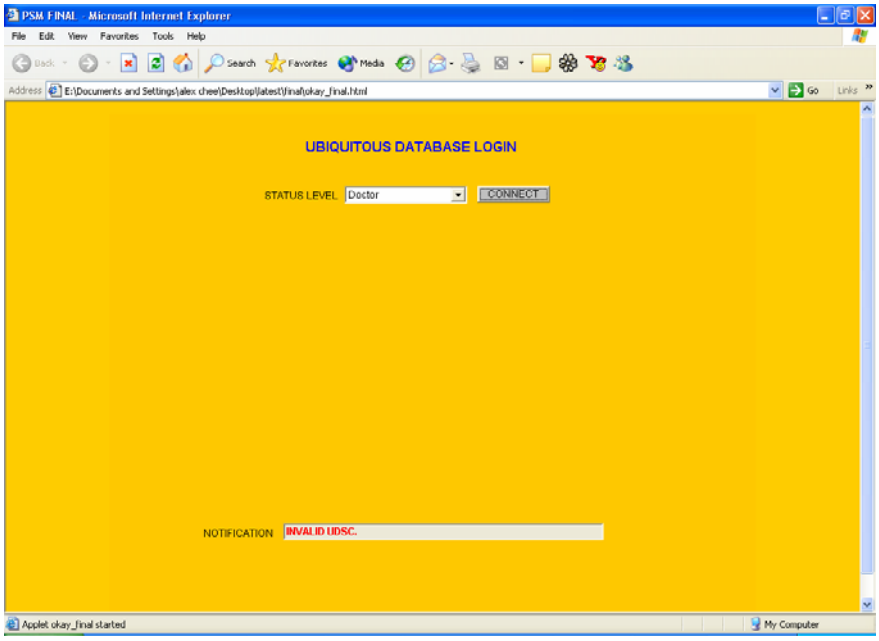


Figure 5.21: Notification Indicate Invalid UDMHC

As shown in Figure 5.22, if user insert the correct card, then he will be prompted the login page. He has options to login or to change his password.

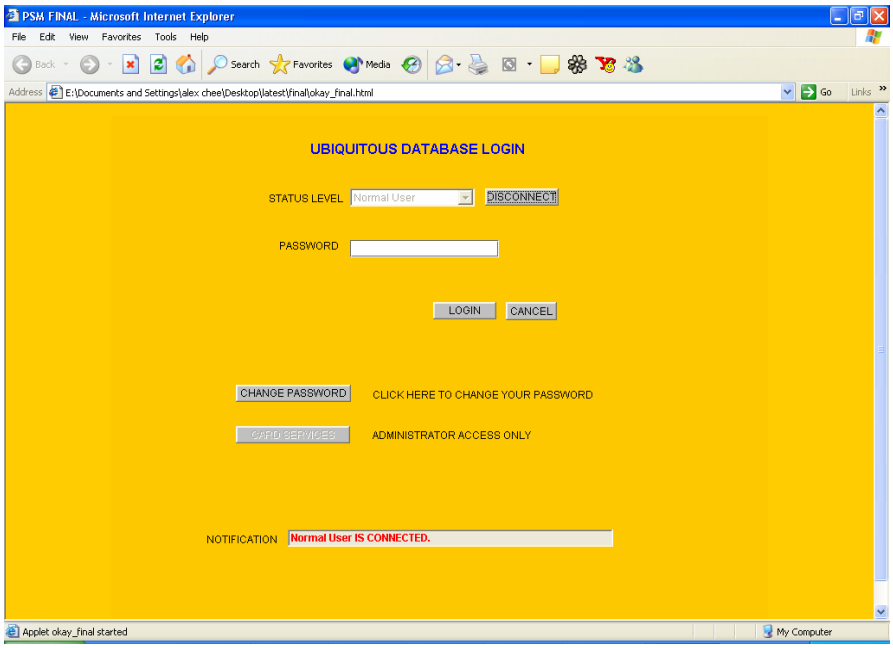


Figure 5.22: Correct Card Inserted

5.3.2 Invalid Password

Error message will be prompted for invalid password key in. Anyway, if user fails to key in correct password for 3 times tries, his card is blocked to prevent Bruce Force attacks. As shown in Figure 5.23, notification indicates the wrong password be inserted and verification fail.

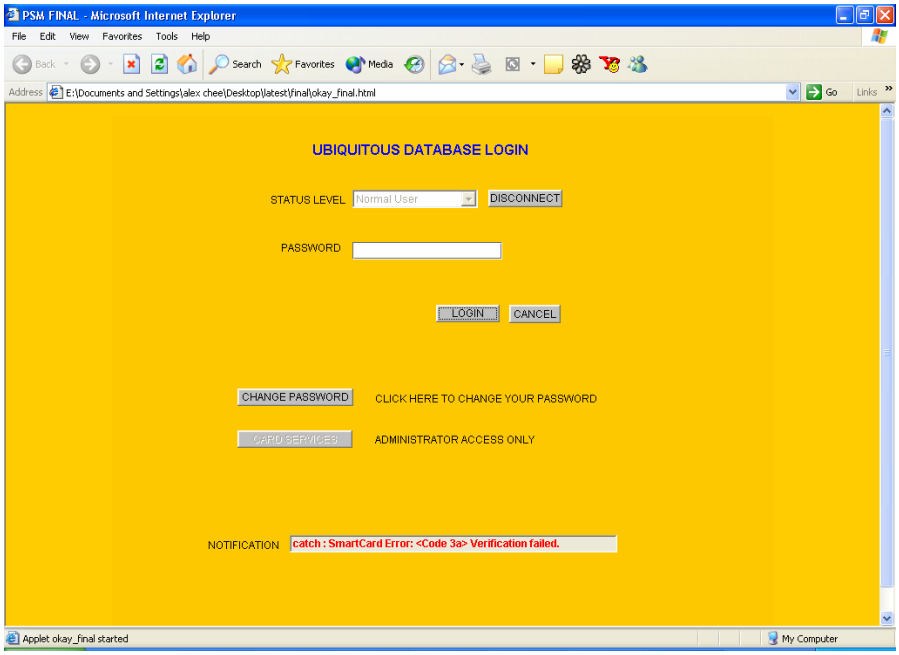


Figure 5.23: Notification Indicate Verification Failed

As shown in Figure 5.24, the card is blocked if the user fails to key in correct password for three time's trial. This is to prevent the Brute Force attacked.

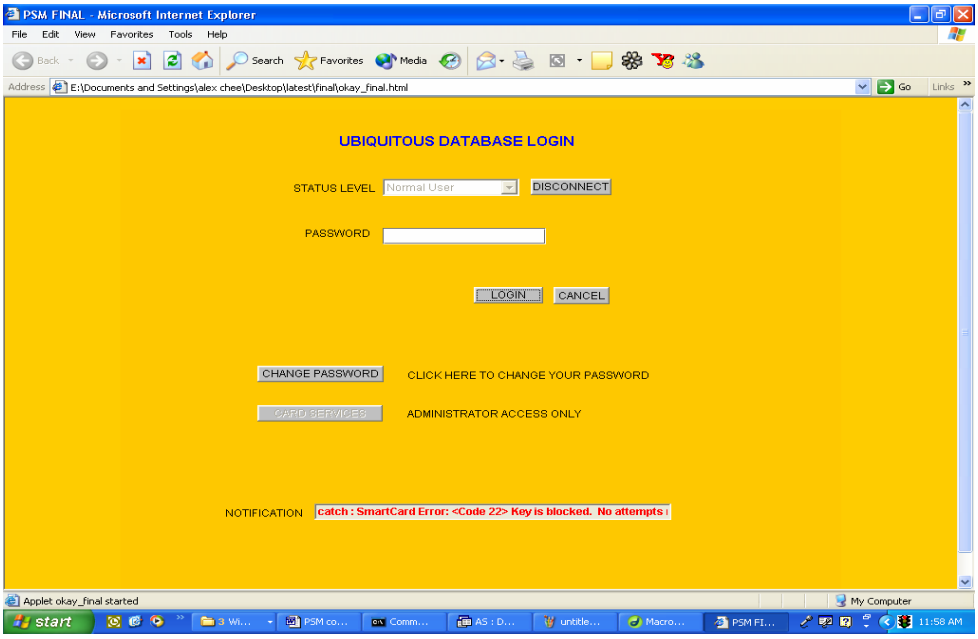


Figure 5.24: Card Blocked for Three Times Password Fail

5.3.3 Change Password Option

Every body has the right to change his password. Anyway, the user needs to verify his old password before change his new password. User need to insert new password twice in order to make sure he remember his new password. As shown in Figure 5.25, user will be prompted to reinsert new password if the both new password do not match.

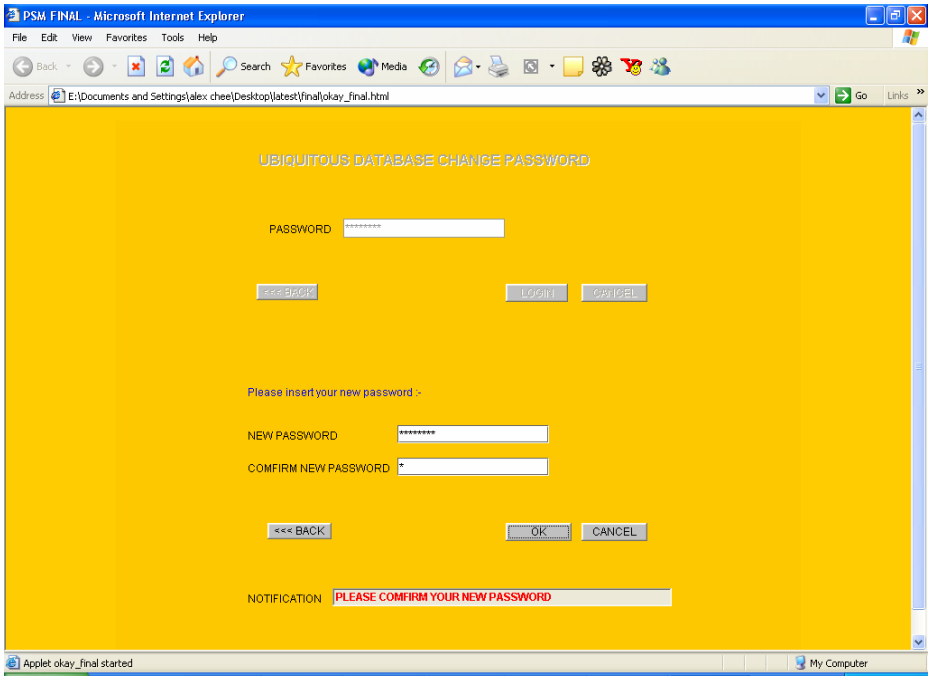
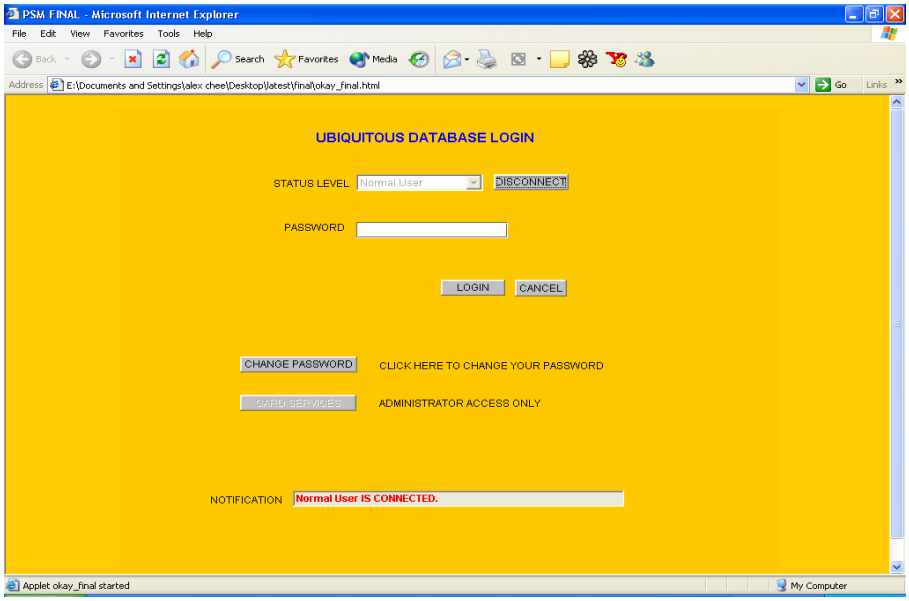


Figure 5.25: Notification for Both New Password Mismatch

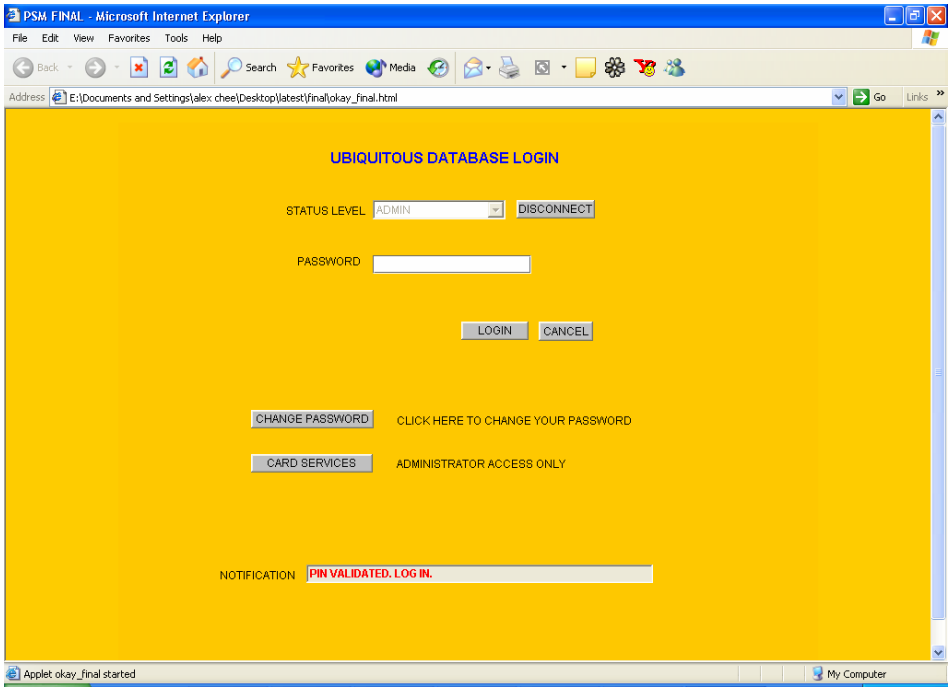
5.3.4 Card Services Option

Card Services contains functions of card lost setting, pin unblocked setting and new card creation setting. Only Admin have the right to access card service button.

As shown in the Figure 5.26 and Figure 5.27, only Admin have the right to access the card services button. Other are prohibited to access the card services option. Figure 5.26 shows that card services button is disabled for user's categories.



5.26: Card Services Button Disabled To User



5.27: Card Service Button Enabled To Admin

5.3.5 Change Card Option

Once Admin enter the card services function, he can exchange his card to other category's card. A prompt invoked to ask Admin insert new card. If Admin insert card mismatch with the categories of card, error prompt invoked and Admin reinsert or re-choose the correct categories. As shown in Figure 5.28, wrong card inserted and Admin prompted to insert new card.

5.28: Invalid Card Inserted

5.3.6 New Card Creation

As shown in the Figure 5.29, the status of the card will be automatically inserted to the card categories column. Admin can choose which hospital to register for the user. After inserting the user's identity number, Admin able to create new card by click on button Create Card. Identity number will be automatically loaded to the user's personal page, whereas hospital name will be loaded to user's medical information.

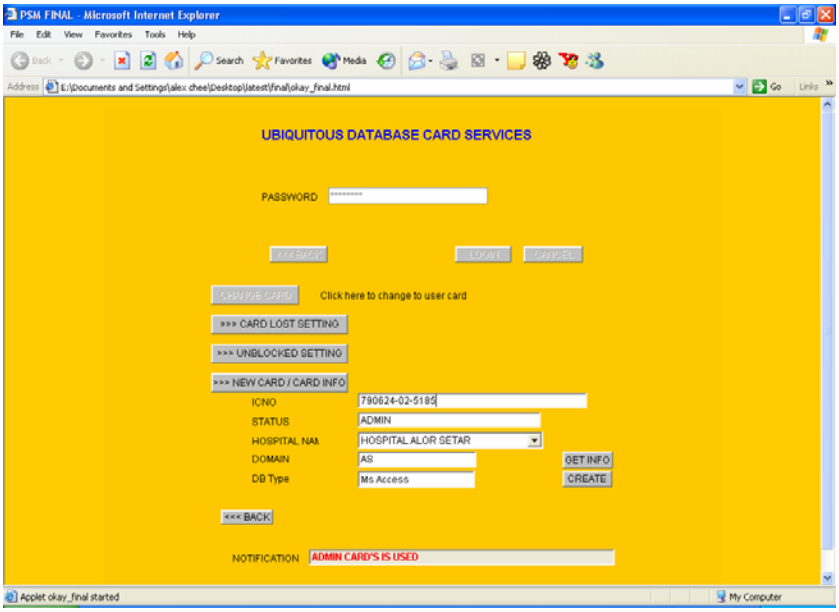


Figure 5.29: New Card Creation

5.3.7 Access Authorities

Admin have permission to access all the functions of the buttons. As shown in the Figure 5:30, all button are enabled to Admin

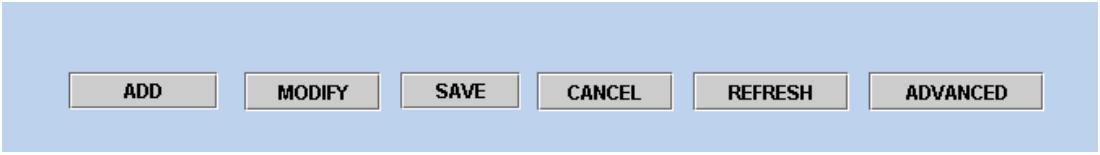


Figure 5:30: All Button Accessible to Admin

Doctor does not have permission to add new user.

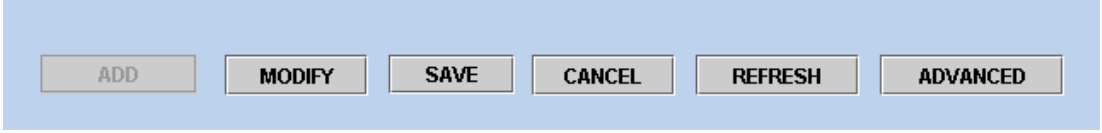


Figure 5:31: ADD Button Disabled To Admin

Nurse and Normal User can only refresh and access detail information via advance buttons.

Figure 5:32: Refresh And Advanced Button Enabled to Nurse and Normal User

5.3.8 Link to Web Database

Based on user's identity number, details information can be accessed by clicking on Advanced button. Figure 5:33 show the basic information in the smart card whereas 5:34 shows the details information in the web.

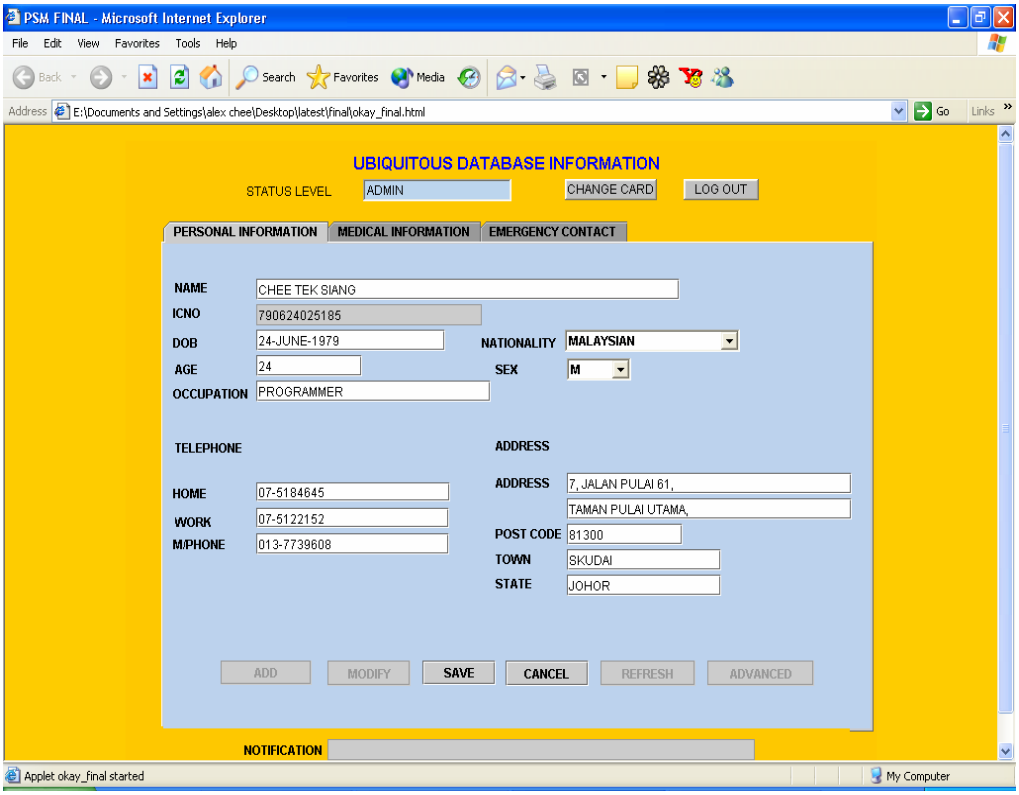


Figure 5:33: Basic Personal Information From Smart Card

Figure 5:34: Detail Personal Information In Web Browser

5.4 Conclusion

All the tests have been done to check the integrity of the system.

CONCLUSION

6.0 Introduction

UDMHC implementation currently is focused on the marriage between smart card and Internet to provide ubiquitous access to health information. Distributed database is implemented since the medical record of user is not centralized and distributed among hospital. The access control level method enables different party access medical information with different priority, hence ensuring program integrity.

6.1 The Advantage of the System

With this, some strengths of the system are :

i. **Accurate**, concise and ubiquity access victim's medical record in case of emergency. The pocket mobility of smart card, coupled with the ubiquitous software configurable web-browser, is promoting a truly mobile and interactive medical information technology.

ii. **Ubiquitous Medical Record Management**. This Ubiquitous Database in Mobile Health Care (UDMHC) is designed both offline and online accessing data from the smart card and Internet. As we call this UDMHC ubiquitous database is regarding its ability to store patient medical database. In this case, the standard web browser operates in an offline standalone mode such that access and updating procedure to smart card's medical record are facilitated by an active applet downloaded from within the card itself. The ability of

smart card to carry its own record management applet while being hosted by a commonly available web browser, presents a powerful paradigm to support a truly mobile and open environment. Importantly, it enables medical personnel to quickly gain access of vital patient's medical record without the need of a hospital or clinic to be equipped with so-called compatible information system. Moreover, UDMHC can easily be updated as new services are performed and new medications are prescribed so your card will always contain up-to-date, vital medical information.

iii. Flexible and Convenience

This system provides flexibility and convenience to the user in getting their medical information just by inserting their smart card in reader. Besides, it saves a lot of times for the physician to getting vital medical record of the user in the case of emergency.

iv. Simple and User Friendly

The user interface is very simple and user friendly. It looks like most website, provides that standardizations to the user. User can click to link to other page.

v. Platform Independent

Since this system is fully developed in Java, hence, it inherits the Java capability in function and can deploy to any platform of operating system.

6.2 Limitation of the System

The system limitation is :

i. Slow speed

Due to current technology constraint, the browser takes long time to load a Java Applet.

6.3 Conclusion

As a conclusion, the implement of Ubiquitous Database in Mobile Health Care is a potential solution to effectively and accurately manage patient’s ubiquitous medical database. UDMHC allows medical-related professional respond to patient's needs more effectively because pertinent information is immediately available when and where it is needed life and death.

In addition, UDMHC contributes better digital database management to medical center. This technology offers the benefits of mobility in pocket which can store useful and accurate medical records of the participating card holder. With the ability of open card, it allows us to develop end-to-end solutions using smart cards that are not bound to one platform, card, and application, make this UDMHC to be ubiquitous accessed anywhere with all compatible smart card reader.

With the deployment of distributed database, users at one site can access data stored at other sites. This is convenient to the user if he wants to retrieve his medical information anywhere. In distributed environment, it is much easier to handle expansion. New hospital can be added to the network without affecting the operations of other sites. This flexibility allows more hospital to expand relatively easily.

BIBLIOGRAPHY

[1] Kohane IS, Greenspun P, Fackler J, Cimino C. (1996) , Szolovits P. Building National Electronic Medical Record Systems via the World Wide Web. Journal of the American Medical Informatics Association. Pages 191-207.

[2] Fabian Ng and Chen Jen Tock. (19-23 March 96) "A Smart Card Medical System For The People With Disabilities," California State University Northbridge's 11th Annual International Conference, "Technology and Persons with Disabilities", Los Angeles

[3] Javasoft (2002) , “Java Card Introduction” . [Online] Available http://www.javasoft.com

[4] M. Weiser. (1991) , “The Computer for the 21st Century.*Scientific American*”. Pages 66-75.

[5] Peter Szolovits, (1995) "A Revolution in Electronic Medical Record Systems via the World Wide Web," International Conference on the Use of Internet and WWW for Telematics in Healthcare, Geneva, Switzerland

[6] Extensible Markup Language (XML) 1.0 (2002), World Wide Web Consortium Recommendation. [Online] Available http://www.w3.org/TR/REC.xml

[7] Lynda Radosevich, (25 Aug 1997) "Health Care uses XML for Records", InfoWorld, [Online] Available http://www.infoworld.com

[8] Health Smart Card (2001) “Hospital Emergency Room” [Online] Available http://www.healthsmartcard.net/pro-hosp.html