Implementation and Analysis of Electronic Medical Records in Mobile Devices

Vikas Chintha

Problem Report Submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Master of Science in
Computer Science

Dr. James D. Mooney, Ph.D. Chair
Dr. Y.V. Reddy, Ph.D.
Dr. Arun Abrahim Ross, Ph.D.

Department of Computer Science

Morgantown, West Virginia
2006

Abstract

Implementation and Analysis of Electronic Medical Records in Mobile Devices

Vikas Chintha

Traditional Electronic Medical Record (EMR) had the following overheads i) physical space for the storage of records and for maintaining files, ii) prescriptions and other documents, Time Consuming to write, search, store and maintain records physically, iii) employers required for maintaining these records which results in human costs, iv) transcription costs which require physicians to enter the records manually, v) difficulty in retrieving the old records that were observed in the past.

The Electronic Medical Record software (EMR) aims to automate and simplify the patient record documentation, storage and retrieval process. It is an Internet based Electronic Medical Record System (EMRS) runs on the user's web browser. This enables universal accessibility from a desktop, tablet PC or handheld. Compared to a conventional windows application, porting this EMR in to Mobile devices EMR offers the user easy accessibility across various locations. The doctors can view or edit patient data from a desktop, PDA handheld device or tablet PC accordingly. The patient data is captured into forms customizing to each specialty such as Cardiology, Neurology etc. Finally the data can be accessed from a mobile web browser and data is displayed on the browser.

ACKNOWLEDGEMENTS

First of all, I thank the Chair person of committee Dr. James D. Mooney for his continuous support and guidance towards my Project. I appreciate for his technical ideas and relevant material provided to me.

I thank Dr. Y.V Reddy and Dr. Arun Abraham Ross for serving in my committee and being a great help, encouragement and support.

I thank Dr. Juggy for teaching the concepts of Web Services.

Finally, infinite regards to my loving parents and Sisters who always encouraged for attaining Master's Degree and having a successful career.

TABLE OF CONTENTS

Chapter 1 Introduction:

1.1 Objective of Problem Report……..1

1.2 Introduction……1

1.3 Advantages of EMR…….....3

1.3.1 Increases Revenue…….3

1.3.2 Reduces Expenses……..4

1.3.3 Reduces Risk…….....4

1.3.4 Improves Quality…….....4

Chapter 2 Concepts:

2.1 Wireless Application Protocol (WAP)……...6

2.1.1 WAP History……6

2.1.2 WAP Definition…….....6

2.1.3 WAP Architecture…….....6

2.1.4 Establishing WAP Network…….……6

2.1.5 Security in WAP……7

2.1.6 WAP Protocol Stack…….....…..7

2.1.6.1 Wireless Application Environment…….....8

2.1.6.2 Wireless Session Protocol…….....8

2.1.6.3 Wireless Transaction Protocol…..……..8

2.1.6.4 Wireless Transport Layer Security ……9

2.1.6.5 Wireless Datagram Protocol……...9

2.2 Wireless Markup Language (WML)…….10

2.2.1 Invention of WML……10

2.2.2 WML Structure…….11

2.2.3 Operation of WML……...……11

2.2.4 Capabilities of WML……....11

2.3 Servlets…….……12

2.4 JDBC Community…….…....13

2.4.1 JDBC working principle……..…….13

2.5 MIDlet…….……..16

2.6 Extensible Markup Language…….…..19

2.6.1 Origination of XML…….…….19

2.6.2 XML vs HTML……..……....19
2.6.3 XML with Document Type Definition ……....20
2.6.4 XML Parser…….……..20
2.7 SOAP……...…….21

Chapter 3 Design

3.1 Design…….……..21
3.1.1 Database Tables……21
3.1.2 JDBC Connection…….21
3.1.1.1 Opening a Connection…….21
3.1.3 Compiling the Servlet…….....22
3.1.4 Deploying the Servlet ……...…23
3.1.5 Hooking the Servlet with a MIDlet……...25

Chapter 4 Security

4.1 Security Issues……..30
4.2 WSA High Level Architecture……..30
4.2.1 Invocation model and data flow as standard Web service……....31
4.2.2 Consuming Web Services…….32
4.2.3 Generating a JSR 172 WSA stub……..32
4.2.4 JSR Features……..33
4.2.5 XML Encoding……...…….....33
4.2.6 Portability and Interoperability……..……..……..33
4.3 Implementation of Web Services in Mobile Devices…….…..……34
4.4 Web Service J2ME Stub…….....34
4.5 Calling a Web service…….....37

Conclusion……..38
Future Work……....39
Bibliography……...40
Index…….42

Chapter 1:

1.1) Objective of Problem Report:

The main objective of this problem report is to find a way to translate or transform the data to be a Wireless Markup Language (WML) format that is accessible from a Wireless Application Protocol (WAP) phone. Therefore, carefully analyzing the medical data in the database and retrieving the data in to the mobile phones along with considering the security issues becomes the important tasks of this project. In order to consider the security issues we use the concept of web services where in the medical data is stored in xml files and data is retrieved from mobile phones using SOAP protocol. The result of this project is that we should be able to browse Electronic medical data from WAP phone emulators.

1.2)Introduction:

Electronic Medical Records (EMR) refers to a set of software products designed for Healthcare industry. Previously, all the prescriptions, appointments, payments, patient meetings were done manually but this software allows to automate all the sources that were done manually earlier. The main goals of EMR are to improve the production and smooth the progress of the quality of care. The most important feature is the ease of the program and the simplicity at which it can be used. Beyond that, the program should be shared with doctor and staff and no one should be intimidated by the complexities of the system. In addition, the software product should offer flexibility and custom features, so that everyone feels comfortable not only with the technical aspects of the program, but also the way in which documents are created. The doctor should be able to customize the program to reflect the way they practice and not have the software program dictate practice protocol. Doctors have a special way to take care of patients in a certain way, and the software and computer has to fit into the work flow, not be perceived as extra work, and not interfere with the doctor-patient interaction. Another mandated feature is physical portability, resulting in, at the point of care documentation. This equates to

avoiding subsequent time, often after office hours, reviewing and updating records on PC or chart. Ultimately, the software and hardware must be able to facilitate communication and save time.

An EMR system is a patient information network as well as an office management tool. It can be used as the primary source for patient records and it is a modern medical database with the same structure, scope and information as a paper-based record. All the medical data can be stored in a huge database. The database is filled with details about the patient and their care. The EMR database has the power to hold data, but even more important, is the ability to retrieve data back out or extract it visibly and or print it out for a third party. There can be volumes of information in a given database, but the doctor may only want to see one specific part. Good EMR software has the equivalent of flip switches or search engines, which allow the viewer to see only what is applicable and or desired. All information needs to be available but necessary information is what is needed. The quick access to and retrieval of information are what doctors consider most impressive. Retrieving and viewing data by date or visit are simple and straightforward when the user interfaces are properly designed. It enables immediate access to information.

Computer-based and Mobile based patient records can remedy the inherent flaws of the conventional paper system through the improvements in accessibility, cost savings, quality, and marketability.

EMR systems create cost-effectiveness within a budget by virtually eliminating dictation and transcription costs, thereby avoiding thousands of dollars in transcription fees. With transcribers out of the picture and with the ease of data entry, fewer personnel are needed. Data entry time may be halved, so fewer people are required to work.

An EMR system minimizes mistakes. There is no need to decipher bad script because completely legible records are produced. Information will never be lost, (backup) no duplicate testing ordered, nor any unnecessary repeat treatment offered.

Not all users will "realize" the immediate value of an EMR. In fact, in a hectic and hurried setting the value is lost when compared to paper, at least if don't extrapolate beyond the present. A featured value is when the patient returns and you don't have to thumb through all those pages in a paper chart. The tangible value is in Data Mining or having all the data in one place and being able to visibly extract or filter the data which is desired, in a rapid and expedient manner. With respect to third party payers, the value consists of being able to manipulate, retrieve and output the data collected, ideally at the point of care. EMR system improves communication by making records available to multiple people, simultaneously. Switching to an EMR system can save doctor hours, which would otherwise have been spent updating patient files. All charts are kept current and provide more detailed patient information than paper-based records. A doctor's extra time benefits patients. Less time is spent waiting and more quality time is available for patient care. Productivity also increases as the practice becomes more streamlined. With a well-managed practice, a doctor may increase the patient base because better treatment is provided.

1.3) Advantages of EMR:

1.3.1) Increases Revenue:

- Allows doctors to see more patients without decreasing patient doctor encounter time.
- Increases collections through improved management and eligibility data
- Optimizes coding and improves managed care payments/bonuses
- Promotes practice growth via re-dedication of office space previously devoted to paper records that can now be utilized for patient care activities
- Enhances revenue through disease management functionality—EMR reporting functions track patient follow-up and compliance with health maintenance protocols

1.3.2) Reduces Expenses:

- Minimizes transcription costs.
- Reduces supply, storage and chart access costs related to the maintenance of paper records
- Increases employee productivity

1.3.3) Reduces Risk:

- Improves quality of documentation
- Increases accuracy of shared information
- Maintains security and integrity of data
- Improves patient safety by checking for possible drug interactions

1.3.4) Improves Quality:

- Increases access to patient information at the point of care, including administrative data within practice management systems
- Improves outcomes through patient health tracking and clinical alerts
- Speeds responses to patients and increases patient satisfaction
- Promotes proactive care management and increased collaboration among providers, including sharing of electronic medical records
- Provides diagnosis support, clinical reference information, and patient education

Chapter 2

Concepts:

2.1) WAP

2.1.1) WAP History: It was established in 1997 by Ericsson, Motorola, Nokia and Unwired Planet and now has over than 90 members, which together represent over 90 % of the global mobile market.

The objectives of the WAP Forum are:

- To bring Internet content and advanced data services to digital cellular phones and other wireless terminals.
- To create a global wireless protocol specification that will work across differing wireless network technologies.
- To enable the creation of content and applications that scale across a very wide range of bearer networks and device types.
- To embrace and extend existing standards and technology wherever appropriate.

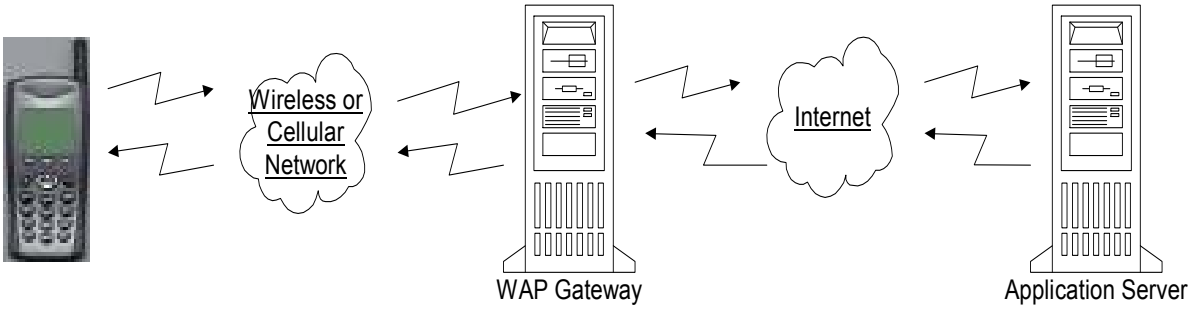
2.1.2) WAP Definition: Wireless Application Protocol enables easy fast delivery of relevant information and services to mobile users with wireless terminals that has small displays and data transfer capabilities. It is a specification for a set of communication protocols to standardize the way in which mobile devices like Pda's, Pocket Pcs use Internet access.

2.1.3) WAP Architecture: The WAP programming model is similar to the Internet model. This provides some benefits, including the familiar architecture and programming environment as well as the ability to use existing tools. Not only Internet standards have been used where possible, but also optimizations and extensions have been made.

WAP specifies two essential elements of wireless communication: i) an end-to-end application protocol ii) application environment based on a browser. The application protocol is a layered communication protocol that is embedded in each WAP user agent. The network side includes a server component implementing the other end of the

protocol that is capable of communicating with any WAP user agents. The role of the server component is also to act as a gateway to route the request of a user agent to an application server. Physically the gateway can be located in a telecom or a computer network, in order to build a bridge between the two different networks.

WAP Architecture:



WAP provides a standard to enable running the same application in different user agents. The standard consists of a browser and a script interpreter. The browser acts like a conventional web browser but instead of HTML, it understands content written in Wireless Markup Language or briefly, WML. Also instead of JavaScript interpreter, WAP browser has a build in script interpreter for running applications written in a script language called WMLScript. Furthermore the browser includes some libraries to allow the application to use certain services of the user agents. Both WML and WMLScript are designed to be used in wireless networks with narrowband data transfer rates, so they are both binary encoded to optimize the data transmission efficiency.

2.1.4) Establishing WAP Network: To establish a connection between a Wap client and a server, the user presses the phone key that has a URL request assigned to it and the user agent sends the URL request to a WAP gateway using the WAP protocol. The WAP gateway then transmits a conventional HTTP request with the specified URL to the web server. Once the web server gets the request, it acts according to the request: it sends the URL file with a HTTP header back to the WAP gateway or it launches the application

that the URL specifies, and then sends the output to the WAP gateway. The WAP gateway in turn verifies the HTTP header and the WML content, encodes them into binary form and sends a WAP response to the user agent, which displays the WML content to the user.

If the server does not understand WAP content eg. WML, separate filters are used to translate the WWW content into WAP content. For example, extra HTML filter is used between the server and the WAP proxy to translate HTML to WML.

2.1.5) Security in WAP: WAP can provide end-to-end security between the two endpoints. If the browser and the server explicitly express the need to use end-to-end security, they must communicate directly in order to be able to use the WAP protocols. Also, if the WAP proxy is trusted or it is known to locate in the same physical place as the WAP client, the end-to-end security can be achieved.

When one specifies an URL, all the variables that the URL uses are accessible. To overcome this security risk, WML provides elements controlling access control. The most common security risks occur when a WAP client voluntarily requests a harmful service masking it as a legitimate service. This may occur by directly accessing a card in the service that has sensitive operations, by gaining access to variables in the service that have confidential information or by clearing the variables. In order to avoid this, the service should use scripts to check that the request comes from an allowed user.

2.1.6) WAP Protocol Stack: WAP Protocol stack is layered in a way similar to OSI protocol stack, in which each of the layers is accessible by the layers above, as well as other services and applications. There are four major layers in the WAP protocol:

- Session layer (WSP)
- Transaction layer (WTP)
- Security Layer (WTLS)
- Transport layer or datagram layer (WDP)

2.1.6.1) Wireless Application Environment (WAE)

The Wireless Application Environment (WAE) is the top-most level in the WAP architecture. It is based on WWW and Mobile Telephony technologies. The primary objective of the WAE is to provide the operators and service providers an interoperable environment on which they can build applications and services which, in turn, can be used in a wide variety of hand-held client terminals. WAE includes the micro-browser that contains functionality for using not only WML and WML Script as previously stated, but also Wireless Telephony Application, namely (WTA and WTAI) -telephony services and programming interfaces as well as content formats including well-defined data formats, images, phone book records and calendar information.

2.1.6.2) Wireless Session Protocol (WSP)

The Wireless Session Protocol provides the Wireless Application Environment a consistent interface with two services: connection-oriented service to operate above the Transaction Layer Protocol (WTP) and a connectionless service that operates above either secure or non-secure datagram service (WDP).

Currently the protocols of the WSP family provide HTTP/1.1 functionality and semantics in a compact encoding, long lived session state with session suspend and resume capabilities, a common facility for reliable and unreliable data push as well as a protocol feature negotiation. These protocols are optimized to be used in low-bandwidth bearer networks with relative long latency in order to connect a WAP client to a HTTP server. [

2.1.6.3) Wireless Transaction Protocol (WTP)

The Wireless Transaction Protocol operates efficiently over either secure or non-secure wireless datagram networks. It provides three different kinds of transaction services, namely unreliable one-way, reliable one-way and reliable two-way transactions. This layer also includes optional user-to-user reliability by triggering the confirmation of each received message. To reduce the number of messages sent, the feature of delaying acknowledgements can be used.

2.1.6.4) Wireless Transport Layer Security (WTLS)

The Wireless Transport Layer Security protocol is based on Transport Layer Security (TLS) or formerly known as Secure Sockets Layer (SSL). It is designed to be used with other WAP protocols and to support narrow-band networks. It uses data encryption with a method that is negotiated at the start of the session to provide privacy, data integrity, and authentication and denial-of-service protection. The latter is needed in cases when data is replayed or not properly verified. When that happens, WTLS detects the misuse and rejects the data in order to make many typical denial-of-service attacks harder to accomplish.

It is up to the applications to enable or disable WTLS features. Whether that happens, it depends to their security requirements and the characteristics of the underlying network, namely, does it use security services at the lower layer.

2.1.6.5) Wireless Datagram Protocol (WDP)

The Wireless Datagram Protocol in WAP architecture covers the Transmission Layer Protocols in an Internet model. As a general transport service, WDP offers to the upper layers an invisible interface independent of the underlying network technology used. In consequence of the interface common to transport protocols, the upper layer protocols of the WAP architecture can operate independent of the underlying wireless network. By letting only the transport layer deal with physical network-dependent issues, global interoperability can be acquired using mediating gateways.

The bearer services, over which WAP is designed to operate, include short message, circuit-switched data and packet data services. Since the bearers offer different types of quality of service with respect to throughput, error rate and delays, the WDP is designed to either compensate for or tolerate these changes. Also, WDP lists all the bearers that are supported and the techniques applied when transmitting data over a certain bearer. These lists will change with new bearers being added as the wireless market grows.

2.2 Wireless Markup Language (WML)

2.2.1 Invention of WML

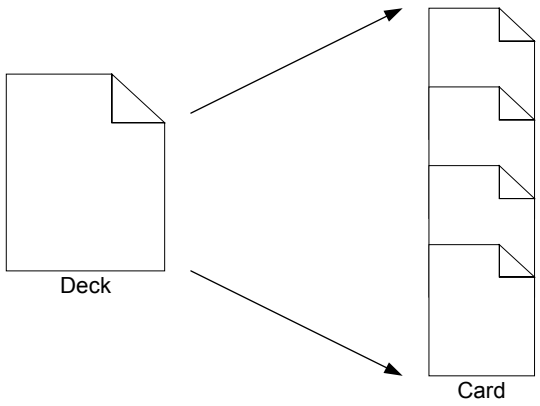
WML is a markup language based on XML (Extensible Markup Language). We will introduce XML in the next section. WML is designed for specifying user interface behavior and displaying content on wireless devices.

An industrial organization, the Wireless Application Protocol (WAP) Forum, is devoted to developing open standards for wireless communication. This standard is the formal specification for WML.

With WML, we can wrap the data content to produce readable output on the device screen. An application designer does not need to know whom he or she is talking to, how big the screen of the client is, or how many keys the keyboard has. He simply knows that a tiny screen is available.

2.2.2 WML Structure

As the diagram shows, a WML document structure is a deck grouped by one or more cards. When accessing a wireless web site, the content will be sent back as a deck. The user will read the first card of the WML document, and then move back or forward between different cards.



WML Structure Diagram

2.2.3 Operation of WML

As the previous section described, when the user accesses a WAP site, it sends back a deck; the user is shown the first card, reads the content, possibly can enter some information, and then moves to another card, the choice of which is dependent on user's actions. The card that is displayed is left to the browser. Different browser will prompt the user for input in different ways. Depending on the device capabilities, the Browser decides how best to present the content

2.2.4 Capabilities of WML

The following items are features of WML

- Text support: Text in a card including bold, Italics, underlined, etc., line breaks and tables are supported.
- International support: The WML uses Unicode to be character set, which has enough bit to present international characters. (16 bits to represent each character)
- Image support: WBMP (Wireless Bitmap) format has been created for displaying images.
- User input: User can input elements into Cards.
- Variables: Variables can be put in the WML code to keep track of hidden information or to manipulate user input.
- Navigation and history stack: WML provides common navigation and history functionalities.
- Optimization for narrow-band: WML has been designed to the high-latency and narrow-band characteristics.

2.3 Servlets:

Servlets are Java classes which service HTTP requests. The **Java Servlet API** allows a user to add dynamic content to a Web server using Java platform. The generated content is generally HTML, but may be other data such as xml.

Advantages Of Servlets:

- **Efficient.** With servlets, the Java Virtual Machine stays up, and each request is handled by a lightweight Java thread, not a heavyweight operating system process. With servlets, however, there are *N* threads but only a single copy of the servlet class. Servlets also have more alternatives for optimizations such as caching previous computations, keeping database connections open.
- **Convenient.** Servlets have an extensive infrastructure for automatically parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, tracking sessions, and many other such utilities.
- **Powerful.** Servlets can talk directly to the Web server. This simplifies operations that need to look up images and other data stored in standard places. Servlets can also share data among each other, making useful things like database connection pools easy to implement. They can also maintain information from request to request, simplifying things like session tracking and caching of previous computations.
- **Portable.** Servlets are written in Java and follow a well-standardized API. Consequently, servlets written for, say I-Planet Enterprise Server can run virtually unchanged on Apache, Microsoft IIS, or WebStar. Servlets are supported directly or via a plugin on almost every major Web server.
- **Inexpensive.** There are a number of free or very inexpensive Web servers available that are good for "personal" use or low-volume Web sites. However, with the major exception of Apache, which is free, most commercial-quality Web servers are relatively expensive. Nevertheless, once you have a Web server, no matter the cost of that server, adding servlet support to it (if it doesn't come preconfigured to support servlets) is generally free or cheap.

2.4 JDBC Connectivity:

JDBC is an API developed by Sun Microsystems that provides a standard way to access data using the Java programming language. Using JDBC, an application can access a variety of databases and run on any platform with a Java Virtual Machine. It isn't necessary to write separate applications to access different database systems (Oracle and Sybase, for example). Using JDBC allows you to write one application that can send SQL statements to different database systems. SQL is the standard language for accessing relational databases.

The JDBC API defines a set of Java interfaces that encapsulate major database functionality, such as running queries, processing results, and determining configuration information. Because JDBC applications are written in Java, applications work on any platform.

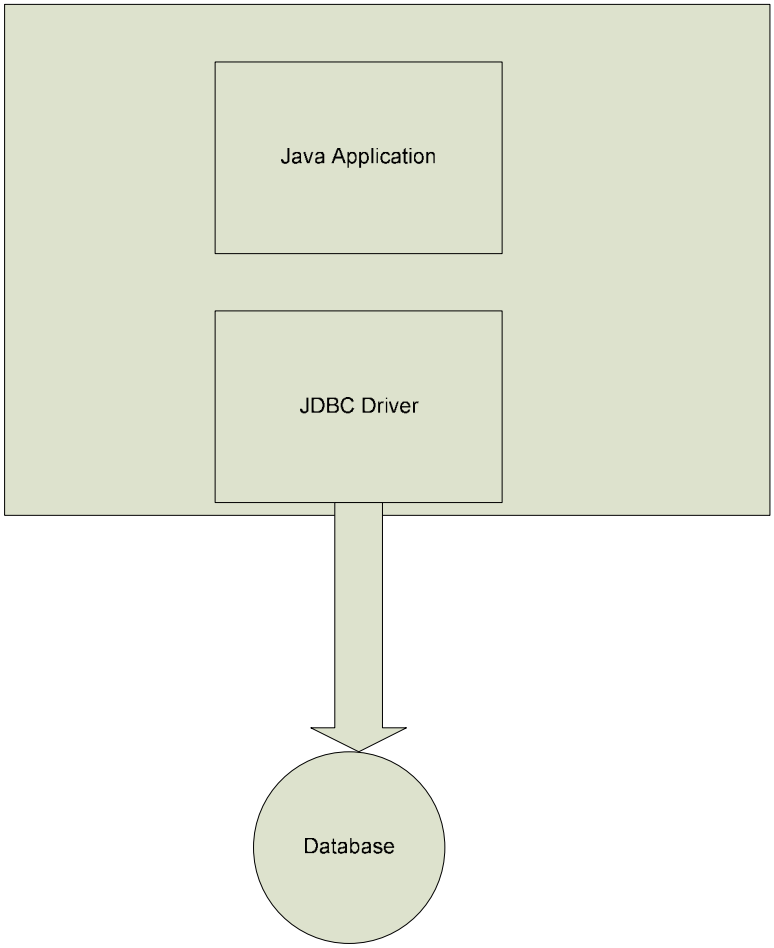
2.4.1 JDBC working principle:

Simply, JDBC makes it possible to do the following things within a Java application:

- Establish a connection with a data source
- Send queries and update statements to the data source
- Process the results

The following figure shows the components of the JDBC model.

JDBC



The Java application calls JDBC classes and interfaces to submit SQL statements and retrieve results.

The JDBC API is implemented through the JDBC driver. The JDBC Driver is a set of classes that implement the JDBC interfaces to process JDBC calls and return result sets to a Java application. The database (or data store) stores the data retrieved by the application using the JDBC Driver.

The main objects of the JDBC API include:

A **DataSource** object is used to establish connections. Although the Driver Manager can also be used to establish a connection, connecting through a DataSource object is the preferred method.

A **Connection** object controls the connection to the database. An application can alter the behavior of a connection by invoking the methods associated with this object. An application uses the connection object to create statements.

Statement, **PreparedStatement**, and CallableStatement objects are used for executing SQL statements. A PreparedStatement object is used when an application plans to reuse a statement multiple times. The application prepares the SQL it plans to use. Once prepared, the application can specify values for parameters in the prepared SQL statement. The statement can be executed multiple times with different parameter values specified for each execution. A CallableStatement is used to call stored procedures that return values. The CallableStatement has methods for retrieving the return values of the stored procedure.

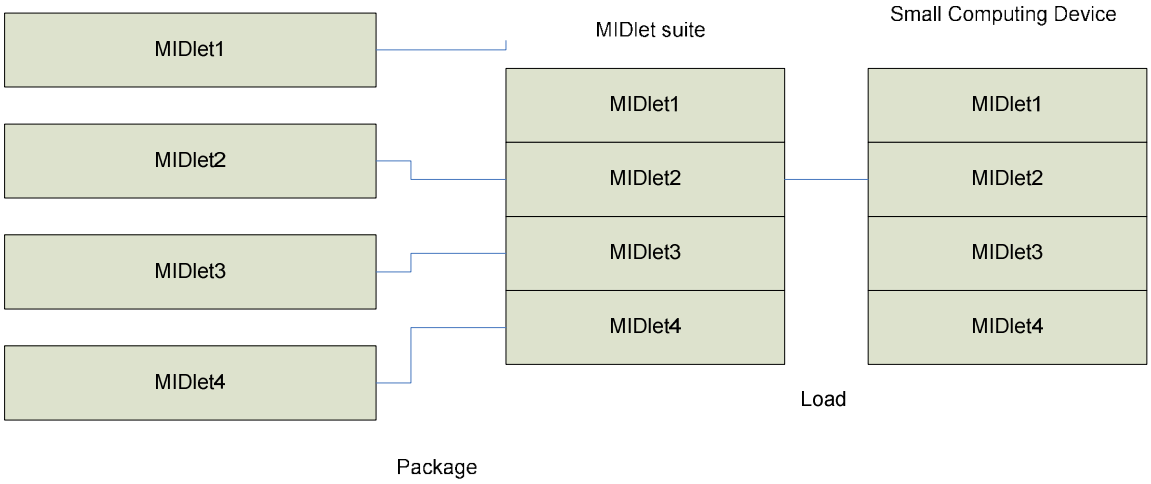
A **ResultSet object** contains the results of a query. A ResultSet is returned to an application when a SQL query is executed by a statement object. The ResultSet object provides methods for iterating through the results of the query.

2.5 MIDlet

A MIDlet is a J2ME application designed to operate on an MIDP small computing device. A MIDlet is defined with at least a single class that is derived from the javax.microedition.midlet.MIDlet abstract class. Developers commonly bundle related MIDlets into a MIDlet suite, which is contained within a single package and implemented simultaneously on a small computing device. All MIDlets within a MIDlet suite are considered a group and must be installed and uninstalled as a group.

Members of a MIDlet suite share resources of the host environment and share the same instances of java classes and run within the same JVM. This means if three MIDlets from the same MIDlet suite run the same class, only one instance of the class is created at the time in the Java Virtual Machine. A key benefit of the relationship among MIDlet suite members is that they share the same data, including data in persistent storage such as user preferences.

Sharing data among MIDlets exposes each MIDlet to data errors caused by concurrent read/write access to data. This risk is reduced by synchronization primitives on the MIDlet suite level that restricts access to volatile data and persistent data. However, if a MIDlet uses multithreading, the MIDlets are responsible for coordinated access to the record store.



MIDlets are packaged into MIDlet suites, which are loaded in a small computing device

Data cannot be shared between MIDlets that are not from the same MIDlet suite because the MIDlet suite name is used to identify data associated with the suite. A MIDlet from a different from a different MIDlet suite is considered an unreliable source.

A MIDlet suite is installed, executed, and removed by the application manager running on the device. The manufacturer of the small computing device provides the application manager. Once a MIDlet is installed, each member of a MIDlet class is given access to classes of the JVM and CLDC by the application manager. Likewise, a MIDlet can access classes defined in the MIDP to interact with the user interface, network and persistent storage.

The application manager also makes the Java archive (JAR) file and the Java application manager descriptor (JAD) file available to members of the MIDlet suite.

MIDlet Programming vs. J2SE Programming

Programming a MIDlet is similar to creating a J2SE application in that you define a class and related methods. However, MIDlet is less robust than J2SE application because of the restrictions imposed by the small computing device.

A MIDlet class must contain three abstract methods that are called by the application manager to manage the life cycle of the MIDlet. These abstract methods are startApp( ), pauseApp( ) and destroyapp( ).

The startApp( ) method is called by the application manager when the MIDlet is started and contains statements that are executed each time the application begins execution. The pause( ) method is called before the application manager temporarily stops the MIDlet. The application manager restarts the MIDlet by recalling the startApp( ) method. The destroy( ) method is called prior to the termination of the MIDlet by application manager.

```
public class BasicMidlet extends MIDlet
{
```

```
  public void StartApp( )
    {
     }
  public void pauseApp( )
    {
     }
public void destroyApp( Boolean unconditional)
   {
   }
}
```

CDC (Connected Device Configuration) implements full J2SE available, but CLDC implements a stripped- down J2SE because of the limited resources in small computing devices.

Floating point math is the most notable missing feature of J2ME. Floating-point math requires special processing hardware to perform floating-point calculations. These small computing devices lack such hardware and therefore are unable to process floating point calculations.

There is an absence of finalize ( ) method. Here the computing devices are too scarce to process the finalize ( ) method.

Another difference is the reduced number of Error Handling exceptions that are supported in J2ME. Exception Handling drains system resources, which are previous in a small computing device and therefore the primary reason for trimming the number of error handling exceptions.

Another feature lacking in the JVM is the ThreadGroup class. WE cannot group threads. All threads are handled at object level. Also users cannot call other programming languages methods and API's primarily because of the memory requirements to execute

such calls. Other features of J2SE that are missing from J2ME are weak references and the Reflection classes.

The standard JVM uses file verification to protect applications from malicious code through the use of a security manager. However, this process is replaced with a two-step process because of the limited resources available on small computing devices. The first step is called preverification and occurs outside the small computing device prior to loading the MIDlet. Preverification requires that additional attributes called stack maps are inserted into a class file by software before the second step runs. Stack maps describe the MIDlet's variables and operands located into the interpreter stack.

2.6 Extensible Markup Language (XML)

2.6.1 Origination of XML

In order to provide content to different types of client without the need to create many different copies of each page, the World Wide Web Consortium (W3C) invents Extensible Markup Language (XML) to be universal format for structured documents and data on the Web. Documents in XML format is easy to read and write. Furthermore, XML is an excellent format for exchanging data among different applications. XML defines semantic tags that break a document into parts.

2.6.2 XML vs. HTML

In the early days of the Internet, HTML was created with the intention of specifying the content to be displayed, leaving decisions as to how to display the content to the browsers. However, nowadays what is encapsulated in an HTML page is much more than the content.

HTML sticks on certain defined tags. Once users want to use some other format of tags, they have to wait till next version of HTML. For XML, there is no such restriction exists. It allows users to define their own tags according to some principles described in Document Type Definition (DTD). Other user can understand the meaning of XML document via DTD. This makes data more flexible and extensible

2.6.3 XML with Document Type Definition

XML can contain arbitrary tags in a document, but it does not define the rules of tags. XML will follow up a Document Type Definition (DTD), which is a set of syntax rules for tags. It tells us what tags we can use in a document, which tags can appear inside other ones, what order they should appear in, or which tags have attributes.

2.6.4 XML Parser

Once we access an XML document, how does the browser read the document? A tool for reading an XML document is called an XML parser or formally called XML processor (9). It will parse data to an application for publishing, editing, or displaying. Microsoft or Netscape has included XML parsers in their browsers.

WAP browsers do not include XML parsers. Users have to use XML parser to retrieve data before sending it to the browser.

2.7 SOAP

Definition: SOAP is a simple XML based protocol to let applications exchange information over HTTP or in a simple manner, SOAP is a protocol for accessing a Web Service.

It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses. SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings defined in this document describe how to use SOAP in combination with HTTP and HTTP Extension Framework.

A major design goal for SOAP is simplicity and extensibility. Such features include

- Distributed garbage collection
- Batching of messages
- Objects-by-reference (which requires distributed garbage collection)
- Activation (which requires objects-by-reference)

Chapter 3

3.1 Design: Design needs these parts for the implementation

1) Database tables
2) Servlets
3) JDBC
4) Apache Tomcat Server
5) Wireless Tool kit

3.1.1 Database Tables: Database has four tables which have the information of Patient, Doctors, Patient Transcriptions and Patient Medication.

3.1.2 JDBC Connection: A Connection object represents a connection with a database. A connection session includes the SQL statements that are executed and the results that are returned over that connection. A single application can have one or more connections with a single database, or it can have connections with many different databases.
A user can get information about a Connection object's database by invoking the Connection.getMetaData method. This method returns a DatabaseMetaData object that contains information about the database's tables, the SQL grammar it supports, its stored procedures, the capabilities of this connection, and so on.

3.1.1.1 Opening a Connection

The general way to establish a connection with a database is to call the method DriverManager.getConnection. This method takes a string containing a URL. The DriverManager class, referred to as the JDBC management layer, attempts to locate a driver that can connect to the database represented by that URL. The DriverManager class maintains a list of registered Driver classes, and when the method getConnection is called, it checks with each driver in the list until it finds one that can connect to the database specified in the URL. The Driver method connect uses this URL to actually establish the connection.
The following code exemplifies opening a connection to a database located at the URL jdbc:odbc:wombat with a username and password:

String url = "jdbc:odbc:wombat";
Connection con = DriverManager.getConnection (url, username, password);

The JDBC 2.0 Standard Extension API provides the DataSource interface as an alternative to the DriverManager for establishing a connection. When a DataSource class has been implemented appropriately, a DataSource object can be used to produce Connection objects that participate in connection pooling and/or Connection objects that can participate in distributed transactions.
An application uses a Connection object produced by a DataSource object in essentially the same way it uses a Connection object produced by the DriverManager. If the Connection object is a pooled connection, an application should include a finally block to assure that the connection is closed even if an exception is thrown. That way a valid connection will always be put back into the pool of available connections.
If a Connection object is part of a distributed transaction, an application should not call the methods Connection.commit or Connection.rollback, nor should it turn on the connection's auto-commit mode. These would interfere with the transaction manager's handling of the distributed transaction.
Thus, jdbc connection is established and program is run using java application along with a server and data is displayed using a url to tomcat server.
The next step is compiling the servlet
3.1.3 Compiling the Servlet: Compiling servlet code is pretty much the same as for other Java development, except for an important twist. Because the servlet API is not a core part of the Java SE platform, you'll need to add it to CLASSPATH before you can compile servlets.
The servlet API is contained in common/lib/servlet.jar under the Tomcat root directory. Add this file to CLASSPATH and user will be able to compile programname.java using javac. User 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-4.1.31\common\lib\servlet.jar

C:\>javac progname.java

3.1.4 Deploying the Servlet

To deploy a servlet, knowledge about web applications is important. A web application is a collection of static content, like HTML and image files, servlets, and other resources that can be made accessible via a web interface. Tomcat comes with several web applications already installed. Create a new web application and place servlet inside the directory.

First, create the web application, since there is already created a new directory inside webapps called midp, where we saved the servlet source code. Now we 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 appopriate and close the tag inline.

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

Once finished adding the context entry, save the file.

What these steps do 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.

Now that web application is created, fill it up. Web applications have a standard directory structure, mandated by the servlets specification. The essential piece of a web application is a web.xml file that describes the various parts of the web application. This file lives in a standard location in every web application; it's always stored as WEB-INF/web.xml create a web.xml file for the new application. The next step is to make the servlet accessible to the outside world. Then know the class name of the servlet, servletname, and make it available under a path like /progname. 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/progname. 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>progname</servlet-name>
    <servlet-class>progname</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>progname</servlet-name>
    <url-pattern>/progname</url-pattern>
  </servlet-mapping>
</web-app>
```

This file tells Tomcat to map the servlet called progname to the path /progname. The **servlet-name** is internal to web.xml; it links the **servlet** element to the **servlet-mapping** element.

Now we have saved our servlet source code in a standard directory underneath WEB-INF called classes. This is where Tomcat expects to find servlet class files, so when we compiled the source code, the servlet class was stored in the right place.

Our servlet is now deployed in the new web application we created, but note that we must restart Tomcat to have it recognize the changes we made in server.xml.

http://localhost:8080/midp/progname. We should see the output of progname.

Next Step is to hook the servlet with a MIDlet.

3.1.5 Hooking the servlet with a MIDlet:

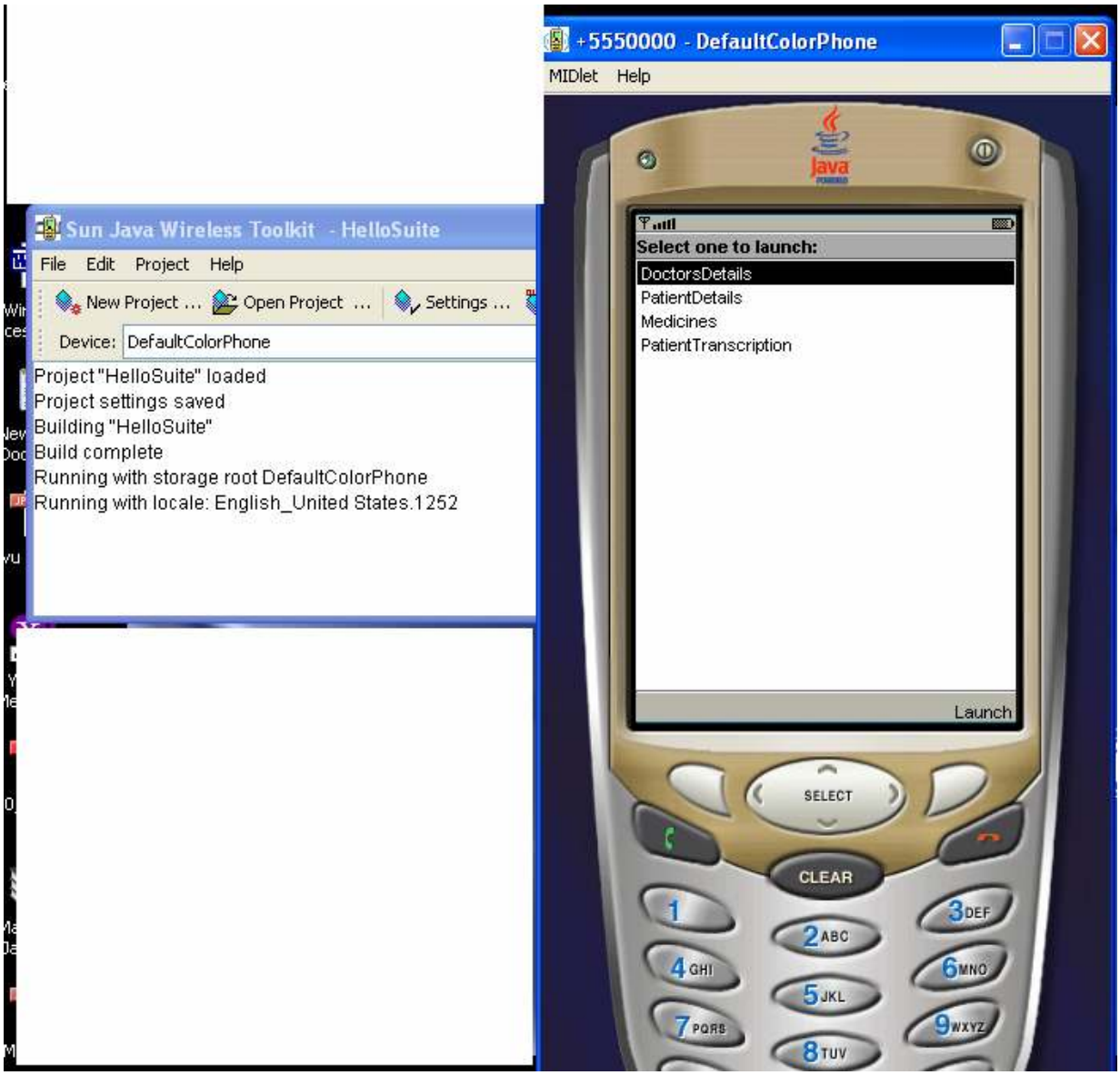
Now that we have a development environment that supports both MIDP and servlets, we need to hook the two worlds together to create an end to end Java application. MIDlets can connect to the world at large via HTTP, and the servlet is available to the world at large via HTTP, so MIDlet can connect to the servlet.

Start **KToolbar** (part of the Sun Java Wireless Toolkit) and open the MIDlet program. We need to create a new MIDlet that connects to servlet, retrieves its output, and displays it.

There are two other things to configure to get MIDlet working. First, we need to tell the toolkit about this new MIDlet. Click on **Settings...**, then select the **MIDlets** tab. Click on **Add** and fill in "MIDlet program name" for both the MIDlet name and class name. we can leave **Icon** blank. Click on **OK** and we need to verify for MIDlet program name listed.

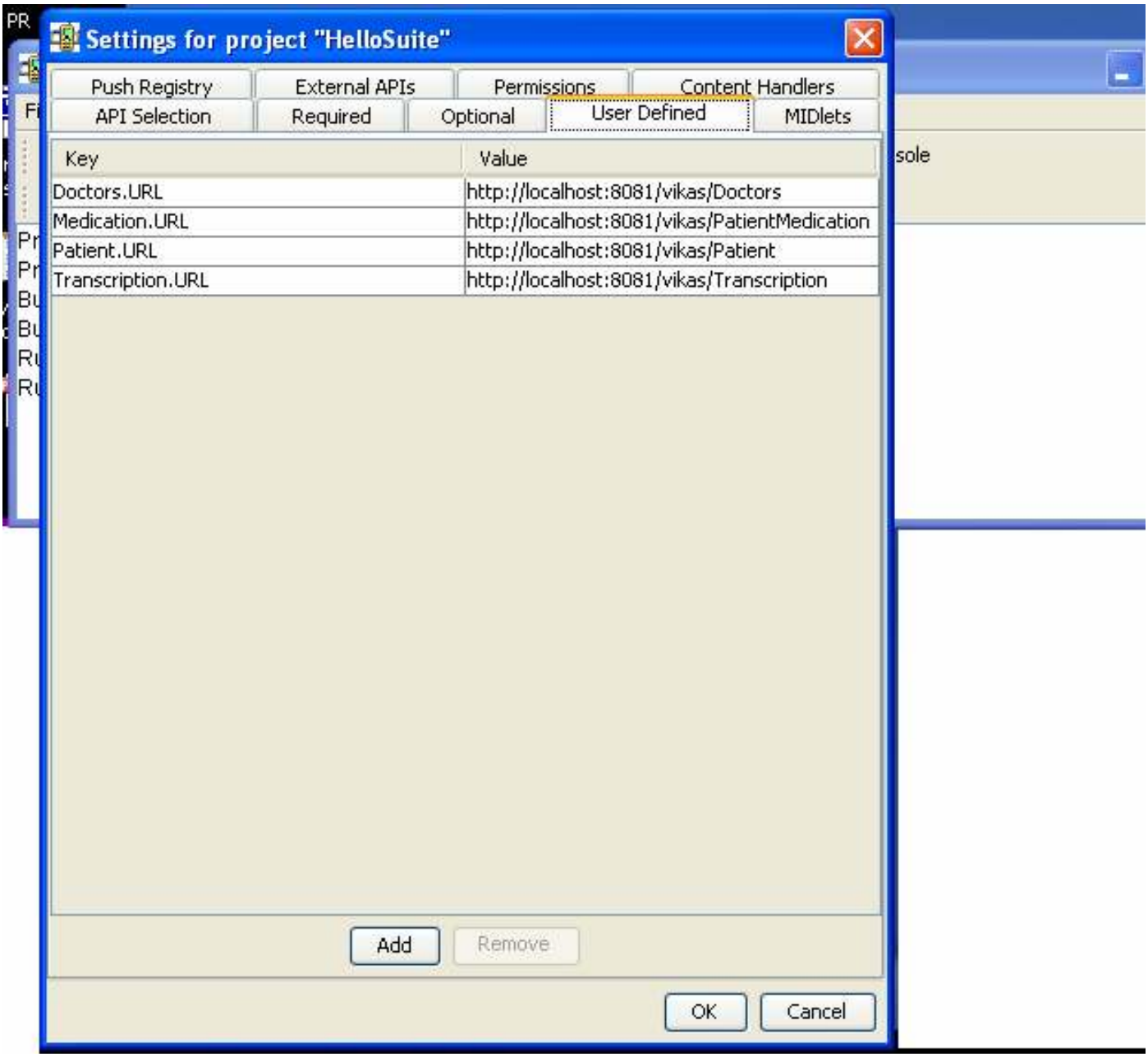
Next, you need to define a system property that MIDlet program uses as the URL for its network connection. (This property is retrieved in the third line of the connect () method of the MIDlet source code.) In the toolkit, click on **Settings...**, and then select the **User Defined** tab. Click on the **Add** button. Fill in the property name as **MIDlet.URL**; the value should be the URL that invokes Servlet, the same URL we used in a browser to test the servlet. When it is finished, click on **OK** to dismiss the project settings window.

Now, in the Sun Java Wireless Toolkit, click on **Build** to build the project. Assuming we don't see any error messages, we are now ready to test the application. Make sure your server is running first. Then click on **Run** and select MIDlet program. Select the **Connect** command. If everything goes well, MIDlet will invoke Servlet and display the results on the device emulator screen.

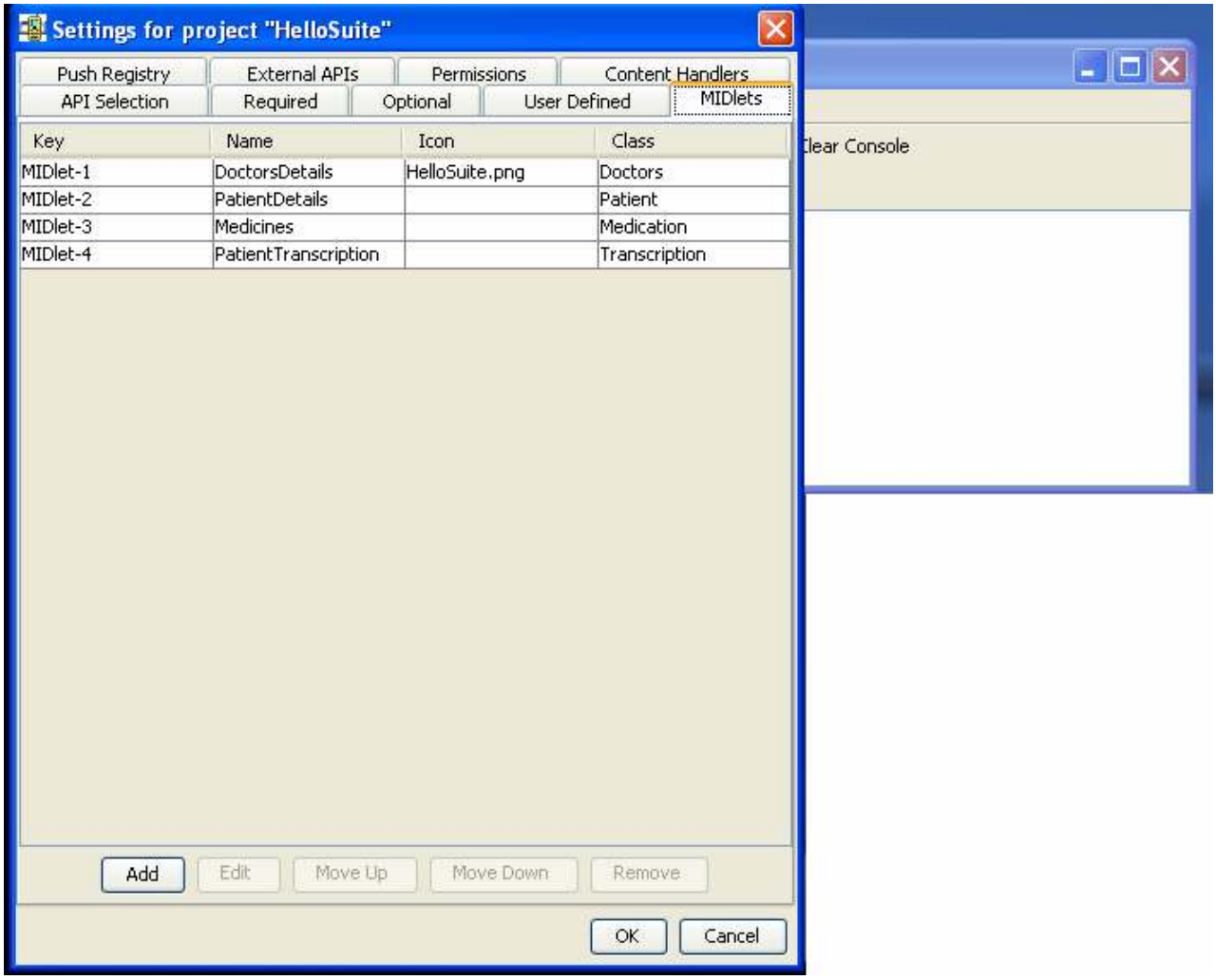


The above screen shot shows the main menu for the patient details, doctor details, medicines and patient transcriptions.

The following screen shot shows the urls for each of the MIDlet.

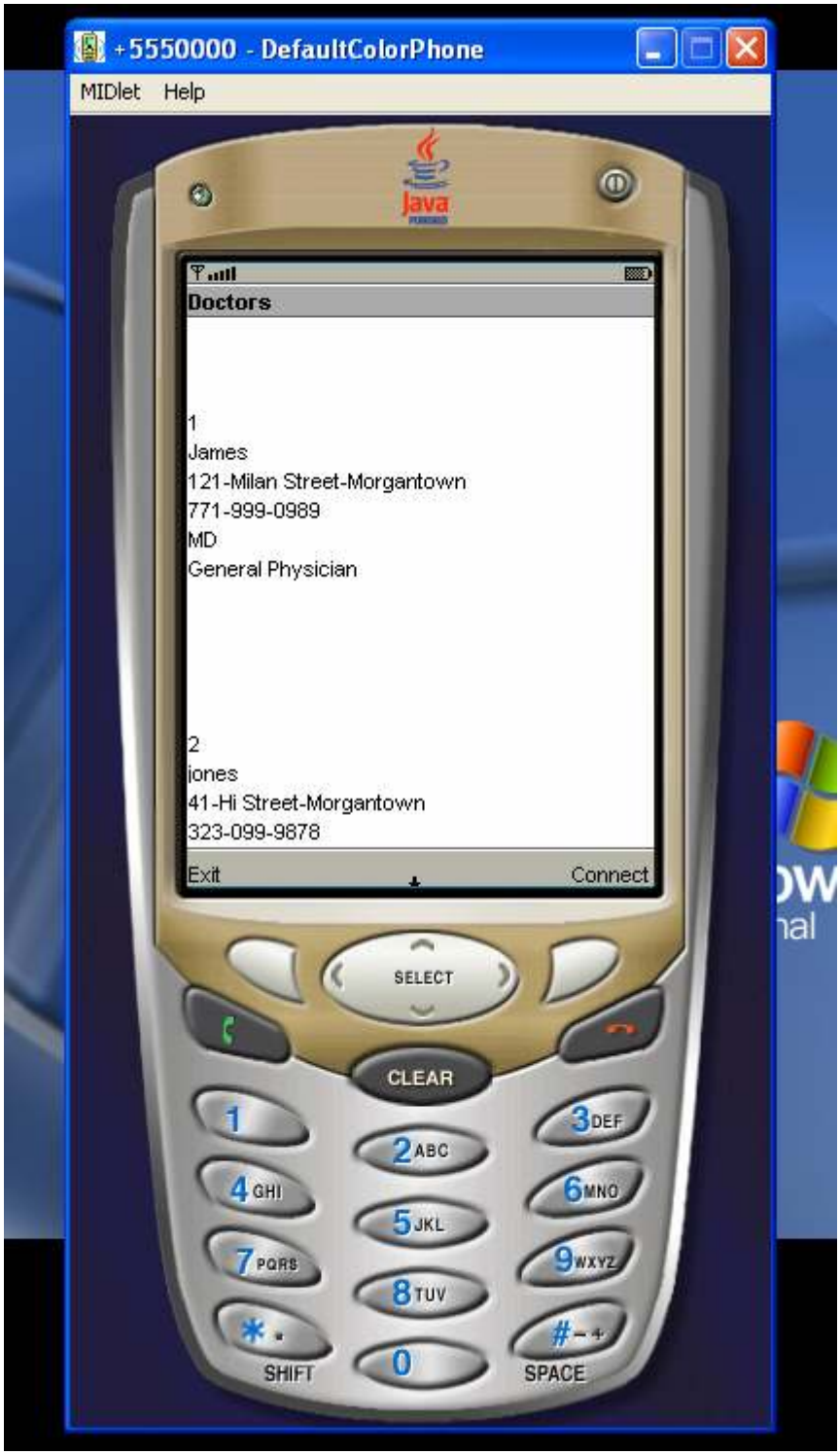


Inorder to include all the MIDlets select the MIDlets tab and include all the MIDlets that are created.



If any new MIDlets are to be added just keep adding them MIDlet-1, MIDlet-2 and so on. These MIDlets can be edited using the Edit button and then MIDlets can be edited accordingly.

This screenshot shows how the data is retrieved from the servlet and displayed in appropriate format. This is an example of Doctors data.


+5550000 - DefaultColorPhone
MIDlet Help
Doctors
1
James
121-Milan Street-Morgantown
771-999-0989
MD
General Physician
2
jones
41-Hi Street-Morgantown
323-099-9878
Exit
Connect
SELECT
CLEAR
1
2 ABC
3 DEF
4 GHI
5 JKL
6 MNO
7 PQRS
8 TUV
9 WXYZ
* .
0
- +
SHIFT
SPACE

Chapter 4

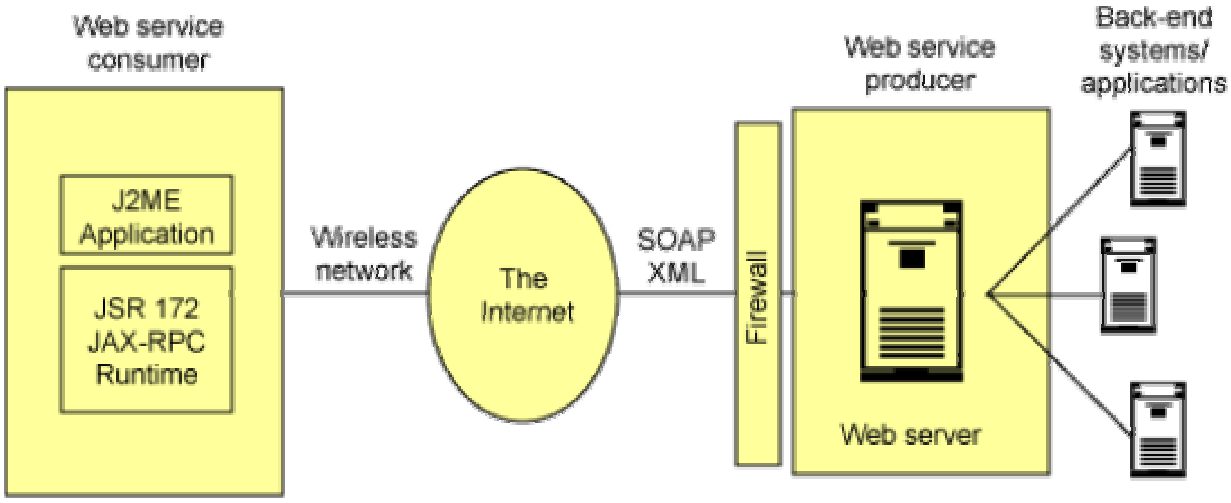
4.1 Security Issues: In order to address the security issues we can save the data in xml files so that the data is understood if and only if DTD is available. The concept of web services can be addressed to ensure security issues. Security is addressed to ease interoperability and allow for large scale software collaboration over the internet by offering services through a language and platform independent XML-based interface.

The promise of interoperability makes web services a good fit with J2ME. The JSR 172 specification, which was released on March 3, 2004, defines how a J2ME-enabled device can become a web services client. There is currently no specification for making a J2ME web service provider.

Web services in J2ME have been available up until now only in 3rd party libraries like kSOAP. However, this means that the library must be integrated with the application, bloating the deployment footprint and runtime size of the application.

JSR 172 WSA approach Web services from the client, service-consumer perspective: WSA provide the remote service invocation API (JAX-RPC) and runtime environment that allow J2ME applications to consume services on the Web, but not to function as service producers (endpoints). Apart from this difference, the rest of the architecture follows the standard architecture/organization of Web services, as illustrated next:

4.2 WSA high level Architecture:



This high-level architecture is structured as follows:

A client, Web services consumer: This is the J2ME application, such as MIDP or Personal Profile-based application, a JSR 172 stub and supporting classes, and the JSR 172 runtime.

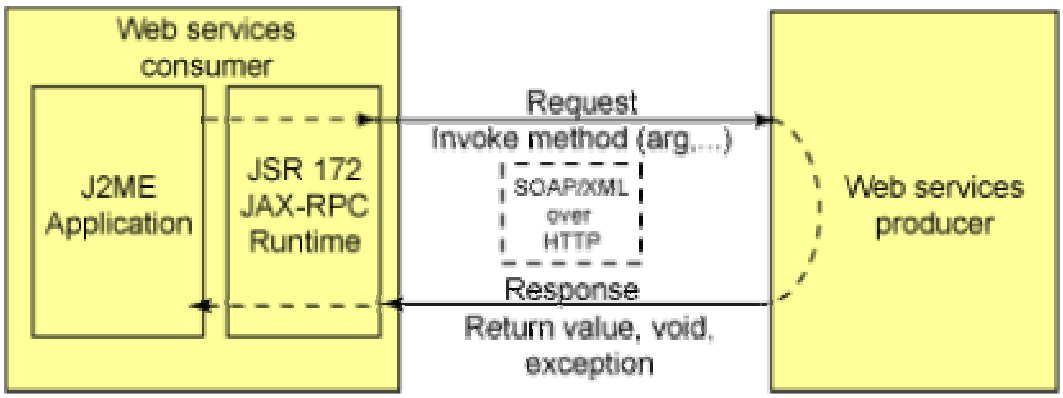
The network: This refers to the wireless and wired networks, part of the Internet, and the communication protocols. Note that JSR 172 does not mandate the use of XML encoding on the device itself, allowing implementations (as long as they're transparent to both consumer and producer) to use more efficient encoding approaches, such as the use of binary protocols between the device and the wireless gateway.

The server, Web services producer: This is a Web server, typically behind firewalls and/or proxy gateways. This server might have access to back-end resources.

4.2.1 Invocation model and data flow as standard Web services:

J2ME applications invoke remote services through the JSR 172 stub(s) and runtime, and typically over HTTP and SOAP. The stubs and runtime hide the complexities associated with remote service invocation including how input and return values are encoded and decoded, and the complexities related to managing network communication to the server. Method invocation follows the synchronous request-response model, as illustrated here:

JSR72invocationmodel



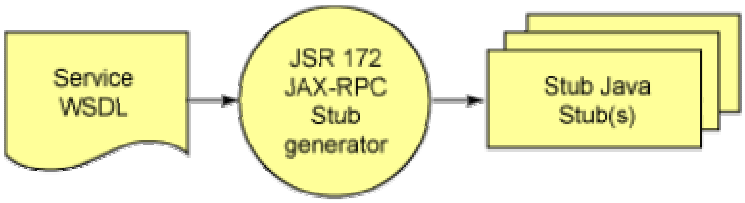
4.2.2 Consuming Web services

To consume a Web service, first create service invocation stubs. These stubs perform tasks, such as encoding and decoding of input and return values, and interfacing with the JSR 172 runtime to invoke the a remote service endpoint. Stubs interface with the runtime through the Service Provider Interface (SPI) of the runtime, which abstracts runtime implementations details, allowing portability of stubs

Stubs are typically generated using a tool that reads a WSDL XML document, which describes the Web service to be used. WSDL documents are, in turn, typically generated via a tool that reads interface definitions, such as Java Interfaces producing the WSDL document.

From our mobile development perspective, the WSDL document you want to consume will typically already exist, so all you have to do is generate the JSR 172 WSA stubs. To generate these stubs, you should use a tool such as the J2ME Wireless Toolkit 2.1 stub generator, shown here:

4.2.3 Generating a JSR 172 WSA stub



This generates the stub Java files, and related supporting classes. It also takes care of the WSDL-to-Java data type mapping, as described in the next section.

Once the JSR 172 JAX-RPC stub and supporting files have been generated, and application has been compiled and deployed onto a JSR 172-enabled device, consuming Web services is very straightforward and almost transparent.

With the J2ME Web Services API, web service support is built into the device rather than having to be integrated with the application. The advantages are obvious: faster deployment, smaller footprint, and an implementation that is optimized for the device and wireless network. The disadvantage is that it might not provide support

4.2.4 JSR supports the following features

- Provides client-side web service support
- Supported by CLDC and CDC configurations
- Target size of 35 kb
- Supports SOAP 1.1
- Supports SOAP messages in literal format
- Does not support SOAP messages in SOAP 1.1 encoding
- Supports WS-I Basic Profile
- Supports WSDL 1.1
- Does not support UDDI
- Does not support server-side web service functionality
- Security is provided by the platform, e.g. the MIDP 2.0 security framework

4.2.5 XML Encoding

It is interesting to see in the specification that XML encoding on the device is not required. What is required is that the device be able to consume XML web services from the internet. However, the XML messages can be transformed into another format before reaching the device. Conversely, the device can invoke web services using non-XML encodings as long as they are transformed to XML before reaching the web server. In other words, devices running on proprietary networks can use a more efficient format for the web service messages as long as it is transparent to the web service consumer and provider.

4.2.6 Portability and Interoperability

Standard web services promote interoperability through the use of WSDL. Web Service clients are created by tools that read the WSDL file that describes a web service. These tools then generate stubs that can be used by the client to access the web service as if it was a local object. The generated stubs are specific to the tool that generates them. For a J2SE web service, this isn't much of a problem. It just means that you must deploy a portion of the web service toolkit used to build the stubs with the web service client. Portability is provided, as usual, by the Java VM.

The J2SE deployment model does not work for J2ME, however. Web service support is provided as part of the J2ME platform on the device. If the support is not consistent

across devices, there would be no portability. It would defeat the purpose of J2ME Web Service if the deployed application was bloated because it's deployment had to include a portion of the web services toolkit.

To avoid this problem, JSR 172 goes a step further than the standard web services specifications. It also requires that each device also provide a JAX-RPC Service Provider Interface (SPI) as part of the runtime environment. Generated stubs must be written to the SPI to ensure portability. In general, J2ME developers do not have to worry about the SPI. It is used by the stub generator tool. Once the stub has been generated, the J2ME developer writes to the stub.

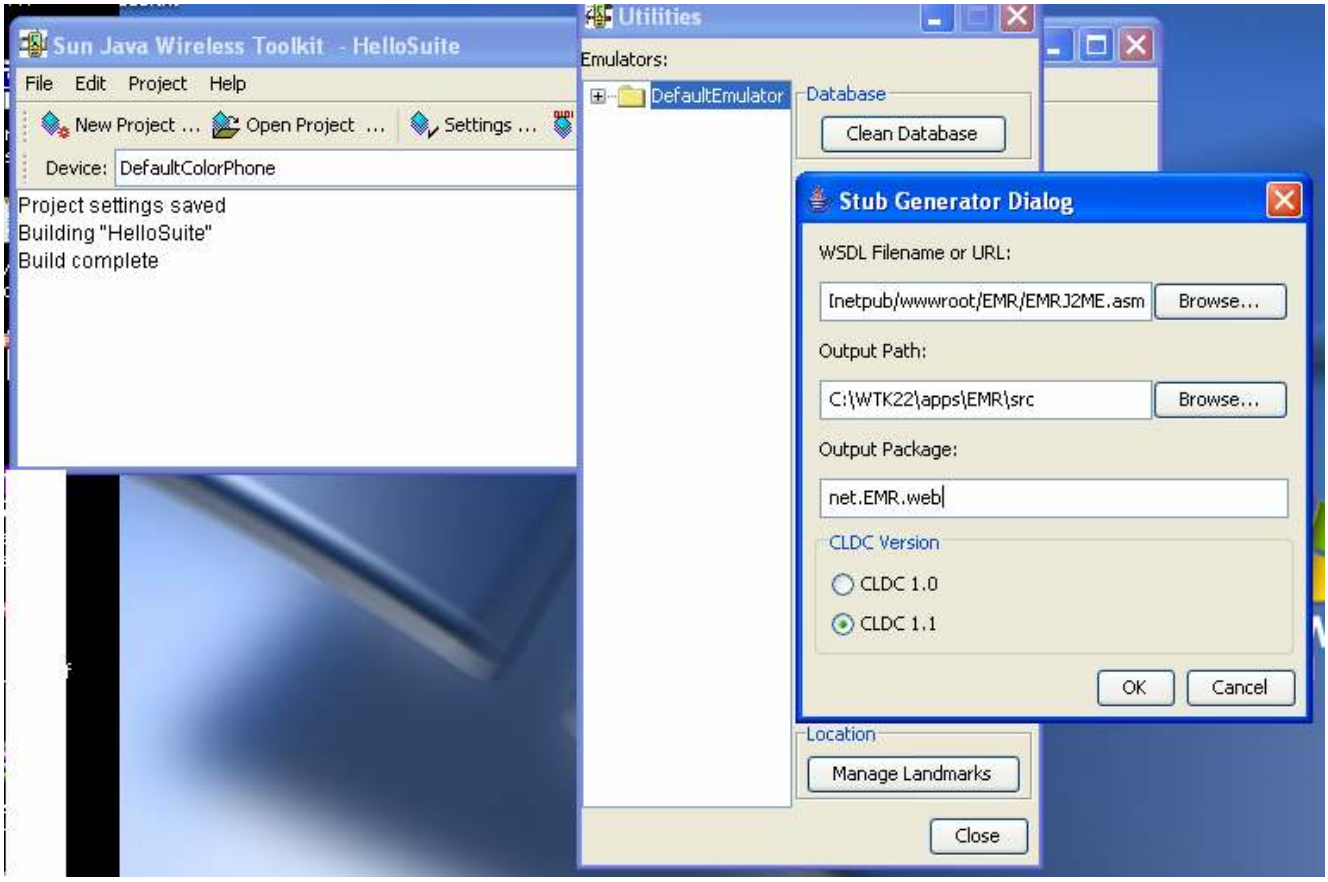
4.3 Implementation of Web services in Mobile services:

First create the web services in .net platform. There are two reasons to create web service in .net rather than using axis because it shows the power of interoperability with web services. Second, it is easier to write a web service for J2ME using .NET than using Axis, one of the major Java-based web service toolkits. The default message format for .NET web services is document/literal. The default message format (for now) for Axis is RPC/encoded. J2ME Web Services only supports the document/literal message format

For example it would be easy to create an application that will allow searching Google from a J2ME application. Although it is quite easy to use the Google web service from J2SE, it is impossible to use it from J2ME because Google has chosen to use rpc/encoded formatting for their SOAP messages.

4.4 Web service J2ME stub

We simply specify the location of the WSDL (as either a file location or URL), the output path, and the output package. Because J2ME Web Services clients are supported on both CLDC 1.0 and 1.1, we must specify which platform to target. Targeting CLDC 1.0 will provide the best portability but will be less efficient if floats are being returned. Under CLDC 1.0, floats are not supported so they will be returned as strings. CLDC 1.1 supports floating point types and operations. For our web service, it doesn't matter since we are only passing ints and strings.



Web services Stub Generator

The toolkit generates 7 .java files and their associated .class files although only the Stub class itself is needed by the J2ME application developer. The rest are support classes used by the stub implementation.

File	Description
EMRJ2MESoap.java	Interface implemented by the stub.
EMRJ2MESoap_Stub.java	Generated stub.
GetDoctors.java	GetDoctors() method parameters.
GetPatients.java	GetPatients() return value.
GetMedcines.java	GetMedicines() method parameters.
GetTranscriptions.java	GetTranscriptions () return value.
ArrayOfString.java	Wrapper for a String[]

Invoking the web service is as easy as calling a method on the stub class. The basic steps are:

- Instantiate the stub.
- Configure the service, e.g. by providing the URL for the web server and indicating whether the service invocation should be considered part of a session.
- Call the desired method on the stub.
- Wait for the return value.

J2ME Web Services do not support asynchronous operations. Therefore, it is usually better to invoke the web service from a separate thread. In fact, the J2ME Wireless Toolkit emulator will give a warning message if the web service is invoked from an event callback thread. Below is an example of how to call our web service.

4.5 Calling a Webservice:

When the MIDlet is started, it loads the EMR titles from the web service. While it is loading, it displays a message to indicate this. Once the titles are loaded, they are displayed as buttons. Note that we are using the new features of MIDP 2.0 to display clickable buttons on the screen and to give a more sophisticated layout than was possible in MIDP 1.0. The code below shows how to do this.

```
StringItem item = new StringItem(null, titles[i], Item.BUTTON);
form.append(item);
item.setLayout(Item.LAYOUT_2 | Item.LAYOUT_NEWLINE_AFTER);
item.addCommand(issuesCommand);
item.setItemCommandListener(this);
```

In the code snippet above, we create a StringItem and tell it to display in a BUTTON style. We also set the layout for the button, telling it to use the new MIDP 2 layout style and that we want a new line after it. Finally, we use another new feature of MIDP 2 that allows us to associate a command and a command listener with an individual item. In a J2SE application, make the ItemCommandListener an anonymous inner class, one for each button. However, with J2ME, every class on creation takes up precious resources.

Instead, we have the MIDlet itself listen to all buttons and then sort out which button was invoked in the callback method. It's not as pretty but is much more efficient.

When we call the web service, we always do it from a new Thread. This prevents the event thread from being blocked by the synchronous call to the web service. Once the thread has retrieved the result, we then update the display. Fortunately, the lcdui GUI toolkit for MIDP is thread safe, unlike Swing. So it is perfectly safe for us to update the display from our thread. It is noticed that the MIDlet itself implements the Runnable interface and is used by the new Thread object to invoke the web service. Again, this is done for efficiency whereas, in J2SE, it is better to create a couple of Runnable anonymous inner classes.

CONCLUSION

This problem report presented includes the complete document on Electronic Medical Records definition, purpose, scope and advantages. The primary objective of this problem report is to find a way to convert the data to a Wireless Markup Language (WML) format that is accessible from a Wireless Application Protocol (WAP) phone..

In this problem report we have development environment that supports both MIDP and servlets, we need to hook the two worlds together to create an end to end Java application. MIDlets can connect at large via HTTP, and the servlet is available via HTTP, so MIDlet can connect to the servlet. Thus we can connect to a servlet and get the data from database.

Also invoking a web service plays an important role in retrieving the data from the web. The promise of interoperability makes web services a good fit with J2ME. The JSR 172 specification, defines how a J2ME-enabled device can become a web services client.

Analyzing the medical data in the database and retrieving the data in to the mobile phones and also invoking the data from Web Services is the final result of this problem report.

FUTURE WORK

- MIDLET should support direct connection with JDBC rather than hooking up MIDLET with a Servlet and servlet in turn connected to JDBC.
- MIDLET client should be more user friendly.
- GUI can be implemented to have more functions and capabilities.
- Need better parsing technologies that support J2ME client to invoke web services efficiently
- Support of web services on all Mobile Devices
- J2ME Web services supports only literal format which means that it can support only .Net Web services, it should be capable of supporting java web services easily and efficiently.
- The mobile web services have a target size of 35kb, which can be expanded to larger size.
- Should support UDDI registry
- This application has been deployed in Wireless toolkit emulator; it should be expanded to real time.

Bibliography

[1] J2ME The Complete Reference.
James Edward Keogh,

[2] Understanding Web services: XML, WSDL, SOAP and UDDI
Eric Newcomer.

[3] J2EE The Complete Reference,
James Keogh

[4] Pei Xen Wu, Wireless Access to Clinical Information, Problem Report
West Virginia University, Morgantown, 2001.

[5] JDBC Tutorials specification and Documentation
http://java.sun.com/products/jdbc/download.html#cdcfp

[6] Web services with Java 2 Micro Edition by Mark Balbes, Principal Software Engineer, http://www.ociweb.com/jnb/jnbApr2005.html

[7] Web services API for Java 2 Micro Edition, Remote Service Invocation API
C. Enrique Ortiz, Mobility Technologist and Writer, J2meDeveloper.com
http://www-128.ibm.com/developerworks/wireless/library/wi-jsr/

[8] Using a J2ME client to access a Web Service
http://developers.sun.com/sw/building/codesamples/j2me-client/index.html,

[9] Wireless Development Tutorial part ii
http://developers.sun.com/techtopics/mobility/midp/articles/tutorial2/

[10] Electronic Medical Records, Overview, specifications and Descriptions
http://omnimd.com/html/EMRS.html

[11] Story Of a Servlet, An Instant Tutorial, Mark Andrews
http://java.sun.com/products/servlet/articles/tutorial/

[12] Learning XML, SOAP, WSDL, UDDI, WAP, WML
http://www.w3schools.com/

[13] EMR on Mobile Devices
http://www.medicaltabletpc.com/content/view/38/33/

Index

A
Axis 34

E
EMR 1,2,3

J
Java 12,13,14,22
J2ME 16,33,35,36,39
J2SE 16,19,23,33,36,37
JSR 30,31,32,33,38

K
Ktoolbar 25

M
MIDlet 16, 17, 18

P
Portability 32, 33, 34

S
Servlet 12
Security Layer 7, 8
Session Layer 7, 8
SOAP 20, 30
Stub 31, 32, 33

T
Transaction Layer 7, 8
Transport Layer 7, 8

U
UDDI 33

X
XML 19, 20
XML Parser 20

W
WAP 5,6,7
Web Service 31,32,34,35
WML 10,11,12
WSDL 32, 33, 34