Medical Data Access with a PDA-based Client

BY

Dejene Hunde

Project Report

Submitted to School of Graduate Studies of Addis Ababa University in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

June 2005

Addis Ababa University School of Graduate Studies

Department of Computer Science

Medical Data Access with a PDA-based Client

BY

Dejene Hunde

Name and Signature of Members of the Examining Board

1. Dr. Solomon Atnafu, Advisor _____

2.

3.

Acknowledgements

First of all, I would like to forward my deepest appreciation and thanks to my advisor Dr. Solomon Atnafu for his motivate and constructive guidance throughout the work. Many thanks and appreciations go to him for the discussions with him always made me think that things are possible. His enthusiasm and encouragement has always inspired me to accelerate to the completion of the work.

I would also like to thank all my instructors at the Department of Computer Science, for their personal commitment and contribution to the success of the graduate program.

My thanks go to my friend and mate Samuel Dadi for he was tireless for giving morals and standing by me in all the difficult moments.

I also would like to extend my thanks to my peer Fistem Meshasha working on related thesis topic. The many times discussions and sharing of ideas and resources with him had a significant contribution to the success of this work.

Finally, I am grateful to my first teacher; my grandmother, who took the role of a father and a grandmother, who took the role of mother, to raise me; and my mother and all the rest of my families, friends, and peers who, one or the other way brought me up to a success in my academic endeavor.

Table of Contents

List of Figures

List of Tables

Abstract

In countries like Ethiopia, there is a very high shortage of medical specialists and doctors. This is a big problem that affects the quality of health care to patients. Due to inadequate treatment and guidance, which is caused by not effectively accessing medical specialists, a patient may suffer tremendously to the extent of losing life. A solution to this problem is to effectively utilize the existing or the available professionals by using modern technologies and by allowing professionals to have adequate access to patient data. An enabling technology in this respect is for example mobile access to patient medical data.

Personal Digital Assistants (PDAs) offer the medical specialists the ability to access the medical data from remote site. PDA-based client systems have already been implemented for medical application purpose in different places. These systems allow the medical specialists to access medical data from anywhere, and at anytime to give better treatment and guidance to patients. But these systems are not accessible to external users. In this project, we propose PDA-based client to access medical data from anywhere possible as wireless infrastructure may allow.

In this work, we explored the tools and technologies available for this purpose and developed a prototype for a PDA-Based client to access medical data from remote sites. We tested the prototype with palm operating system Emulator (POSE). The prototype enables a medical specialist to access the medical data of a patient so that he can give better treatment and guidance. We believe that this approach will tap the problem of the medical professionals shortage in Ethiopia.

Keywords: Mobile Medical Data Access, PDA-based Client, MAMI, POSE

Chapter One

1. Introduction

1.1 Medical Data Access

A medical professional requires having access to relevant patient data or medical data so that the patient can get adequate treatment and guidance. The medical data to be accessed should be correct, accurate, and on time so that medical errors, which are responsible for tremendous patient suffering, loss of life, and that costs billions of dollars, can be minimized. Different medical data management systems have been used in the last decades [5].

Computerized Medical data management system is one of the systems that used to store, access and manage medical patient data. In computerized medical data management system, Database Management system (DBMS) is used to store and to manage a medical patient data. In most of the systems that exist today, medical data are accessible only from fixed stations. That means the medical professionals cannot access the medical data from anywhere, and at any time to give adequate treatment and guidance for the patient. In order to allow the medical professionals to access medical data from anywhere, and at anytime, mobile information systems are very important [3].

Until recently, Mobile information systems are not common. But with the developing capabilities of mobile computing, it is possible to access medical data from anywhere, and at any time to effectively exploit the insufficient resource of medical specialists especially in countries like Ethiopia [3].

In Ethiopia, like other countries, there is scarce of medical doctors and specialists. In order to alleviate this problem, the uses of mobile medical information systems can play a great role. Thus, to access medical data wirelessly, devices like PDA are convenient tools [5].

1.2 Personal Digital Assistant and Medical Data Access

Personal Digital Assistant (PDA) has been developed for information management by combining the power of a user-friendly computer with the practicability of mobile handheld devices. Basic functions of a PDA include personal information management such as an electronic diary, address book, to-do-list, notepad, calculator, expense tracker and alarm clock. Furthermore, PDAs can be used as communication devices and as client devices to access data. Although PDA has similar function to desktop and laptop personal computers, it possesses several distinguishing, mostly favorable, characteristics. For instance, since PDAs utilize "flash memory", PDAs turn on instantly, and tend to crash less frequently. While in operation, they can be moved and handled with some facility, without fear of data corruption or mechanical breakdown. In addition, PDAs use a stylus on a touch-sensitive screen for text input and handwriting recognition, which tend to be much more practical and less cumbersome than the standard computer keyboard interface [3, 6, 7].

PDA operates with different software and it can also easily exchange programs and data with their PC complements. A process referred to as "hot-syncing" accomplishes data transfer routinely, which involves establishing a PDA-PC connection via a "cradle" device through a USB or serial port. Many PDAs can also wirelessly "beam" data to compatible PDAs and printers using infrared transceiver ports. Newer PDA models even have the capability to use compact flash, secure digital or memory sticks for expandable direct data storage and exchange [3].

Clinicians are becoming more accustomed to using portable technology such as PDAs for clinical use in daily care. Due to the small size of PDAs, their portability and ability to provide drug and clinical reference information at the point of care, PDAs are useful as an information tool at the bedside. PDAs have been referred to as the doctor's "New Black Bag"[1, 2,7]. PDA allows the clinicians to [1,2]:

- enter and access the medical data remotely,
- calculate appropriate drug doses,
- provide databases of important information, and
- offer other forms of bedside clinical decision support.

In recent studies, it is pointed out that PDAs can affect the efficiency and quality of the patient care. Thus, a computer stored patient record system linked to a client PDA would impart many potential benefits for both patient and clinician [1,2].

1.3 Problem Statement

There is a very high shortage of medical doctors and specialists in Ethiopia. This is a big problem that affects the quality of the health care of patients. A patient may suffer tremendously to the extent of losing of life due to this problem. This is because a patient cannot get adequate treatment and guidance.

To exploit the medical professionals in the country, there should be a means by which the medical specialists and doctors can give effective service. Empowering medical specialists to have access to medical data at anywhere, and anytime to give treatment and guidance to a patient is one modern approach that can help to alleviate the problem.

1.4 Objectives of the project

The general objective of this project is to develop a prototype of a PDA based client to access medical data so that the medical specialists and doctors can access mobile medical data from anywhere, any time to give treatment and guidance to the patient. This lets us to effectively exploit the insufficient resources of medical specialists and doctors.

The specific objective of this project is:

- To design a client PDA interface, which lets the users to enter and access the information from the Medical database (server).
- To explore the technical capabilities and come up with recommendations for pervasive medical database in Ethiopia conditions.
- To develop a prototype of a PDA-based client to access medical data.

1.5 Significance of the project

The result of the project can be applied to allow physicians to access patient data from remote site. That means it can help the physicians to access the medical data from the medical database (server) while they are “on the go”. This is especially very important for Ethiopia, which has inadequate number of medical expertise through the country. For instance, clinician can enter and access patient information at the point of care when he/she is on the go. He/she can also update the patient information. Doctors can also care for the patient by retrieving the information from the remote site.

As mentioned above, the result of the project can be applied to allow the clinician, doctor, etc to give treatment and guidance for the patient by accessing and entering the information from the remote site.

1.6 Organization of the Report

This project report is outlined as follows: Chapter 2 discusses related works on using a PDA-based client data access. In Chapter 3, problem analysis of the project is discussed. The tools/technologies used to implement the prototype of the project is discussed in chapter 4. Chapter 5 presents the architecture of the system, and the medical database design. The design of the prototype, the result of the prototype is represented in Chapter 6. Chapter 7 presents the conclusion and future works.

Chapter Two

2. Related Works

2.1 PDA as Client to access Medical data

PDAs have already been implemented for medical application purpose in different places. The University of Washington (UW) Neonatal Intensive Care Unit (NICU) has implemented a computer-based patient database system that can be accessed by PDA client. It used client/ server architecture to implement the system. In the client side, Handspring Visors, which is Palm OS PDAs model, is used to allow the physicians to enter and access information. In the server side, Pendragon Forms, which is one of the PDA database applications for Palm operating system (OS), is used to link the information on the PDA to the central database and it is housed on the Personal Computer (PC) as middleware. Microsoft access has been used as the ODBC compliant database to store all patient information on the PC. The programming language used was visual basic. Preset forms allow Visors both to query the PC database for information and enter new information using PDA [1, 2].

After the UW system was implemented, MobileNusreTM, a prototype of mobile nursing information system using PDA was developed in 2001 [8]. The MobileNurseTM was used for retrieving patients' information such as physicians' orders and test results at anywhere or anytime. Client/server architecture was used. The ultimate goal of this system was to implement mobile computing system, which communicates with Hospital Information System (HIS), directly. The framework of the development was divided into two stages. First, to implement the interactive system between PDA and mobile support system (MSS). MSS is a unit server located at a nursing station. It stores and communicates patient data with PDA. Second, to interface PDA with HIS [8]. Figure 1 represents Architecture of MobileNurseTM system, which is the framework of the development

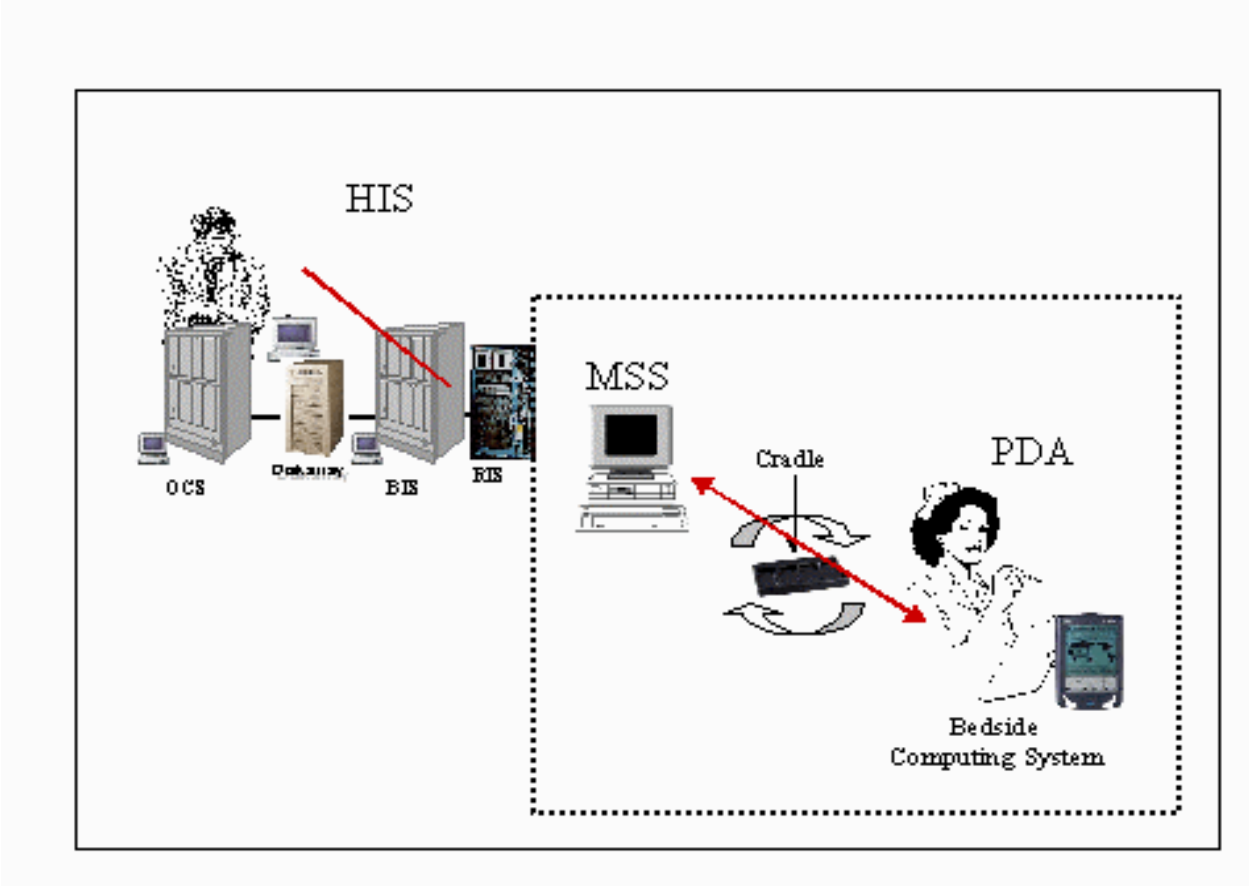


Figure 1: Architecture of MobileNurseTM system [8]

PDA communicates with HIS via MSS. The autosynchronization module was developed to interchange updated information between PDA and MSS consistently. If PDA is placed on the cradle, autosynchronization with the MSS is executed [8].

MobileNusreTM was tested at the Clinical Trial Center, Seoul National University Hospital, Korea. Six registered nurses volunteered and performed nursing care for four stimulating patients with MobileNusreTM. A Pentium PC was used as mobile supporting system, which supported and communicated data with PDA. During the clinical trial, doctors can enter physician's order into MSS and the information was interchanged between MSS and PDA when autosynchronization was activated. Nurses were able to refer patient data with PDA at any time or any place [8].

In distributed healthcare organization, handheld computers (or PDAs) as clients can also support to access patient information and update master database wirelessly [11]. A prototype Clinical Trials Information System on PDA-based client was developed to eliminate medical errors, save personnel time, and minimize the need for off-line validation since 2004. Client/server architecture was used. In the client side, an iPAQ 3800 series Pocket PC with 802.11b wireless network capabilities was used. In the server side, a Pentium based desktop running Windows XP with a Lynksys 802.11b wireless access point was used. The Pocket PC software was developed with Microsoft eMbedded Visual Tools 3.0. The server software was developed with Microsoft Visual Basic 6.0 and Microsoft Access 2002. To optimize the user interface, user input was restricted to numeric data and the selection of text strings from previously defined lists. The data was organized into a hierarchy of independent display objects, and used focus-and-context visualization to display thumbnail views of objects not currently in focus [11].

In summary, PDAs as clients can allow a medical specialist to access patients' information to provide treatment and guidance of patient at anywhere or at any time. This helps to exploit the insufficient resources of medical specialists and doctors in many countries.

2.2 PDA-based data access

Since 2002, Yuanli Wang [4] has implemented "Java applications on PDA" project for small business applications. The project was implemented for sales system to maintain and access the customers' information. Client/Server architecture is used to implement the project. On the client side, Palm OS based PDAs are used as client side platforms. Sun's Java 2 Platform Micro Edition (J2ME) technology is used to develop the client side Java applications, which can run on PDAs machines with stringent memory budget and with processors a lot less powerful than those used on typical desktop and server machines. Sun's Java 2 Platform Micro Edition (J2ME) technology consists of CLDC and MIDP APIs, which are used to develop client side applications. Others Java platforms namely J2SE and Java 2 Enterprise Edition (J2EE) are also utilized to build an end-to-end business application in the project. To test the project, the development environment software such as Palm OS Emulator (POSE) and Sun's ktoolbar is installed and used. The client device needs an HTTP connection to communicate with a backend servlet running in a web server [4].

On the server side, a servlet is running in the Apache Tomcat web server. The servlet receives data from PDA clients. Upon receipt of the client data, it opens a database connection to the Microsoft Access data source, and executes queries to insert the data into the database. After the data is saved to the database, the servlet will send a message back through the same HTTP connection it gets the data from. The message tells the device whether the data is successfully synchronized with the database [4].

PDAs as a multimedia clients in MILLION, a distributed multimedia system was developed in an ESPRIT project [12]. Using PDAs as clients in mobile multimedia applications makes very attractive because of their small size and ease of use. Alas, the cost for integration of PDA in distributed multimedia systems is high, as the resource limitations on PDA require client adaptation, and as software development on PDA is comparatively expensive. PocketWeb was used as a tool facilitating PDA integrating in multimedia applications based on the web protocols. While PocketWeb is primarily perceived as web browser for end users, it actually facilitates integration of PDA in intranet/internet applications based on the web protocols. PocketWeb embodies strategies for handling multimedia on PDA, thus minimizing if not eliminating the cost for adaptation of user interfaces to PDA resources. It is applied within the ESPRIT project MILLION for integration of PDAs as clients in a multimedia system for the tourism sector. Client/server architecture was used. The objective of MILLION was to develop a coherent set of tools and services for municipal tourism, including tools for commercial transactions. In the cities of Venice, Bologna, and khania/Crete MILLION system pilots are being installed currently [12].

To sum up, none of all the above-related systems are accessible for external users. So, in this project in order to exploit the opportunities of pervasive computing for medical data access, we are required to develop the PDA client locally. In this project, a PDA client allows the medical specialists and doctors to access medical data for providing treatment and guidance for the patient. This helps us to exploit the scare resources of the medical specialists and doctors in our country.

Chapter Three

3. Problem Identification

Patient medical records are very important documents to give better treatment and guidance to patients. For this reason, it should be easily accessible and retrievable. The major purposes of patient medical record are to [21]:

- store the patient data,
- store the medical specialists' findings and the treatment given to patients,
- keep the medical history of a patient such as laboratory results, X-ray results, etc,
- serve as a formal and legal report of care,
- Provide information for public health, epidemiological study and clinical research.

Electronic patient medical record system is one of the methods that used to handle the patient medical record. The Electronic patient medical record system would provide the following capabilities [21]:

- record patient care that makes patient chart,
- record activities dealing with the patient's care: this covers all activities to schedule or record patient care including ordering clinical laboratory and X-ray tests, medications and receiving back results; documenting patient care, producing documents in the chart; and making patient appointments.
- display the clinical information that makes up the patient medical record,
- quick overview of clinical information for a patient i.e., a summery of patient medical record to the medical specialist (e.g. current medications, past encounters, allergy, immunization, etc),
- select clinical information and search for clinical information for display,

- identify patients of interest to medical specialists,
- Enable clinical research.

Butajira Hospital has been chosen because we believed that it could represent patient record management needs in the country. Furthermore, since the objective of this project is to develop mobile medical information system, associating our proposal on an established system will reflect the real environment requirements.

An Electronic patient Record system has been implemented in Butajira Hospital since 2004 [21]. In Electronic patient record system, medical data are accessible only from fixed stations. In order to allow the medical professionals to access medical data from anywhere, and at anytime, mobile information systems are very important.

Currently, the Butajira Hospital is using electronic health care system. This alleviates the problems of paper-based patient medical record system in the hospital. But, this does not mean that there is no problem. Like other hospitals in Ethiopia, there is very high shortage of the medical specialists and doctors. This is a big problem that affects the quality of health care to patients.

The following are the impacts of this problem on the quality of patient health care. For instance, if there is no specialist to a particular disease in the hospital, the referral will be written to Tukir Anbensa Hospital. Again, if there is no specialist to that particular disease in Tukir Anbensa Hospital, the referral will be written to aboard. During this referral process, the following defects can be occurred.

- The patient may suffer tremendously and loss of life.
- The cost is unaffordable for most of patients. Because of this, many patients cannot get adequate treatment and guidance for the disease. They die due to the disease

In general, the shortage of medical professionals affects the quality of health care to patients in Ethiopia.

In addition to automating patient record management, in this work, we recommend that mobile access to medical data will improve the efficiency of patients' health care. Mobile access to medical information helps to effectively exploit the scarce resources of medical professionals in the Hospital. This improves the quality of patients' health care in Butajira Hospital, and in Ethiopia in general.

Chapter Four

4. System Development Environment

In this chapter, the tools and development environment those are relevant to attain the objectives of this project are presented.

4.1 Java 2 Micro Edition (J2ME)

J2ME is the newest and smallest addition to the Java family. Sun has extended the scope of Java technology with the introduction of Java™ 2 Platform, Micro Edition (J2ME ™) in 1999 [18. J2ME is a set of technologies and specifications developed for small devices like pagers, mobile phones, and PDAs. It is also the enabling technology for Java applications running on machines with stringent memory budget and with processors that are a lot less powerful than those used on typical desktop and server machines. It is a standard subset of the Java (J2SE) class libraries, which enable Java applications to run on small computing devices. Just like the other Java technology, J2ME platform maintains the following qualities [14, 15, 16, 17, 18, 19]:

- Providing built-in consistency across products, which enable applications to run anywhere, any time, over any device.
- The power of a high-level object-oriented programming language with a larger developer base.
- Portability of code.
- Safe network delivery, and
- Upward scalability with J2SE and J2EE.

J2ME has many advantages over the technologies targeting at the small wireless device such as Wireless Application protocol (WAP) or i-mode [15, 16, 17, 18, 19]:

- J2ME applications can reside on the client device. The applications can function on the client device even when the device is used independently without connecting to the network. A certain amount of data can be stored on the device before being finally transferred to the server through wired or wireless connection at the user's convenience.

- J2ME offers powerful client side functionalities that make efficient use of client side computing resources.
- J2ME applications are more secure and developers have full access to HTTPS for end-to-end application security.
- A J2ME application need only minimal infrastructure requirement and does not incur extra infrastructure work, such as special network protocol implementation.
- J2ME applications are widely embraced by the Industry. Nearly every handset, PDA, and carrier has adopted the J2ME specification.

J2ME APIs are designed as two major building blocks of configurations and profiles. These are described as follows [15, 16, 17, 18]:

Configurations- defines a set of class libraries representing common functions available for a broad range of devices. It also defines the basic run-time environment as a set of core classes and a specific Java Virtual Machine (JVM) that run on specific types of devices. It consists of a combination of a virtual machine and a minimal set of class libraries designed to provide the base functionality for a distinct set of devices with similar characteristics, such as network connectivity, processor power and memory. Figure 2 describes the two J2ME configurations. The two configurations are described below.

- **Connected Device Configuration (CDC)** is designed for devices with more memory, faster processors and greater network bandwidth. Examples include TV set-top boxes, Internet TVs and high-end communicators. This configuration is used with the C Virtual Machine (CVM) and is used for 32-bit architectures requiring more than 2 MB of memory.
- **Connected Limited Device Configuration (CLDC)** defines targeted Java platforms that are small, resource-constrained devices, each with a memory budget in the range of 160KB to 512KB. This configuration is composed of the Kilo Virtual Machine (KVM) and core class libraries that can be used on a variety of devices such as cell phones, two-way pagers, personal organizers, home appliances, Palm hand-held computer (or Entry level PDAs), etc. It is used specifically with the KVM for 16-bit or 32-bit devices with limited amounts of memory, small processors, and intermittent network connections. It is the configuration (and the virtual machine) used for

developing small J2ME applications. From a development point of view, its size limitations make CLDC more interesting and challenging than CDC. It is also used to develop our drawing tool application. Thus, it is a standard, portable, and minimum-footprint configuration for small resource-constrained mobile devices.

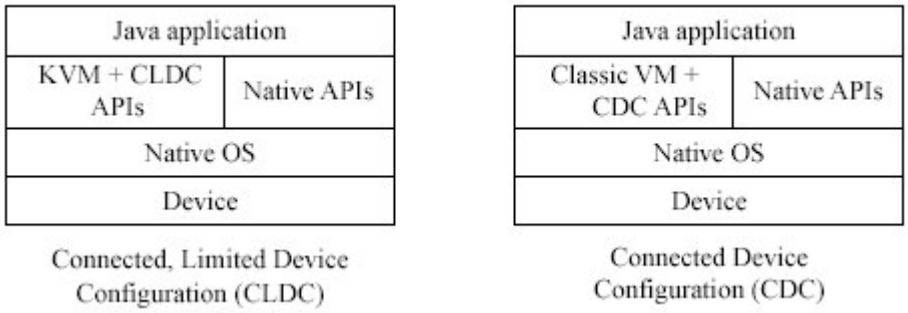


Figure 2: The two J2ME configurations

Profiles-defines the minimum set of the application programming interfaces (APIs) available on a particular "family" of devices representing a particular "vertical" market segment. Profiles are implemented "upon" a particular configuration. It is the most visible layer to users and application providers. Figure 3 shows J2ME profiles [16, 17, 18, 19].

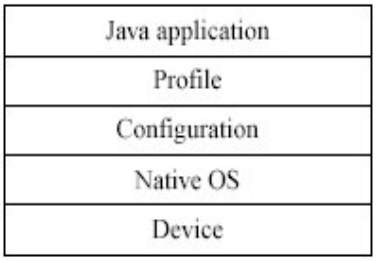


Figure 3: J2ME profiles

A number of profiles are defined or are in development. But the best-supported profile up to now- is Mobile Information Device Profile (MIDP), which is built on CLDC [4, 14].

MIDP provides a standard platform for small, resource-constrained, Wireless-Connected Mobile Information Devices (WCMID). It also facilitates the development of Java applications that run on mobile devices such as J2ME enabled mobile phones and entry level PDAs. MIDP, which is

built on top of Connected Limited Device Configuration, can be considered as a vertical set of classes. This means that MIDP has access to all the classes provided by the Connected Limited Device Configuration. The MIDP profile, together with CLDC, constitutes a complete software platform for small, resource-constrained, wireless-connected mobile devices with the following characteristics [14, 16, 17]:

- 512k total memory (ROM + RAM) available for Java runtime and libraries.
- Limited power, and typically battery operated.
- Connectivity to some type of wireless network with possibly limited bandwidth.
- User interfaces with varying degrees of sophistication.

MIDP is a specification that provides a core set of libraries for writing Java applications targeted for mobile devices. MIDP APIs provides classes to be used in mobile Java applications: application, user interface, persistent storage, networking, and timers [14, 16].

4.2 Programming in MIDP

Performance looms as a big concern of the developer to write a program for small computing devices. Memory and processing speed are the primary platform factors determining the execution speed of a computer program on such devices. Hence, these same constraints are challenging factors when developing applications for PDA using MIDP [4].

MIDlet is another name for Mobile Information Device Profile (MIDP) application.
It is a Java application that conforms to the specifications set out by the Connected, Limited Device Configuration and the MIDP. Every MIDlet device contains application management software, which controls the installation and execution of the MIDlets. A MIDlet is the name given to the code that executes on your mobile device. MIDlet programming is easy as compared to programming with Java 2, Standard Edition, because the MIDP API is simpler [14, 16].

For any class to become a MIDlet, first of all it should inherit from the javax.microedition.midlet.MIDlet class. Henceforth, we will take a look at the development of a MIDlet from several different aspects. We will therefore examine the MIDP life cycle methods, the GUI objects, the HTTPConnection, and the persistent storage class subsequently [4, 17].

.

4.2.1 MIDP Life Cycle methods

The life cycle methods of a MIDlet is one of the most important things programmers need to learn before they can start writing even the simplest MIDlet. Three life cycle methods are inherited from the MIDlet abstract class. They are startApp(), puaseApp(), and destroyApp(), all of which are to be implemented [4, 14, 16].

When a MIDlet is activated, the startApp() is executed automatically, puaseApp() and destroyApp() can be called by the programmer from the active state, which cause the MIDlet to enter Paused state and Destroyed state respectively. The Destroyed state means the execution of the MIDlet program is terminated. The MIDP API does not explicitly describe what exactly a Paused state is like and how different it is from the Destroyed state. Actually, the puasedApp() method is rarely called in the sample MIDlet programs developed to date. The State-Transition Diagram (STD) depicts the general MIDlets life cycle. It is shown in figure 4 below [4, 14, 16].

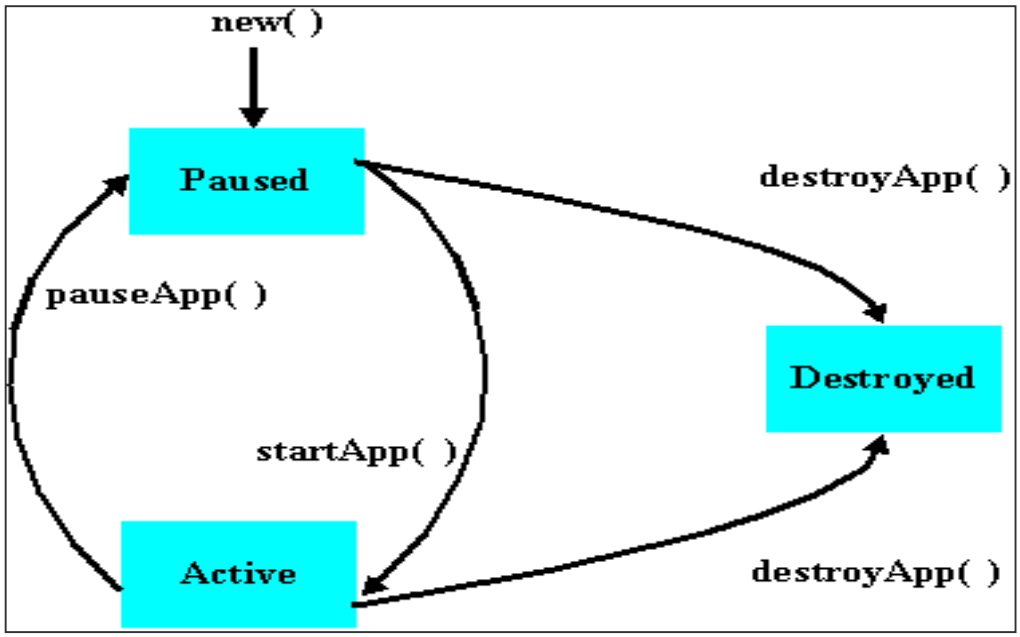


Figure 4: MIDlet life cycle

The common practice in using the MIDlet life cycle methods is to declare objects that belong to a MIDlet at the beginning of the MIDlet class. Instantiation of these objects is carried out in the startApp() method. In the destroyedApp() method, resources held by the MIDlet such as a open

Internet connection or a open connection to data store should be released, then the notifyDestroyed() method is called to notify the application management software that MIDlet has entered the Destroyed state [4, 14, 16].

4.2.2 MIDP GUI Classes

The MIDP GUI classes are in package javax.microedition.lcdui. These are GUI components that have been optimized for the mobile information device profile. The Display class controls the display and input process of the system. There is one instance of Display for each MIDlet. A reference to that instance can be obtained by calling the getDisplay() method. The Displayable class family represents various kinds of screens upon which the visible GUI components (Items) are displayed. A MIDlet can have multiple instances of Displayable. The Displayable and Item classes and their inheritance structure are shown below [4, 14, 16].

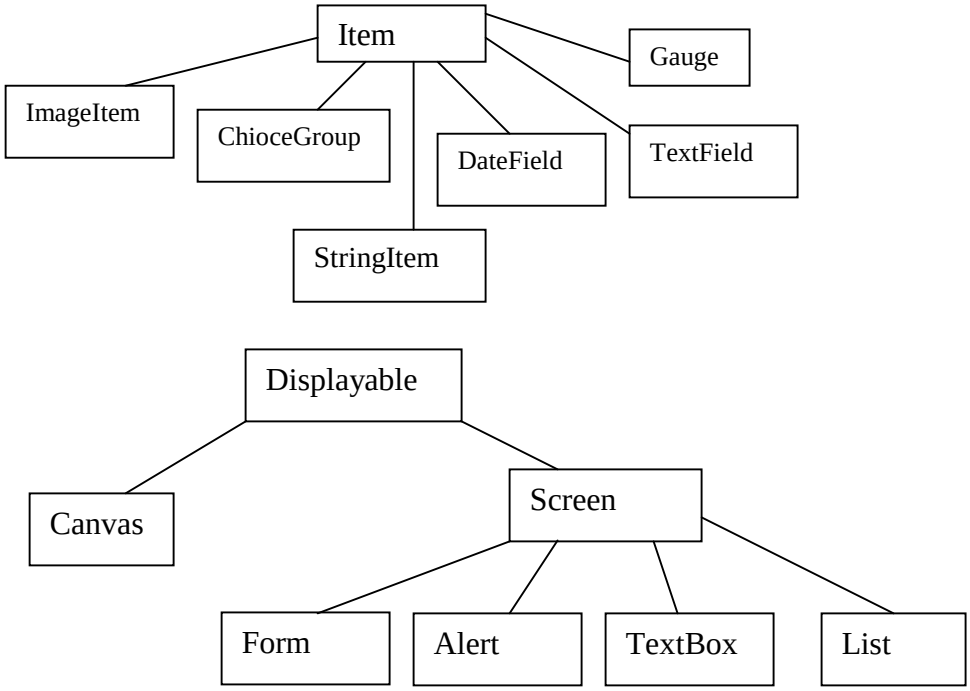


Figure 5: MIDP GUI Class Hierarchy

CommandListener and other interfaces are the basis for constructing MIDP GUI event model. It is found in the same packages as the GUI classes. A MIDlet implements the CommandListener interface and becomes a listener of its Command object events. By implementing the CommandAction() method of the CommandListener interface, the MIDlet can respond to actions

invoked on different command objects as appropriate. The appearance of a Command object differs on different devices and with different Command property. On Palm PDA, it always looks like a button [4, 14, 16].

4.2.3 MIDP Persistent Storage Classes

MIDP API has a mechanism with which data are persistently stored. The RecordStore class in the javax.microedition.rms package is designed for this purpose. Once created, a record store exists even when the MIDlets are not active. MIDlets within a MIDlet suite can create multiple record stores. Each of these record stores should have different names. According to MIDP API, a record store belongs strictly to the MIDlet suite from which it has been created. That means a record stores is only accessible by the MIDlets in the MIDlet suite that created the record store. When the MIDlet suite is removed from a platform, so are all the record stores that belong to it. A record store can also be deleted explicitly by any MIDlets that has access to it [4, 14, 16].

Data is stored as byte arrays in the record store. Records in a certain record store are uniquely identified by an integer number called recorded. The first record created in the record store will have recordId of 1. Each record added later will be assigned a recordId one greater than the record added before it. If a record is deleted from the record store, the recordId for this record is not reused. The method enumerateRecord() is often used to retrieve the data in a record store. The method returns a RecordEnumeration object, which can be used to further process each record that has been retrieved. One more thing the developer should be reminded about is that the record store is not actually closed until closeRecordStore() is called as many times as openRecordStore() was called. It is suggested that openRecordStore() and closeRecordStore() be strictly coupled in the code. In cases where the coupling of **open** and **close** of the record store is difficult to implement, a while loop can be used to repeatedly perform the close record store operation until the recordStoreNotOpenException is thrown [4, 14, 16].

4.2.4 MIDP Internet Connections

The javax.microedition.io package includes the connection classes for devices to communicate with the outside services. An HttpConnection to the servlet on the remote web server is opened by the calling the Connector class's static open() method. After the

connection is established, an OutputStream and an InputStream are created to send and receive data to and from the Internet connection [4, 14, 16].

In this project, the data sent over the Internet connection is formatted in a certain way. The server at the other end of the connection knows the data format. It extracts records from the stream of raw data, encodes them in SQL queries, and stores the records into the database. After the database operation is completed, the server sends a message back through the same connection telling the PDA client whether the synchronization process is successful or not [4, 14, 16].

4.3 J2ME Wireless Toolkit

The J2ME Wireless Toolkit is created by Sun Microsystems. It is a set of tools that is powerful and easy to use. It provides a compiling and testing environment for developing applications for CLDC/MIDP compliant mobile phones and entry level PDAs. It automates several of the tasks related to building MIDP applications [4, 10].

The core component of the J2ME Wireless Toolkit is KToolbar. It comes with the J2ME CLDC and MIDP runtime environment, documentations, and examples, so developers do not even need to download and install the CLDC or MIDP on their own. Using the KToolbar, developers no longer have to go through the hassle of writing complicated batch files to compile the source code and build the device-specific executable files. With the KToolbar, developers are also freed from loading the recompiled code to the device each time a little change is made to the source code. It does almost everything for you except for writing the code. Testing and running the application is as easy as hitting the run button on the menu bar. Developers can test their applications on several mobile phone models including Palm Operating System Emulator (POSE). KToolbar can also be configured in a way that it can find POSE installed on the computer, and run applications on the POSE when needed [4, 10].

POSE is a software program that runs on your PC, and acts in almost every way just like your Palm OS hardware device, PDA. POSE consists of two components [4, 13]:

1. The emulator program simulates the hardware.

2. A ROM image file contains the Palm Operating System that you will need to use the emulator.

4.4 Java Servlets

A servlet is a Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a request-response paradigm. It runs on a Web server, acting as middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server. It has full access to all Java APIs, hence it is a powerful tool for developing web-based applications. Their job is to [4, 13, 20]:

- Read any data sent by the user.

  This data is usually entered in a form on a Web page, but could also come from a Java applet or a custom HTTP client program.

- Look up any other information about the request that is embedded in the HTTP request.

  This information includes details about browser capabilities, cookies, the host name of the requesting client, and so forth.

- Generate the results.

  This process may require talking to a database, executing an RMI or CORBA call, invoking a legacy application, or computing the response directly.

- Format the results inside a document.

  In most cases, this involves embedding the information inside an HTML page.

- Set the appropriate HTTP response parameters.

  This means telling the browser what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

- Send the document back to the client.

This document may be sent in text format (HTML), binary format (GIF images), or even in a compressed format like gzip that is layered on top of some other underlying format .

4.4.1 Servlet interface

Architecturally, all servlets must implement the Servlet interface. This enables a servlet to reside in the framework provided by the Web server that will call the servlet's methods. This interface defines the following methods that describe the servlet's life cycle [15, 20]:

- public void init(ServletConfig config) throws ServletException
  This method is called once when the servlet is loaded into the servlet engine, before the servlet is asked to process its first request.

- public void service(ServletRequest request, ServletResponse response) throws ServletException, IOException

  This method is called to process a request. It can be called zero, one or many times until the servlet is unloaded. Multiple threads (one per request) can execute this method in parallel so it must be thread safe.

- public void destroy()
  This method is called once just before the servlet is unloaded and taken out of service.

The servlet packages define two abstract classes that implement the interface Servlet. These classes provide default implementations of all the **servlet** methods. These are described as follows [15, 20]:

- Class ***GenericServlet*** (from the package *javax.servlet*)

- Class ***HttpServlet*** (from the package javax.servlet.http)

Web-based servlets typically extend class ***HttpServlet.*** Class ***HttpServlet*** overrides method service to distinguish between the typical requests received from a client Web

browser. The two most common HTTP request types (also known as request methods) are ***GET*** and ***POST*** [15, 20].

Both ***GET*** and ***POST*** requests get information from the server. But in the case of ***POST*** request, the results are not cached on the local computer. The two requests can send data to the server; however, GET requests only include that data as part of the URL. POST requests have a separate message in which data is placed [15, 20].

Class ***HttpServlet*** defines methods ***doGet*** and ***doPost*** to respond to ***GET*** and ***POST*** requests from a client, respectively. These methods are called by the ***HttpServlet*** class's ***service*** method, which is called when a request arrives at the server [15, 20].

Methods ***doGet*** and ***dopos***t receives as arguments an ***HttpServletRequest*** object and an ***HttpServletResponse object*** that enable interaction between the client and the server. The methods ***HttpServletRequest*** make it easy to access the data supplied as part of the request. The HttpservletResponse methods make it easy to return the servlet's results in HTML to the web client [15, 20].

4.4.2 Multitier Applications: Using JDBC from a Servlet

Servlets can communicate with databases via JDBC (Java Database Connectivity). The servlet-based application provides the capability to connect to any relational database registered as an ODBC data source using JDBC. After the application is connected to the database, it can browse the database meta-data and instance, and manipulate the database by issuing ad hoc Structural Query Language (SQL) statements. The application can be run using a browser (Netscape 4.0 or Internet Explorer 4.0 or higher) [15].

JDBC is a mechanism that allows Java to communicate with databases using a standard Application Programming Interface (API). It provides a uniform way for a Java program to connect to a variety of databases in a general manner without having to deal with the specifics of those database systems [15].

In multitier architecture, Web servers are increasingly used to build the middle tier. They provide the business logic that manipulates data from databases and communicates with client

applications. Servlets, through JDBC, can interact with popular database systems. Developers do not need to be familiar with the specifics of each database system [15].

4.5 Apache Tomcat Web Server

Tomcat is a freely available reference implementation of the Java Servlet and JavaServer Pages (JSP) specifications. It is developed and released under the Jakarta project as the official JSP 1.1/Servlets 2.2 reference implementation by the Apache software Foundation [9].

The mission of the Jakarta Project is to provide commercial-quality server solutions based on the Java Platform that is developed in an open and cooperative fashion. The product of this project, Tomcat, is a reference implementation of the Java Servlet and JSP Specifications, which can run standalone as well as integrated into the Apache Web Server. This reference implementation provides an operational definition for the Enterprise Java™ JSP and Servlet application-programming interfaces (APIs). Tomcat available to any company or developers to be used in web servers, development tools, and to create dynamic, interactive Web sites [9].

To sum up the above development environments of this project, we used CLDC/MIDP, J2ME Wireless Toolkit 2.2, Servelts with JDBC, and Apache Tomcat Web server 5.0.16. The configuration of these all tools will be described in the following chapter 4.

Chapter Five

5. System Design

5.1 System Architecture

The architecture, which is used in this work, is three-tiered client-server model. More can be read on three-tiered client architecture [9]. On the client side, Palm OS PDAs are used. MIDlet suite that contains a group of MIDlet applications interfaces is used to develop for the client side applications. To run the MIDlet applications on the target device, PDA, the code should be converted to .prc file.

Applications development under a real environment was not possible due to material resource limitations. Thus, we used an emulator to develop our prototype. To simulate the Palm OS PDA hardware device, we used Palm Operating System Emulator (POSE). POSE can be configured with J2ME Wireless Toolkit development environment. The MIDlet application enables PDA client to store records entered by the user for some time and then send the data to the Apache Tomcat web server [4].

On the server side, a Servlet is running in an Apache Tomcat web server. The servlet, which is middleware, receives data from PDA clients. Upon receipt of client data, it opens a database connection to the Microsoft access data source, and executes queries to insert the data into the database. After the data is stored to the database, the servlet will send a message back through the same HTTP connection it gets the data from [4].

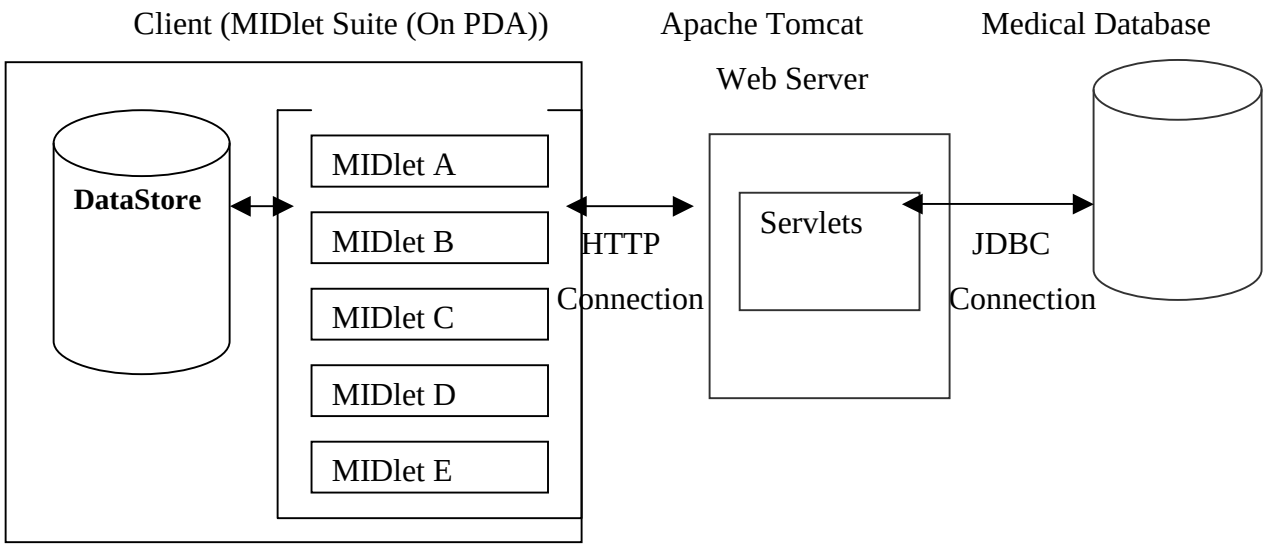


Figure 6: General Architecture for accessing medical data on a PDA.

In figure 6, there is a MIDlet suite, which is the interface to the PDA client. This MIDlet suite consists of MIDlet application such as MIDlet A, MIDlet B, MIDlet C, MIDlet D, and MIDlet E. The MIDlet applications interface enables PDA client to store records entered by the user for some time in the **DataStore** as shown in the figure. The **DataStore** is handled by Record Management System (RMS), which is explained in the previous chapter.

The MIDlet Application interface also enables PDA client to hook up with the servlet on the Apache Tomcat web server using HTTP connection. Servlet is used as a middleware to get access with the medical database on the server. Servlet is connected with the medical database using JDBC connection as shown in the figure above. The technologies here are explained in previous the chapter.

5.2 The Prototype

Based on the system architecture presented in the previous section, we have designed the prototype called “Mobile Access to Medical Information (MAMI)”. MAMI is a PDA-based client/server application that is used to access medical data from medical database server. Because of the limitations that are inherent in small mobile computing devices, the prototype development requires us to change the way in which we code the applications. This requires us to consider the following basic design issues [22].

To design MAMI, there are several basic design issues which we should be considered to develop the MIDlet applications for small mobile devices. This basic design issues are described in Annex II.

5.2.1 Design of the Prototype Developed

In order to develop the prototype, the MIDlet application interface will be used. This application interface runs on small mobile devices such as PDAs, mobile phones, etc. In the design of the prototype, the structure of the client and server environments is described. Figure 7 describes the structure of the Client Environment and Figure 8 describes the structure of the Server Environment.

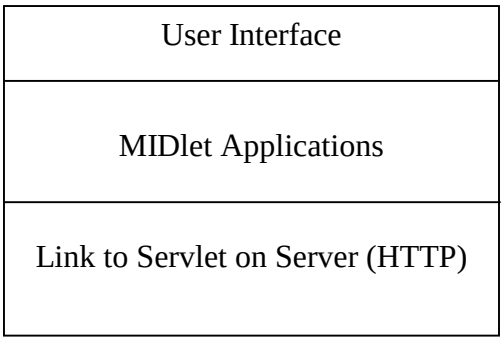


Figure 7: Structure of the Client Environment

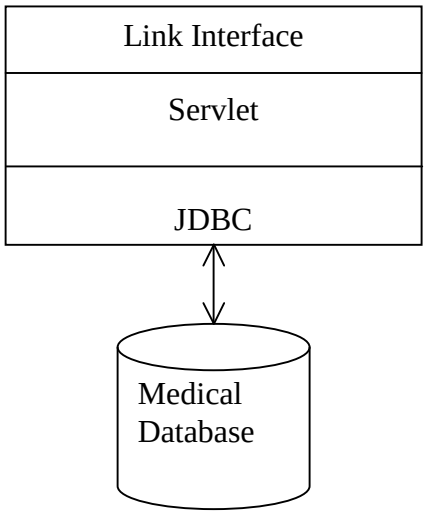


Figure 8: Structure of the Server Environment

The design of the MIDlet applications can be described using class diagram. The class diagram in Figure 9 describes the major classes that have been used in the prototype. It shows the methods of each class and the relationship between classes.

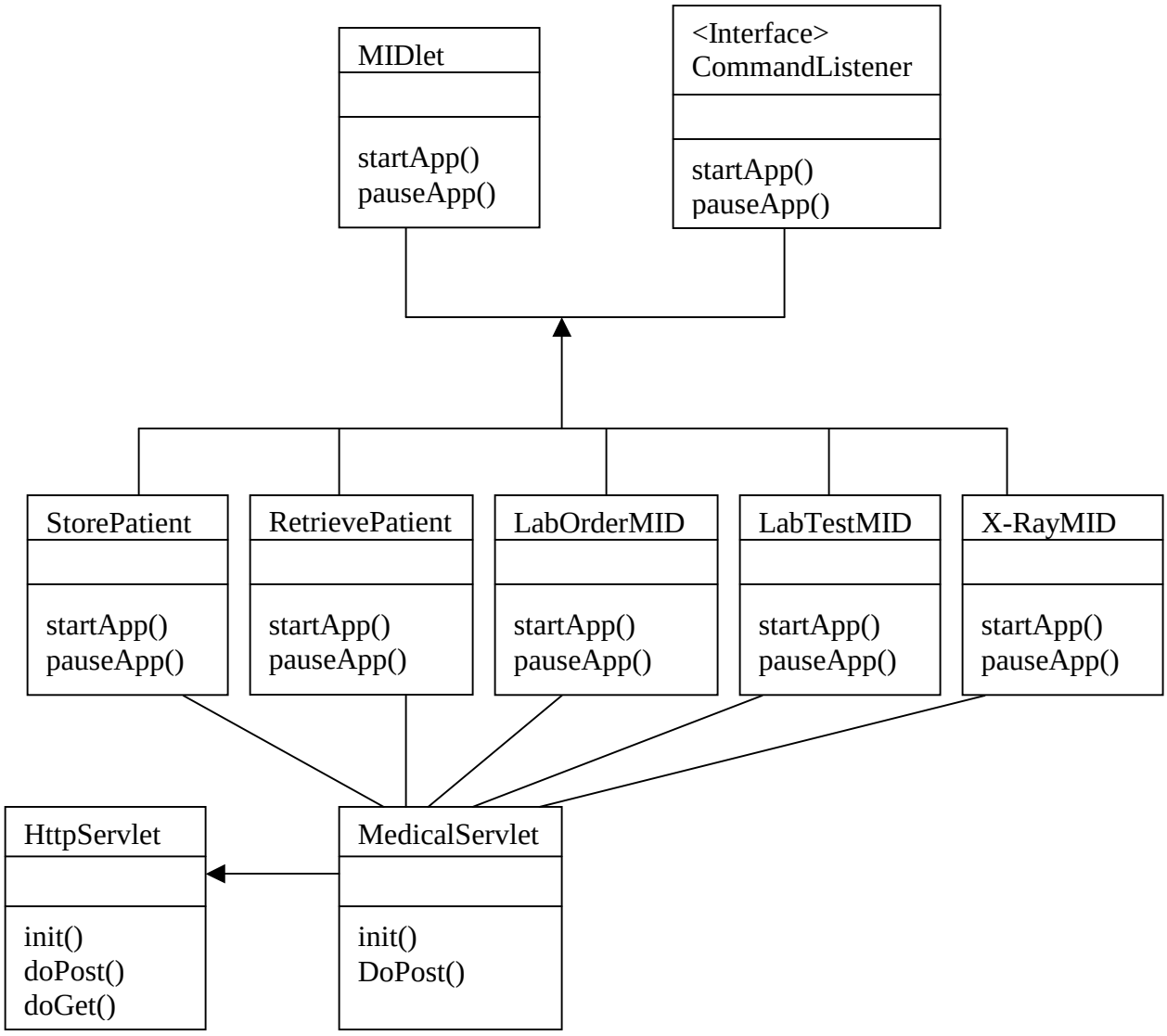


Figure 9: Class diagram of the prototype of the project

At the heart of the diagram, there are MIDlets that forms the user interface. The MIDlets are all inherited from a common parent class **MIDlet** found in the MIDP javax.microedition.midlet package. MIDlet classes define several MIDlet life cycle methods: startApp(), pauseApp(), and destroyedApp (), which are discussed earlier.

The MIDlets classes also implement the CommandListner interface in the javax.microedition.lcdui package. MIDP GUI package supports an event model similar to those

supported by other GUI Java platforms such as Swing and Applet. By implementing the listener interface's method CommandAction ( ) and registering itself as an event listener, a MIDlet is able to react to all kinds of events that happen to the GUI objects within the MIDlet.

As it is discussed earlier, mobile small devices have very limited memory and processing power. They can hardly be qualified as platforms for building sophisticated storage systems. It is neither efficient nor secure to store large amount of sensitive data on PDAs or mobile phones for a long time. They are quite sufficient for storing small amount of data entry generated by users while they are on the go.

To achieve the ultimate goal of remotely saving all the information stored temporarily on the device to the database on the server side, a servlet is nicely fitted into the scene. A servlet is a Java program running in a web server. It receives a client request from a HTTP connection and sends back responses to its client. The MedicalServlet is a servlet class inherited from javax.servlet.http.HttpServlet. Two of the servlet lifecycle methods, init ( ) and destroyed ( ), are overridden. In the init ( ) method, a JDBC connection is opened to the Microsoft Access database used as the project data store with the name Butajira Hospital. The design of this medical database is discussed in the preceding chapter. The JDBC connection is closed in the destroyed () method, and the related resources are released. The doPost () method gets information extracted from the devices storage class by reading from the client HttpSevletRequest. It reformats the information as records for different kind of database tables, and inserts the records to the appropriate tables.

5.2.2 The Prototype MAMI

In order to realize our proposal for mobile access medical information, we developed a prototype called Mobile Access to Medical Information (MAMI). MAMI is developed using J2ME. The objective of MAMI is to allow medical doctors/specialists to access medical data from anywhere and anytime so that the patient can get better treatment and guidance. MAMI is developed using the configuration described in ***Annex I***.

In order to test MAMI, a medical database is necessary. The medical database is used to store patient medical records, medical history, appointments and daily schedules of patients for the medical specialists; medication order, clinical laboratory, and X-ray tests, etc [21].

For the purpose of associating our MAMI to a real situation, we have chosen to use the medical database of Butajira Hospital. The Butajira Hospital medical database was designed by Newayneh Ketshela at Department of Information science entitled “Developing Information system for Butajira Hospital”[21].

To describe how MAMI works, let’s consider some examples in which medical doctors/specialists can access patient information to give treatment and guidance using his/her PDAs. The applications in MAMI main menu are described in **table 6.3**.

Applications	**Description**
Query	Used to retrieve the patient information from the medical database on the web server.
DataEntry	Used to store the patient information into the medical database
MedicalHistory	Used to retrieve the medical history of the patient from the medical database.
LabResults	Used to display/show the laboratory results of the patient from the medical database (This is only in alphanumeric format)
ImageRetrieval	Used to display the X-ray result of the patient form the medical database (This is in image and alphanumeric format)

Table 6.3: Descriptions of the applications in MAMI main menu

Example 1: A medical doctor/specialist wants to access the patient record with an ID number 1112/97. To do this, he/she opens MAMI to get its main menu. After this, he/she launches the Query in the MAMI main menu. This is shown in the figure below.

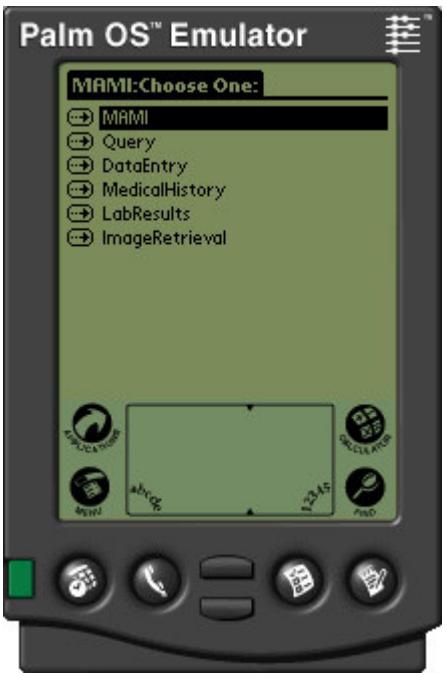


Figure 10: MAMI Main Menu

After the Query application is launched, he/she types the patient ID number using keyboard. Then click on **Display** command to connect with the **MedicalServlet** servlet on the web server.

Figure 11: Query Request Interface

The connection will be made with the **MedicalServlet** servlet on the web server using HTTP connection. Then, **MedicalServlet** servlet connects to the Butajira Hospital Medical database using JDBC connection. After connection, the servlet retrieves the patient record requested from

the patient table in the Medical database. And it also returns the response to the client based on the request. Then the requested data will be displayed on the PDA screen of Query Result Interface.

Figure 12: Query Result Interface

Example 2: Assume a medical doctor/specialist wants to see the medical history of the patient under treatment. From Figure 11 MAMI main menu, launch MedicalHistory application to get MedicalHistory Request Interface.

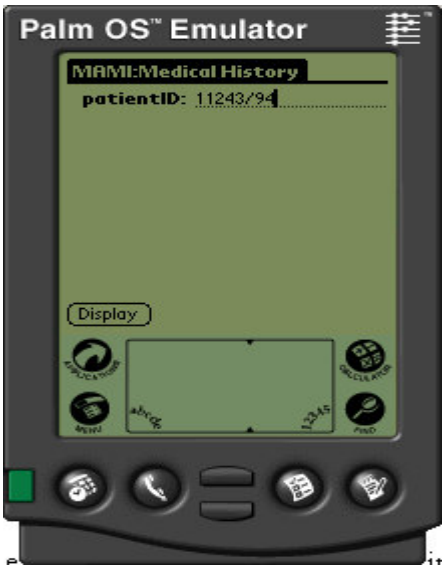


Figure 13: MedicalHistory Request interface

After the MedicalHistory application is launched, the patientID should be entered into the text field on the interface-using keyboard. After the patientID is entered, click on **Display** command to display the medical history of the patient.

The connection will be made with the MedicalServlet servlet on the web server using HTTP connection. Then, the servlet retrieves the medical history of the patient from Laborder table in the Butajira Hospital Medical database. And it also returns the response to the client based on the request and the requested information will be displayed on PDA screen of Medical History Results Interface.

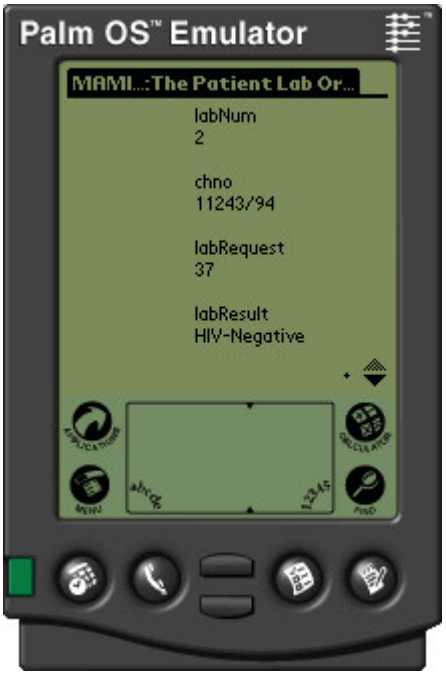


Figure 14: Medical History Results Interface

Example 3: A medical doctor can also store the patient information during treatment and guidance. From Figure 11 MAMI main menu, launch DataEntry application to get DataEntry Request Interface. After the DataEntry application is launched, he/she enters PatientID, First Name, Last Name, Sex, BirthDate, Address, MaritalStatus, DateReg, and Visit Status.

Figure 15: Data Entry Request Interface

After all data are entered, click on **Save** command to connect the application with MedicalServelet on the web server. The connection will be made with the MedicalServlet servlet on the web server using HTTP connection. Then, the servlet stores all patient data into patient table in Butajira Hospital Medical database and also responses on whether the data is successfully inserted into the database or not.

Figure 16: Data Entry Results Interface

Based on the above three examples of the result, we can concluded that the medical doctors/specialists can access medical data using PDA from medical database server to give treatment and guidance to a patient being anywhere and at anytime. This is very important to exploit the scarce of medical specialists in Ethiopia.

Chapter Six

6. Conclusion and Future Works

6.1 Conclusion

In Developing countries like Ethiopia, there is a very high shortage of medical specialists and doctors. This is a big problem that affects the quality of health care of patients. A patient may suffer tremendously to the extent of losing of life due to this problem. This is because a patient cannot get adequate treatment and guidance.

To exploit the medical professionals in the country, there should be a means by which the medical specialists and doctors can give effective service. To alleviate this problem, mobile access to medical information system is proposed in this project.

Until recently, mobile information systems are not common. But with the increasing capabilities of mobile computing, it is possible to access medical data from anywhere, and at anytime using PDA-based client. This helps to effectively exploit the scarce resource of medical specialists in a country like Ethiopia.

Three-tiered client/server architecture was proposed. On the client side, Palm OS PDAs are used. MIDlet suite that contains a group of MIDlet applications interfaces is used to develop for the client side applications. To run the MIDlet applications on the target device, PDA, the code should be converted to .prc file.

On the server side, a Servlet is running in an Apache Tomcat web server. The servlet, which is middleware, receives data from PDA clients. Upon receipt of client data, it opens a database connection to the Microsoft access data source, and executes queries to insert the data into the database. After the data is stored to the database, the servlet will send a message back through the same HTTP connection it gets the data from.

Applications development under a real environment was not possible due to material resource limitations. Thus, we used an emulator to develop our prototype. Based on the architecture proposed, a prototype of the project called MAMI was developed. To simulate the Palm OS PDA hardware device, we used Palm Operating System Emulator (POSE). POSE can be configured with J2ME Wireless Toolkit development environment.

With MAMI, the medical specialists and doctors can access the medical data while on move or while they are away from their work site to give better treatment and guidance to patients. This is very practical at places where there is a very high shortage of medical professionals like in Ethiopia. We believed that this empowers the medical specialists to have access to the medical data from a place where they are to give effective services.

Thus, this helps us to effectively exploit the insufficient resources of medical professionals in the country. This also improves the quality of patient care, since it enables the medical specialists to have more time for patients and provide ‘hand on’ care.

6.2 Future Works.

Future works include, developing PDA-based client system under real situation to allow the medical specialists and doctors to access the medical data from wherever they are to give better treatment and guidance to patients. Security is a crucial issue in accessing patient medical data. It can thus be considered in the future works. Considering the expanding wireless network infrastructure in the country, the initiatives in this work can be extended to give additional services to a national wide health network with different applications.

References

[1] Aaron E. Carroll, MD, Sunil Saluja, MD, Peter Tarczy-Hornoch, MD, Development of a Personal Digital Assistant (PDA) Based Client/Server NICU, Patient Data and Charting System, University of Washington, USA, Nov. 2001 http://faculty.washington.edu/pth/NICU-PDA.PDF visited on March 14, 2005

[2] Aaron E. Carroll, MD, Sunil Saluja, MD, Peter Tarczy-Hornoch, MD, The Implementation of a Personal Digital Assistant (PDA) Based Patient Record and Charting System: Lessons Learned, University of Washington, USA, 2001 http://faculty.washington.edu/pth/AMIA_PDA.pdf visited on March 14, 2005

[3] Medical Error Reduction and PDAs, Reviewed Articles, Feinberg Medical School of Northwestern University in Chicago, Illinois, USA, Feb. 2003. http://www.int-pediatrics.org/ PDF/Volume_18/18-2/69_77_ip1803.pdf Visited on Mar. 25, 2005

[4] Yaunli Wang, Building Java Applications on PDA (Master Thesis), Arizona State University East, USA, May 2002 http://pooh.east.asu.edu/Lindquist/Students/rptYuanliWang.pdf visited on Mar. 13, 2005

[5] Michael A. Grasso, Clinical Applications of Handheld Computers, George Washington University School of Medicine, Washington DC, USA, 2004 http://www.csee.umbc.edu/~mikeg/ papers/GrassoMA_CBMS_2004_Handheld.pdf Visited on March 30, 2005

[6] Chris Tull, Introduction to PDA, 2000 http://archive.devx.com/wireless/articles/PDA/PDAIntro.html (Visited on March 23, 20005)

[7] Terena Solomons, Beam me up! Supporting PDAs (Personal Digital Assistants) in medical libraries: new technology, Hollywood Private Hospital Library, USA, 2003 http://www.vala.org.au/vala2004/2004pdfs/57Solom.PDF Visited on March 30, 2005

[8] Implementation of Mobile Computing System in Clinical Environment: MobileNurseTM, Seoul National University, Korea, November 2001.
http://www.pdacortex.com/downloads/MobileNurse.pdf Visited on April 19, 2005

[9] Jonathan Knudsen, Wireless Development Tutorial Part II, Wireless Developer, USA, September 2003
http://developers.sun.com/techtopics/mobility/midp/articles/tutorial2/ Visited on April 21, 2005

[10] Jonathan Knudsen and Dana Nourie, Wireless Development Tutorial Part I, Wireless Developer, USA, September 2003
http://developers.sun.com/techtopics/mobility/midp/articles/wtoolkit/ visited on April 21, 2005

[11] Michael A. Grasso, Clinical Applications of Handheld Computers, George Washington University School of Medicine, Washington, DC, USA, 2004
http://www.csee.umbc.edu/~mikeg/ papers/GrassoMA_CBMS_2004_Handheld.pdf
Visited on May 2, 2005

[12] Markus Lauff, Hans-Werner Gellersen, Multimedia Client Implementation on Personal Digital Assistants, Interactive Distributed Multimedia Systems and Telecommunication Services (IDMS), (PP: 283 – 295), Springer-Verlag London, UK, 1997

[13] Deitel, Deitel, and Nietel, Internet and World Wide Web How to Program, Pearson Education, ISBN 0-13-016143-8, 2000

[14] Mark J. Balbes, Java 2 Micro Edition Part 2 “Mobile Information Device Profile”, 2002
http://www.stlwebdev.org/membership/ sigs/wireless/talks/J2MEForWirelessSIG.pdf
(Visited on March 27, 2005)

[15] Vartan Piroumian, Wireless J2ME Platform Programming, USA,
(Chapter 1, Introduction to Java 2 Micro Edition (J2ME) Platform), March 2002.
http://developers.sun.com/techtopics/mobility/ midp/chapters/j2mevartan/0130449148.pdf
(Visited on March 25, 2005)

[16] Jafar Ajdari, Java 2 Mobile Information Device Profile (MIDP), Helsinki University of Technology, 2001 http://www.tml.hut.fi/Studies/Tik-111.590/ 2001s/papers/jafar_ajdari.pdf (Visited on March 23, 2005)

[17] Wisdom Assen BV, J2ME Programming in a mobile world, December 2001 http://www.openpsa.net/archive/wisdom_white_paper_j2me.pdf (Visited on March 16, 2005)

[18] Urs Steiner, J2ME: Introduction, Configurations and Profiles, University of Zürich, 2001 http://www.ifi.unizh.ch/~riedl/lectures/Java2001-j2me.pdf (Visited on March 23, 2005)

[19] Presented by developerWorks, J2ME: Step by step, http://www.digilife.be/quickreferences/ PT/J2ME Step by step.pdf (Visited on March 27, 2005)

[20] Marty Hall, Larry Brown, Core Servlets and JavaServer Pages, Second Edition, August 2003 (Chapter 2: First Servlets) http://csajsp-chapters.corewebprogramming.com/ CSAJSP-Chapter2.pdf (Visited on March 19, 2005)

[21] Newayneh Ketshela, "Developing Information system for Butajira Hospital", (Masters Thesis). Addis Ababa University, School of Information Studies for Africa, Addis Ababa University, 2004.

[22] Eric Giguere, Mobile Information Device Profile for Java 2 Micro Edition: Professional Developer's Guide, Release 1.0, (Chapter 3: Programming Strategies for Small Devices), Feb. 2001. http://www.ericgiguere.com/books/j2me/j2me-chapter-3.pdf Visited on May 9, 2005

Annex I: System Configuration

In order to configure the system for the project, we should use the following tools [9, 10]:

- Installing the J2SE SDK
  - It should be version 1.4.2 or above. It is available on the site http://java.sun.com/j2se/.
- Installing the J2ME Wireless Toolkit
  - Install J2ME wireless toolkit. To install, execute the installation file. The installer tries to locate your J2SE SDK, if it's having trouble make sure you are pointing it to the directory where you installed the J2SE SDK. You will also need to specify whether the toolkit will run by itself (standalone) or be integrated with an IDE. The files for the toolkit will store in c:\WTK22 unless you specify a different directory, and the installer creates shortcuts for various parts of the toolkit.
  - To run the Toolkit itself, select the ***KToolbar*** shortcut. You should see the following window.

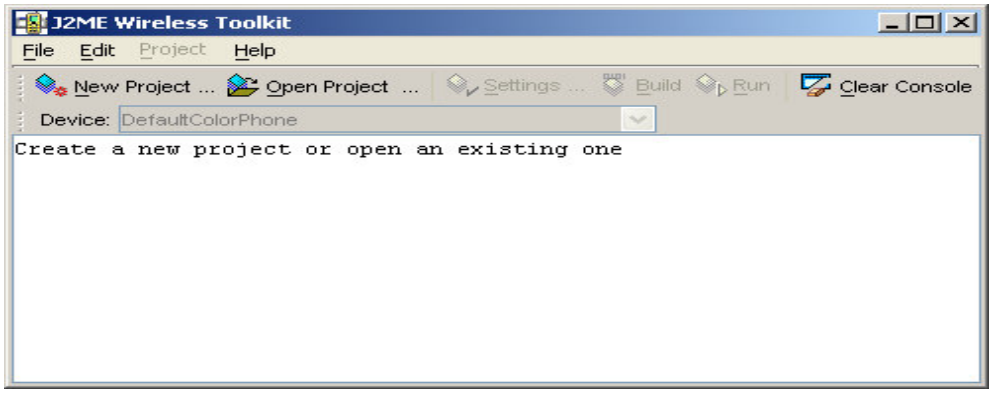


Figure 17: J2ME wireless Toolkit window

- In this project, the toolkit runs in integrated environments by specifying the URL of the servlet on the web server.

- After you run the toolkit, you can create a new project or open existing projects.

- When you create a new project, the toolkit stores each project in a subdirectory of the apps directory. The name of subdirectory is the same as the name of the project. Here, the toolkit has created a new directory, c:\WTK22\apps\HelloSuite. Each project subdirectory has a standard structure:

Figure 19: Project directory structure

- The ***bin*** directory contains the compiled MIDlet suite (a .jar file) and the MIDlet suite descriptor (a .jad file). The ***lib*** directory is the location for any additional JAR files you would like to include in your project. ***res*** directory is the location for resource files, like images or text files, that should be bundled with your MIDlet suite. Finally, the ***src*** directory is the directory where your source code should be saved.

- After you build your project, several additional directories are created. Since the toolkit uses these directories internally, you can ignore them.

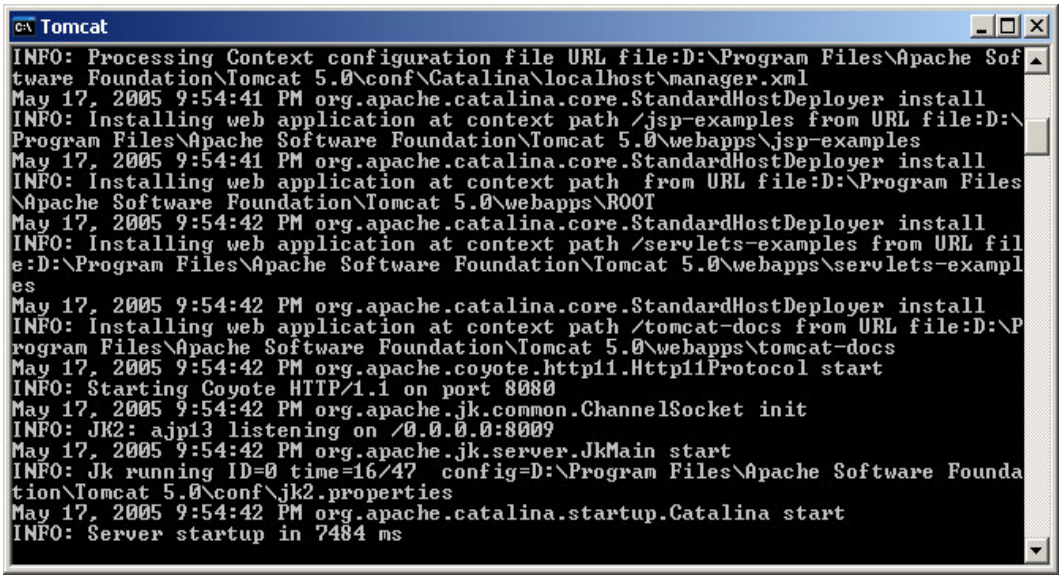
- It is available on the site http://java.sun.com/products/j2metoolkit/

➢ Installing Apache Tomcat web server

- Download Tomcat web server and install it.

- To run Tomcat you will need to tell it where to find your J2SE installation. To do this, put the location of your J2SE installation in the JAVA-HOME environment variable. On my machine, the variable has the value c:\j2sdk1.4.2.

- To run Tomcat, open a command window and change directories to Tomcat's bin directory. Type **startup** and stand back. A new window will open up and display copious initialization messages:



You can use a browser to test if Tomcat is really running. Try to open the URL http://localhost:8080/ and see what happens. If Tomcat is running correctly, you will see a default page from Tomcat with links to some servlet and JSP examples.

To shut down Tomcat, open another command window. Change directories to Tomcat's bin directory and run the **shutdown** command.

After the above tools are installed, we should use the following procedures.

1. Write a MIDlet application named ***HitMIDlet.java***, which runs on devices such as Mobile phones, PDAs by using any editor like notepad, etc. Save ***HitMIDlet.java*** in the ***src*** directory of your project. On my computer, this file is saved in *c:\WTK22\apps\HelloSuite\HitMIDlet.java*.

2. Writing your servlet source code using any editor.

3. Building a web application with a very specific directory structure on the server under the Tomcat root directory named **webapps**. This directory structure makes easy for the server to find the pieces of your applications. For instance, save the servlet source code in a file

under the Tomcat root directory named webapps/midp/WEB-INF/classes/servletName. Create the midp directory and its subdirectories now.

4. Compiling the servlet- since the servlet API is not a core part of the J2SE platform, you will need to add it to your CLASSPATH before you can compile servlets. The servlet API is contained in *common/lib/servlet.jar* under the Tomcat root directory. You can edit the CLASSPATH in the system properties or do it on the command line, as this Windows example demonstrates:

```
C:\>set CLASSPATH=\jakarta-tomcat-4.1.27\common\lib\servlet.jar

C:\>javac HitServlet.java
```

5. Deploying the servlet- you already created a new directory inside webapps called midp, where you saved the servlet source code in procedure no. 2. Now, you'll need to edit one of Tomcat's configuration files to tell Tomcat about the new web application. Open the conf/server.xml file with a text editor. In this file, web applications are called contexts. Scroll down to find the Context entry for the examples web application, which begins like this:

```
<!-- Tomcat Examples Context -->
<Context path="/examples" docBase="examples" debug="0"
         reloadable="true" crossContext="true">
```

Above or below this lengthy context entry (it's closed by </Context>, many lines down), create a new context entry for your new web application. It will look similar to the opening tag for the examples context, but you'll change the names to midp as appropriate and close the tag inline.

```
<!-- MIDP Context -->

<Context path="/midp" docBase="midp" reloadable="true"/>
```

Once you have finished adding the context entry, save the file.

The use of these steps is map incoming HTTP requests to a web application in a particular directory. Specifically, any incoming HTTP request that begins with "/midp" (the **path**) will be handed off to the web application located at webapps/midp (the **docBase**). The **reloadable** attribute helps a lot with debugging; it tells Tomcat to reload automatically any servlet class you change so you don't have to restart the server.

6. Now that you have created a web application, fill it up. Web applications have a standard directory structure, mandated by servlets specifications. Web.xml file is the essential piece of a web application that describes the various parts of the web applications. It lives in a standard location in every web application; it is always stored as WEB-INF/web.xml.

For your new application, you should create a web.xml file to make the servlet accessible to the outside world. For example, the class name of the servlet is HitServlet, and you ‘d like to make it available under a path like /hits. The path for the servlet is relative to the path for the web application, so the full path to the servlet will be http://localhost:8080/midp/hits. Copy the following text and save it as webapps/midp/WEB-INF/web.xml under the Tomcat root directory:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app

    PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

    "http://java.sun.com/dtd/web-app_2_3.dtd">
```

```
<web-app>
  <servlet>
    <servlet-name>bob</servlet-name>
    <servlet-class>HitServlet</servlet-class>
  </servlet>
    <servlet-mapping>
    <servlet-name>bob</servlet-name>
    <url-pattern>/hits</url-pattern>
  </servlet-mapping>
</web-app>
```

This file tells Tomcat to map the servlet called HitServlet to the path /hits. The **servlet-name** is internal to web.xml; it links the **servlet** element to the **servlet-mapping** element. The name **bob** is only a friendly example; you can choose whatever name you want.

To test your handiwork, go to a browser and navigate to http://localhost:8080/midp/hits. You should see the output of HitServlet.

7. Hooking Up a MIDlet to the Servlet- Now that you have a development environment that supports both MIDP and servlets, you are going to hook the two worlds together to create an end-to-end Java application. MIDlets can connect to the world at large via HTTP, and the servlet you just wrote is available to the world at large via HTTP, so it's a pretty simple matter to have a MIDlet connect to the servlet.

Start **KToolbar** (part of the J2ME Wireless Toolkit) and open the MIDlet project that you created in procedure no. 1. You're going to create a new MIDlet that connects to your servlet, retrieves its output, and displays it. For instance, you create the MIDlet HitMIDlet, which connects to your servlet HitServelt [9,10].

There are two things to be configured [9, 10]:

- First, you need to tell the **toolkit** about this new MIDlet. Click on **Settings**..., then select the **MIDlets** tab. Click on **Add** and fill in "HitMIDlet" for both the MIDlet name and class name. You can leave **Icon** blank. Click on **OK** and you should see HitMIDlet listed.

- Next, you need to define a system property that `HitMIDlet` uses as the URL for its network connection. In the **toolkit**, click on **Settings...**, then select the **User Defined** tab. Click on the **Add** button. Fill in the property name as **HitMIDlet.URL**; the value should be the URL that invokes `HitServlet`, the same URL you used in a browser to test the servlet. When you have finished, click on **OK** to dismiss the project settings window.

Now, in the J2ME Wireless Toolkit, click on ***Build*** to build the project. Assuming you don't see any error messages, you are now ready to test the application. Make sure your server is running first. Then click on ***Run*** and select HitMIDlet. If everything goes well, HitMIDlet will invoke HitServlet and display the results on the device emulator screen [9, 10].

Annex II: Basic Design Issues

There are several design issues, which should be considered to develop the MIDlet applications for small mobile devices. The basic design issues that we consider in the development of our prototype are described as follows.

1. Keep the design simple

To simplify the applications, we should remove unnecessary features. After removing unnecessary features, the next step is to reuse the user interface wherever possible. A large portion of an interactive application is spent dealing with the user interface. Reusing the user interface makes the application smaller but also makes it easier for the user to learn the application [16, 22].

Finally, we should provide a single-action path for each feature. We should use simple, consistent, and unique ways to invoke particular features. This approach is less confusing for the users, and there is less code for us to write. This approach also forces us to look carefully at the overall user-interface design [22].

2. Move Computation to the Server

The second design issue is to avoid executing computationally intensive tasks on small mobile devices. Instead, let a server computer run them for you. One of the alternatives to this design issue is to tie up the device (potentially making its user interface unresponsive) for several seconds or minutes). In many ways, this issue is similar to deploying a thin-client Web application where most of the logic is in the Web server, leaving the Web browser to handle the user interface [22].

Finding the right balance between what to do with the small mobile device and what to do on the server is tricky and depends on both the application and the device's connectivity. Obviously, a device that has a wireless radio can connect to a server more often than a device that has only cradle-based communication. But the cradle-based communication is faster and essentially free, while wireless communication can be slow and expensive. Therefore, even if you can connect to the server on needed basis, it might be time-prohibitive or cost-prohibitive to download large amounts of data [22].

Letting the server do some of the work does not have to be complicated. Even simple things that are done on the server can make a big difference in your application's responsiveness. For example, it is preferable to let the server sort data rather than downloading data from the server and sorting it on the device. The download time will not improve, but you will save the time that your application spent sorting the data after the download. Later, after changes have been made to the data, the application can resort the data and you will benefit because the data is already in sorted condition [22].

3. Use Less Memory at Run Time

As it is discussed above, the runtime memory capacity of a small mobile computing device can be quite limited. Some simple strategies are used to reduce the amount of runtime memory that is used in Java applications [4, 22].

i) Use Scalar Types

Each object that is used must be allocated from the runtime memory heap. There is no way to declare objects that are allocated on the stack. The object's constructor runs as part of that allocation process. Therefore, each object that is allocated has an impact on the application's performance as well as the amount of memory required. To reduce the number of objects that are allocated, it is preferable to use the scalar types (the non-object types such as **int** and **Boolean),** in place of objects [4, 22].

For instance, the methods of java.awt.Component define two variants of the setSize method:

public void setSize(int width, int height);

public void setSize(Dimension size);

The first variant takes only scalar types, while the second one requires to allocate a Dimension object. When the first variant is call ed, it generates a bit more code—but it avoids an object allocation. Besides, the second variant is defined as follows [4, 22]:

public void setSize(Dimension size){

setSize(size.width, size.height);

}

Although a call to the second variant is a bit cheaper in terms of byte code generation, it ends up executing more code in addition to incurring the object-allocation expense.

ii) Use Lazy Instantiation

Another technique to reduce overall and peak memory usage is to only allocate objects, as they are needed. This technique is usually referred as lazy instantiation. In this technique, checking for a null object reference, which is always initialized to a default value in Java, will be done [4, 22].

iii) Release Resources Early

This design issue makes sense to release resources such as database connections, network connections, files, and so on as soon as possible. This is equally important for small devices and desktop systems. Do not hang onto them longer than necessary.

Releasing resources are very important to free the resource for use by another application, and to enable the system to free any memory associated with that resource. In particular, do not depend on **finalizers** to free resources. **Finalizers** might never run; they are not even supported by some Java interpreters. Always provide methods for explicitly freeing the resources, and document their use. If your Java platform does support **finalizers**, however, it is still a good idea to define **finalizers** for each resource-using class just to ensure that the resources are really freed [4, 22].

4. Avoid Exceptions

Java's built-in support for exception handling is extremely convenient. However, exceptions are sometimes overused. In general, you want to reserve exceptions for unusual or unexpected (exceptional) situations [22].

Errors that are expected to occur in the normal course of running an application should be handled through other means. By avoiding exceptions, you can reduce the size of the class files and also reduce the number of objects that are allocated (because each exception throws an exception object) [22].

5. Performance

With a small device, performance is critical. It should be always considered when writing code. Here are a few strategies to explore the performance during designing applications for small devices.

i) Use Local Variables

It is generally slower to access class members than to access local variables. If you are using the same class member over and over, such as within a loop, it might make sense to assign the value to a temporary variable stored on the stack and to use that temporary variable in place of the class member. This optimization might not make sense when dealing with data that is shared by multiple threads. Using local variables also makes sense when dealing with arrays. Each time an array element is accessed, the Java interpreter performs a bounds check in order to ensure that the array index is valid. If you access the same array element more than once, it is better to store that array element in a local variable and access it from there. For example, instead of [4, 22]:

```
Char[] buf = ....; // get an array somehow
for( int i = 0; i < buf.length; ++i ){
if( buf[i] >= '0' && buf[i] <= '9' ){
....
} else if( buf[i] == '\r' || buf[i] == '\n' ){
....
}
}
```

Use a local variable to reduce the number of references to buf[i]:

```
for( int i = 0; i < buf.length; ++i ){
Char ch = buf[i];
if( ch >= '0' && ch <= '9' ){
....
} else if( ch == '\r' || ch == '\n' ){
....
```

```
}
}
```

ii) Avoid String Concatenation

Java makes it easy to build strings by concatenation. It is natural to do this kind of coding. Let's consider the following example below [4, 22]:

```
public String indent( String line, int spaces ){
String out = "";
for( int i = 0; i < spaces; ++i ){
out += " ";
}
return out;
}
```

From a performance point view, however, this coding is extremely poor. String concatenation involves creating a new ***StringBuffer*** object, calling its ***append*** method, and then calling its ***toString*** method in order to obtain the final string. Concatenation inside a loop (as shown earlier) can lead to the creation of many short-lived String and StringBuffer objects, which affects performance and also increases the application's peak memory usage. The better solution is to do most of the work yourself, as in the following example below [4, 22]:

```
public String indent( String line, int spaces ){
StringBuffer out = new StringBuffer();
for( int i = 0; i < spaces; ++i ){
out.append( ' ' );
}
return out.toString();
}
```

This simple change can drastically reduce the number of objects that are created.