Distributed Gaming using J2ME and XML

CS 297 Project Report

Rekha Vaddepalli

(rekha_vad@yahoo.com)

Advisor: Dr. Chris Pollett

Table of tables

1. Introduction

The use of small computing devices has been increasing in the recent years as the people have become more time conscious and want to utilize the time that they are mobile. So, various technologies that are suitable for small devices have emerged. Java 2 Micro Edition (J2ME) technology is one of them.

In my CS297 course, I have completed various deliverables that will help me understand J2ME concepts and would further help me with my CS298 in which I am planning to implement a digital cash system for the mobile devices. Using the digital cash system, the user of the mobile device will be able to transfer the digital cash to the user of the other device anonymously.

The second section covers the technologies that are used in this project.

The third, fourth, fifth and sixth sections cover the descriptions of the first, second, third, and fourth deliverables respectively. The seventh section concludes the report. The eighth section gives the references for the report and the deliverables.

2. Technologies involved

2.1 J2ME

J2ME is the reduced version of Java API that is designed to operate within the limited resources of the mobile devices. It consists of the configurations and the profiles. The Java Community Process Program defined **configuration** as the Java run-time environment and core classes that operate on each device. The Java Virtual Machine for a particular small computing device is defined by the configuration. There are two configurations – CLDC (Connected Limited Device Configuration) and CDC (Connected Device Configuration). CLDC devices are low-end mobile devices that include pagers, cell phones, digital assistants. They usually have a 16-bit or 32-bit architecture, have 160KB to 512KB of available memory, and are battery powered and use KJava Virtual Machine (KVM). CDC devices are high-end devices that

include digital set-top boxes, home appliances etc. They usually have a 32-bit architecture, 2MB of available memory, and use complete functional JVM.

A **profile** consists of Java classes that enable implementation of features for either a particular small computing device or for a class of small computing devices. The different profiles available right now for CDC configuration are Foundation Profile, Game Profile, Personal Profile, Personal Basis Profile, and RMI Profile. The CLDC profiles available are Mobile Information Device Profile, PDA profile, and Information Device Profile. Each profile has a different functionality.

2.1.1 MIDlets and MIDlet Suites

Java applications that run on MIDP devices are known as MIDlets. A MIDlet consists of atleast one Java class that must be derived from the MIDP-defined abstract class javax.microedition.midlet.MIDlet. The MIDlets may be grouped into a MIDlet suite. The MIDlet classes, other required class files or images must be built into a single JAR file. The JAR's manifest file must be written. This file contains the packaging information that tells the device what is in the JAR. Similar information is also provided in another file called Java application descriptor (JAD). If any of the attributes appears in both of the manifest and JAD file, the value in the JAD file takes precedence.

2.1.2 MIDlet execution environment

A MIDlet must have a public default constructor that may be used to perform any initializations.

This is what a skeleton MIDlet looks like:

```
public class MyMIDlet extends MIDlet
{
        MyMIDlet()
        {
        }

        protected void startApp() throws MIDletStateChangeException
        {
```

```
        }

        protected void pauseApp()
        {
        }

        protected void destroyApp(boolean unconditonal)
                                        throws MIDletStateChangeException
        {
        }
}
```

The technologies that were used in addition to J2ME are XML, Java Servlets, and Oracle Database.

2.2 EXtensible Markup Language (XML)

The World Wide Web Consortium created XML as a way to disseminate complex, structured data and documents over the Web. All XML documents must be well formed. But they need not be valid documents. The two methods for defining a valid document are DTD (Document Type Definition) and schemas. A DTD or schema specifies how a document's elements can work together to create a specific structure. DTDs may be useful to create and manage large sets of documents, to define clearly what markup may be used in certain documents and to provide a common frame of reference for documents that many users can share. XML Schema is one of many schema specifications. Unlike a DTD, schemas are simply XML documents that use XML's standard markup syntax to define the structure for other documents.

2.3 Servlets

A servlet is a server-side software component, written in Java, that dynamically extends the functionality of a server. Servlets provide a framework for implementing the client-server applications. While working with mobile devices, it becomes all the more important to shift the load from the small device to the server and this can be accomplished by the usage of servlets.

2.4 Oracle Database

This project uses Oracle Database by the server for storing the information needed for processing the client's requests. In this project, most of the work is delegated to the server by the client devices. So, the server is required to store lot of information in the database. Specifically, Oracle XML Database, which is the database that combines the benefits of both XML and Oracle technologies, is used.

3. Deliverable One: Using J2ME's Graphical User Interface and Record Management System

3.1 Description of the Deliverable One

The Deliverable One introduced me to J2ME's GUI and RMS (Record Management System) concepts. In this deliverable, user will be presented with a menu with the options--"Enter Info" and "List". Using the Enter Info option, the user can enter the business card information like First Name, Last Name, Phone Number, and Email ID in the form that gets displayed. The user can then submit the information using the submit button on the form. Using the "List" option, he can get the list of all the business cards that have been stored.

3.2 J2ME's Graphical User Interface

There are three kinds of user interfaces that an application make use of. They are command, form and canvas based interfaces. A command-based user interface consists of instances of the Command class. A form-based user interface consists of an instance of the Form class that contains instances derived from the Item class such as text boxes, radio buttons, check boxes, lists etc A canvas-based user interface consists of instances of the Canvas class using which images can be drawn by the user. The device's screen is referred to as the display and the reference to an instance of the MIDlet's Display class can be obtained to interact with it.

3.3 Record Management System

Record Management System (RMS) is like a combination of the file system and the database management system that enables the user to store the information or data in columns and rows like in a table of a database. SQL cannot be used to interact with the data as RMS is not a relational database. So, for the functions like sorting, searching and manipulating information stored, RMS API and Enumeration API have to be used.

3.3.1 Record Store:

A record store is a collection of records organized as rows (records) and columns(fields). Multiple record stores can be created by a MIDlet in a MIDlet suite as long as the name of the record store is unique. Duplicate names for the record stores can be used in other MIDlet suites.

3.4 Details of the deliverable

3.4.1 Using Record Store

The following code shows the way to add a record to the record store

```
//opening the Record Store
recordstore = RecordStore.openRecordStore("myRecordStore",true);
byte[] outputRecord;
firstname = “Rekha”;
lastname = “Vaddepalli”;

ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
DataOutputStream outputDataStream = new DataOutputStream(outputStream);

//writing the values to the DataOutputStream
outputDataStream.writeUTF(firstname);
outputDataStream.writeUTF(lastname);
outputDataStream.flush();

//Converting the DataOutputStream to byte array
outputRecord = outputStream.toByteArray();

//adding the record to the Record Store
 recordstore.addRecord(outputRecord,0,outputRecord.length);

//closing the record store
recordstore.closeRecordStore();
```

Table 1: Adding a record to the Record Store

3.4.2 Record Enumerations:

RecordStore API includes a method called enumerateRecords() to search efficiently through all the records in the record store to get the identifier for any record.

The following call gets a static snapshot of all the entries in the record store.

RecordEnumeration enum = recordStore.enumerateRecords(null,null,false)

The following code shows how to get the records using RecordEnumeration

```
//Traverse forwards
while(enum.hasNextElement())
{
        byte[] record = enum.nextRecord();
        //Do something with this record
}

//Traverse backwards
while(enum.hasPreviousElement())
{
        byte[] record= enum.previousRecord ();
        //Do something with this record
}
```

Table 2: Using RecordEnumeration

RecordEnumeration has a reset() method which allows to restart the iteration from the beginning.

4. Deliverable Two: Transmitting s record wirelessly from one device to another

4.1 Description of the Deliverable Two

The purpose of the Deliverable Two is to establish a connection between two wireless devices and transfer information wirelessly. This deliverable consists of three classes- TransferRecord.java, RegisterServlet.java, and ListingServlet.java

TransferRecord.java is the application to be loaded onto the wireless device. The welcome screen of the application has a menu with the options- Register, Add Record, Transfer Record, Receive Record, Delete Record, List Records, and Quit. Using the "Register" option, the user can register his device's IP address at the central server. Using "Add Record" option, the user is prompted with the input screen where he can enter the Name and Phone Number and add the record into the local record store. This is done using the J2ME's RMS functionality.

The "Receive Record" option facilitates the user who wishes to receive a record from the other user by starting the server and receiving the record.

Using "Transfer Record" option, the user can transfer a record to the other receiving device. When this option is selected, the Record Search Screen is displayed using which the user can select the record that he wishes to transfer by specifying the Name value of the record. Then he has to select List Destinations option for the display of all the destinations registered at the central server. From the list of the destinations, he can select a destination for the record transfer. Then it connects with the receiving device and transfers the record. When the transfer is completed, the Success Screen is displayed at the receiver's end.

Using "Delete Record" option, the user can delete a record. Using "List Records" option, the user can list all the records in the local record store.

The RegisterServlet.java connects to the Oracle database and puts the client's IP address into it. The ListingServlet.java connects to the Oracle database and queries it and returns a list of the IP addresses that are registered with the server. The connection between the two wireless devices in this deliverable is achieved using DatagramConnection.

4.2 Installing VM onto Pocket PC

I have downloaded the IBM's WebSphere Studio Device Developer (WSDD) with Websphere Micro Environment with which IBM's J9 VM was installed.
To install the VM onto the Pocket PC, initially establish an ActiveSync connection between the development machine and the Pocket PC and copy the J9 runtime files onto Pocket PC. Then, create a device configuration, build, and launch configurations for the Pocket PC as given in the WSDD tutorial.

4.3 Details of the deliverable

Usage of the J2ME's GUI and RMS is done in the same way as in the Deliverable One. The way in which the datagram connection is set up between the two devices is given below. The coding details for the connection between the device and server is shown in the section for the Deliverable Three.
The following code shows how to send a datagram

```
DatagramConnection sender =
(DatagramConnection)Connector.open("datagram://target:32767");

Datagram dgram = sender.newDatagram(64);

Dgram.writeUTF("hello");

Sender.send(dgram);
```

Table 3: Sending a datagram

The following code shows how to receive a datagram

```
DatagramConnection receiver
        =(DatagramConnection)Connector.open("datagram://:32767");

byte[] buffer new byte[256];

Datagram dgram = receiver.newDatagram(buffer,buffer.length);

        for(;;)
        {
        dgram.setLength(buffer.length);
        receiver.receive(dgram);
        int length = dgram.getLength();

                //printing out the contents of the datagram
                 for(int i=0; i<length;i++)
                 {
                  System.out.println(buffer[I]+ " ");
                 }

        }
```

Table 4: Receiving a datagram

5. Deliverable Three: Connecting Pocket PC to the HTTP server

5.1 Description of the Deliverable Three

Deliverable Three shows setting up of a connection between Pocket PC and HTTP Server. The user of the Pocket PC can invoke the application on the Pocket PC by tapping the application's name on the Start menu. Then a form that takes in the details of the baby like name, parents' names, doctor's name, birth date etc is displayed. When the user presses the submit button, the details are put in the XML format, the information gets stored in the local device and the Pocket PC connects to the Tomcat server. This deliverable is extended with the Deliverable Four using which the user can upload the baby details to the Oracle database.

The following code shows how to define the URL of the servlet

```
Private static String url =
            "http://anouki:8088/examples/servlet/XMLOracleservlet";
```

Table 5: Defining the URL of the servlet

HTTP Connection can be obtained and dealt with by the following code

```
HttpConnection conn = (HttpConnection)Connector.open( url );
Int rc = conn.getResponseCode();
if( rc == HttpConnection.HTTP_OK )
{
//Do something
}
else
{
//Display that different response code is received
}
```

Table 6: Obtaining HttpConnection

The connection is made on the new thread to make the GUI responsive.

The following is the code to set the connection parameters

```
conn.setRequestMethod(HttpConnection.POST);
conn.setRequestProperty("User-Agent",
                        "Profile/MIDP-1.0 Configuration/CLDC-1.0") ;
conn.setRequestProperty("Content-Language", "en-US");
conn.setRequestProperty("Accept", "application/octet-stream");
conn.setRequestProperty("Connection", "close");
conn.setRequestProperty("Content-Length",
                          Integer.toString( data1.length));
```

Table 7: Setting Connection parameters

6. Deliverable Four: Connecting HTTP Server to the Oracle Database

6.1 Description of the Deliverable Four

The Deliverable Four is an extension of the Deliverable Three. In this deliverable, the connection between the HTTP Server and the Oracle Database is set up. The server gets the data from the client and puts in the Oracle XML database. So this will include setting up of the database and getting the connection from the server.

6.2 Introduction to Oracle XML Database

Oracle XML Database combines the advantages of both the Oracle technology and the XML technology. It contains a number of tools for processing XML into and out of the database: New datatypes like XMLType (for XML), and URI-Ref (for logical pointers) were added to the kernel for direct XML storage.

6.3 Creating Oracle XML Database

To create Oracle XML Database, initially connect to the database as sysdba, Then, create directory XMLDIR as 'C:\CS297\ActualDel2\HTTPDB', grant read on directory XMLDIR to public with grant option, and connect to the database as scott / tiger.

To store the XML document in the database, it must first be converted to XMLType instance. The function to create this is given below

```
Create or replace function getDocument(filename varchar2) return clob
  Authid current_user is
       xbfile bfile;
       xclob clob;
       begin
           xbfile  := bfilename('XMLDIR2',filename);
          dbms_lob.open(xbfile);

          dbms_lob.createtemporary(xclob,TRUE,dbms_lob.session);
          dbms_lob.loadfromfile(xclob,xbfile, dbms_lob.getlength(xbfile));
          dbms_lob.close(xbfile);
          return xclob;
        end;
```

Table 8: Function getDocument()

An XML schema for the data should be written and put in the directory created. The

XML schema used in this deliverable is given below

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="baby">
  <xs:complexType>
    <xs:sequence>
        <xs:element name="BabyName" type="xs:string"/>
        <xs:element name="MomName" type="xs:string"/>
        <xs:element name="DadName" type="xs:string"/>
        <xs:element name="DocName" type="xs:string"/>
        <xs:element name="DueDate" type="xs:date"/>
        <xs:element name="HospitalName" type="xs:string"/>
        <xs:element name="BirthDate" type="xs:date"/>
        <xs:element name="BirthTime" type="xs:time"/>
        <xs:element name="Weight" type="xs:integer"/>
        <xs:element name="Height" type="xs:integer"/>
        <xs:element name="ColorEyes" type="xs:string"/>
        <xs:element name="ColorHair" type="xs:string"/>
        <xs:element name="BirthStone" type="xs:string"/>
        <xs:element name="Flower" type="xs:string"/>
        <xs:element name="NickNames" type="xs:string"/>
        <xs:element name="PediatricianName" type="xs:string"/>
        <xs:element name="FirstBirthDayVenue" type="xs:string"/>
        <xs:element name="FirstBirthdayCake" type="xs:string"/>
        <xs:element name="FirstBirthdayOutfit" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:schema>
```

Table 9: XML Schema

The code to register the schema with the database is given below

```
Begin
     dbms_xmlschema.registerSchema( 'baby.xsd',
                                    getDocument('baby.xsd'),
                          TRUE, TRUE, FALSE, FALSE);
 end;
```

Table 10: Register Schema

Then, create the user table with the XML schema written. The code to do this is given below.

```
CREATE TABLE BABY_BOOK_ENTRIES of XMLType
  XMLSCHEMA "baby.xsd"
  ELEMENT "baby";
```

Table 11: Create table

6.4 Using Servlets and JDBC

This deliverable consists of XMLOracleServlet.java which connects the server to the Oracle XML database through JDBC and puts the data sent by the client into it. This servlet then returns the response code to the client so that the client can display the result accordingly.

The XMLOracleServlet.java has a doPost() method that carries out all the work of the servlet.

The following code shows how to read the HttpRequest

```
ServletInputStream in = request.getInputStream();
DataInputStream din = new DataInputStream( in );

String text = din.readUTF();
Din.close();
```

Table 12: Reading Http request

The following code shows how to register the JDBC driver

```
DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
```

Table 13: Registering the JDBC driver

The following code shows connecting to the Oracle database, forming the statement and executing it

```
String username="scott", password="tiger",
StringBuffer sqlXml;

DbURL="jdbc:oracle:thin:@anouki:1521:ORA9IDB";

Connection
connection=DriverManager.getConnection(dbURL,username,password);

Statement statement=connection.createStatement();

SqlXml.append("insert into BABY_BOOK_ENTRIES values (xmltype('" );
sqlXml.append("<baby xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-
instance\" xsi:noNamespaceSchemaLocation= \"baby.xsd\"> ");
SqlXml.append(text);
SqlXml.append("</baby>");
 SqlXml.append("'))");

Statement.executeUpdate(sqlXml.toString());
 Statement.close();
Connection.close();
```

Table 14: Connecting to database, statement creation and execution

The following code shows how to form the response and send it back to the Pocket PC.

```
Response.setStatus( response.SC_OK );
ServletOutputStream out = response.getOutputStream();
Out.close();
```

Table 15: Forming response and sending

7. Conclusion

Through all the deliverables done so far, I have come to know that though J2ME is a subset of J2SE, one must be careful in designing a J2ME application. The reason is that a small computing device has a radically different hardware configuration than traditional computing devices. The differences lie in the inconsistency of a network connection and diminishing longevity of the power.

Using Oracle XML, one can have the advantages of both relational database and XML. Also, it has a number of tools to use when processing the XML data into and out of the database. The Deliverable Two was to transfer the records wirelessly between two devices, the first option to do this was to do using infrared beaming. But, this proved to be difficult as MIDP has limited functionality and does not support infrared beaming directly. So, we came up with this scheme where a server sits in between the clients and provides each client with the list of IP addresses from which he can select a peer and establish a socket connection.

Digital Cash systems are gaining importance in the recent world. So, implementing cryptographic algorithm for ensuring the user security and anonymity is an interesting and a challenging task.

8. References

[SM0203] Sun Microsystems Inc.(February 1, 2003). Future Java Technology for the Wireless Services Industry. Retrieved May 21, 2003.

[SR02] Srikanth Raju (2002). XML and Java language programming for wireless devices: a tutorial. Retrieved May 21, 2003.

[SM0503] Sun Microsystems Inc. Programming strategies for small devices. Retrieved May 21, 2003.

[SSKKVNT] Shanmugasundaram, Shekita, Kiernan, Krishnamurthy, Viglas, Naughton, and Tatarinov. A General Technique for Querying XML Documents using a Relational Database System.

[SM02] Jonathan Knudsen (March 7, 2002). Parsing XML in J2ME, XML in a MIDP Environment. Retrieved May 21, 2003.

[TVBSSZ02] Tatarinov, I., Viglas, D.S., Beyer, K., Shanmugasundaram, J., Shekita, E., Zhang, C. (June, 2002). Storing and Querying Ordered XML using a Relational Database System. Proceedings of the 2002 ACM SIGMOD international conference on Management of data, 204-215.

[K02] Kim Topley (2002). J2ME in a Nutshell: A Desktop Quick Reference. (1st ed.). O'reilly Publications.

[J03] James Keogh (2003). J2ME: The Complete Reference.(1st Ed). Tata McGraw-Hill.

[P02] Paul Tremblett (2002). Instant Wireless Java with J2ME.(1st Ed). McGraw-Hill.

[VM02] Victor Votsch, and Mark Walter. (March 02). http://otn.oracle.com/tech/xml/xmldb/pdf/xmldb_buswp.pdf. Retrieved Dec 01, 2003.