J2MEMicroDB: An Open Source Distributed Database Engine for Mobile Applications.

Marc Alier
Universitat Politècnica de Catalunya
C. Jordi Girona Salgado, 1-3
Barcelona – E08034
+34934137885
malier@lsi.upc.edu

Pablo Casado
Universitat Politècnica de Catalunya
C. Jordi Girona Salgado, 1-3
Barcelona – E08034
+34934137885
pcasado@lsi.upc.edu

Maria José Casany
Universitat Politècnica de Catalunya
C. Jordi Girona Salgado, 1-3
Barcelona – E08034
+34934137885
mjcasany@lsi.upc.edu

ABSTRACT
Mobile distributed applications must be able to work offline when no network connection is available, or simply to spare bandwidth or money. To do so the mobile client must be able to store and handle structured data. Java 2 Micro Edition (J2ME) does not support neither object serialization nor relational table management. When most legacy systems still rely on database engines, it also seems that J2ME only considers the *Webservices API* as a tool to access data on the server side. No JDBC API is provided to send and retrieve data from the mobile device to the DBMS in the server.

This paper presents an Open Source library for J2ME that implements a lightweight database engine that allows CDLC devices (such as the currently available mobile phones) to handle object serialization easily, SQL relational database management and JDBC, using SQL to access any DBMS in the mainframe.

Categories and Subject Descriptors
H.3.4 **[Information Systems]**: Systems and Software. *Distributed systems.*

General Terms
Design.

Keywords
Mobile devices, data management, storage, distributed client applications, ubiquitous information access, distributed software development.

1. J2ME FOR MOBILE CLIENT APPLICATIONS
J2ME is one of the available technologies to develop advanced mobile applications. It provides portability and cross-platform compatibility. J2ME adapts to devices with limited hardware capabilities, variable quality wireless communications to unreliable networks and mobility. Besides, the number of mobile phones that include a Java Virtual Machine is growing day by day.

For the previous reasons we selected J2ME for our developments. We focused on the development of advanced mobile applications with ubiquitous information access. These applications must be able to work online when there is network connectivity as well as offline when there is no network connectivity or no need to access online information. To do so, some caching mechanism on the mobile side must be provided. The limitations of network bandwidth have also been considered. Although network technology advances very quickly and network bandwidth increases, wireless networks bandwidth are still limited and expensive. For this reasons, the information delivered to the mobile device must be minimal and must fit in the users needs [1]. Finally, mobile users may experience sudden disconnections in which cases they will not be able to access online information or perform any other action that requires a network connection.

In order to accomplish our goal of developing advanced mobile applications, the possibilities and limitations of the Java technology for mobile devices were studied. We found the following important limitations related to data access:

Lack of database support. J2ME does not provide developers with JDBC support to access remote databases. J2ME only provides a limited JDBC Optional Package for the CDC configuration in the foundation profile [2]. This JDBC Optional Package is based on the JDBC 3.0 Specification. But it does not support some common features of JDBC such as connection pools, setting parameters to the CallableStatement interfaces or SQL 99 types (Struct, Array, Ref, SQLData and SQLOutput interfaces). Besides, this optional package is not available for the CLDC configuration, so developers must create their own mechanisms to access remote databases.

Lack of structured persistent data storage on the mobile device. The first requirement for offline operation is to store and manage application data on the mobile device. The standard J2ME profiles provide limited data storage and management capabilities. On one hand MIDP devices have only record stores from the Record Management System (RMS) to store data [3]. The developers must use low level programming interfaces to store data in these records stores; there are no object serialization capabilities. On the other hand the Foundation profile has plain random files.

Ja4Mo Workshop/PPPJ 2007, Sep. 5--7, 2007, Lisboa, Portugal.

This paper presents the J2MEMicroDB library, a lightweight Open Source library and some of its applications. J2MEMicroDB provides:

- Access from any J2ME mobile client application to any remote DBMS server and the possibility of view materialization on the mobile client.
- A high level API to store persistent data on the mobile devices. It provides object serialization as well as a small relational database and the associated methods to manage this database.
- A Backup/ Restore mechanism to backup the small database stored on the mobile device on a memory card.

This database is lightweight, a very important feature because it has to run on devices with limited storage capabilities, and is distributed under a Free and Open Source license like GPL (General Public License). We considered this last issue to be an important one, because the source availability allows ad hoc optimization and minimizes the risk of the project.

2. INDUSTRY ALTERNATIVES

Nowadays there aren't available DBMS for any mobile device. There are microDBMSs specifically created for some mobile devices such as Palm OS, Symbian or Windows Mobile. These microDBMSs are limited proprietary solutions for a specific hardware and it is not possible to access these systems using J2ME.

Due to the nonexistence of generic mechanism for accessing databases in J2ME, software companies have developed their own solutions. Now we would like to mention some of these projects that access databases from mobile devices.

IBM toolbox for J2ME is a package that allows developers to create mobile applications that access remote databases located on an OS/400 and i5/OS server [4]. There is also an Open Source version of this package called JTOpen [5], which provides developers with a small JDBC driver to connect a mobile application (once more) with a remote OS/400 server. Another company that has developed a small JDBC driver in order to communicate a mobile application with a remote database server is DataMirror and its product is PointBase Micro [6].

CodeBase for J2ME is a high-speed database engine for J2ME. It allows developers to create fully functional database applications. Oracle Lite is a small version the Oracle DBMS software that provides access to remote databases and to data stored on a local database [7].

Sybase Anywhere [8] is a small database for J2ME devices that provides access to remote databases and to data stored on a local database and offers MIDP 2.0 support. It is not an Open Source product. Finally the HSQL Database Engine [9] is an open source database based on Thomas Mueller's Hypersonic SQL project. It is written in java and it is included in many J2EE application servers. On mobile devices, HSQL runs on the foundation and personal profile.

3. J2MEMICRODB

3.1 System Architecture

J2MEMicroDB is based on a three-tier architecture as shown on figure 1. It includes a J2EE proxy server application that runs on a HTTP server and manages the access from the mobile client to remote databases. On one hand, this proxy server application uses the JDBC API to connect to remote databases and on the other hand uses the HTTP protocol to communicate with the mobile client. The current version of the J2EE proxy is a prototype. The next step consists of turning this application into a parameterized one that may include: 1) data encryption, 2) mobile user access control management and 3) web administration of the system.

J2MEMicroDB also includes a J2ME library that provides advanced persistent storage capabilities on the mobile client [10] application. In the following section the system architecture of this part of the project is described.

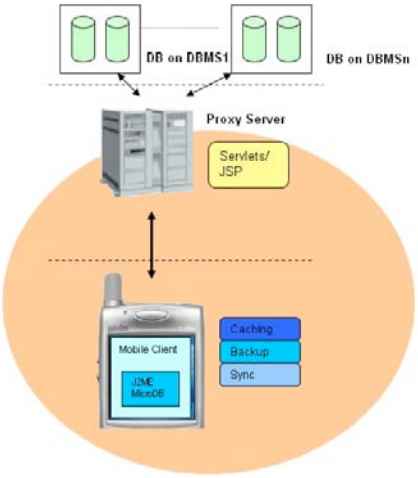

Figure 1. Three-tier architecture of the system.

3.2 The mobile client API

In the development of the mobile-side library the limitations of the current version of J2ME described in section 1 were considered. The first step was the development of an API above the RMS system provided by J2ME. This API provides four types of files that allow object serialization and is easy to use. The main goals of this library are: 1) extend the limited search functionalities of the Record Management System provided by J2ME and 2) provide object serialization.

This library provides the following four types of files:

- The PDAStoreSequence provides sequential access to the objects stored in the file.
- The PDAStoreRMS provides both sequential and direct access to the objects. The direct access to an object is provided given its logical position in the file.
- The PDALabeled Store provides direct access to objects by label.
- The PDALabeledStoreSequence provides direct and sequential access to objects by label.

The previous files are implemented in classes that extend the PDAStore abstract class. This class defines the methods that must be implemented in the four types of files.

The design of this API is based on an interface (PDAStoreable) that must be implemented each time a developer uses one of the files. This interface provides the methods to serialize/deserialize objects from the Record Store and is used, as well, as an abstraction layer to store any object in a file. The following figure shows an example of the PDAStoreable interface and a code fragment.

```
public class Contact implements PDAStoreable {
                    ….
   public void setData(byte[] b, int size) {
                    …
            public byte[] getData(){
                    …
               public int size(){
                    …
```

```
File= new PDALabeledStoreSequence("contacts");
          File.write(contact,"label")
               File.seek(name);
        contact=(Contact)File.read();
               File.close();
```

Figure 2: Example of the PDAStoreable interface.

Above this layer a relational database layer was built. This new layer allows developers to create a database on a mobile device and manage data stored on relational tables. This layer provides the following functions.

- Access to any remote DBMS using a J2EE proxy server. This part is described in section 3.3.
- A caching mechanism to store persistent data in relational tables in the mobile client.
- A backup / Restore mechanism of the database on an additional memory card. This part is described in section 3.4.

And, finally, the fourth layer gives a limited SQL interpreter to query the local database is provided. The following figure shows the architecture of the mobile client API.

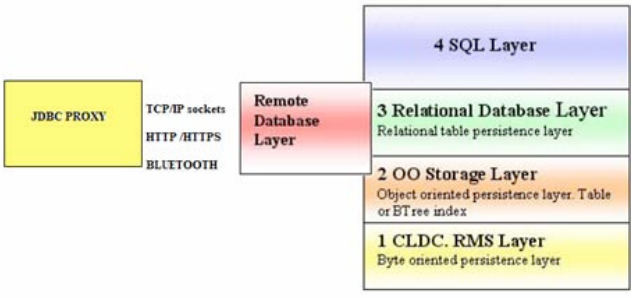

Figure 3: Mobile Client layer architecture.

3.3 Access Remote databases

One of the libraries included in the J2MEMicroDB project is a library to access any remote database from the mobile client. To do so we used the three-tier architecture described in section 3.1. In this section the API to access the remote database is described.

The following issues were considered in the development of this library:

- The connection between the mobile client and the server may be severed suddenly.
- The response time from the server may be high due to the limitations of wireless communications.
- The response from the server must include the minimum amount of data due to the limited storage capabilities of the mobile client.

To deal with the problem of wireless communication we decided to use an asynchronous communication system between the mobile client and the proxy server application. This asynchronous communication is implemented using java listeners. We defined two interfaces: the RemoteProxyConnection and RemoteProxyListener. The first interface is used to send asynchronous requests to the Proxy Server, and the second interface is used to receive asynchronous responses from the Proxy Server. The methods that must be implemented when using these interfaces are shown in the following figure:

```
Public interface RemoteProxyConnection{
    public void addListener(RemoteProxyListener listener);
    public void sendMessage (String msg);
}
Public interface RemoteProxyListener{
  public void performeResponse(String response,
                               boolean Exception);
}
```

Figure 4: Interfaces to provide asynchronous communications.

The use of asynchronous communications allows the client application to send requests to the proxy server without waiting for any response. Each request sent to the server is identified by a number in order to be able to retrieve the response when the server sends it back to the client. The requests sent by the client may be any SQL statement. When the request is a query the result may be stored on the mobile client as a materialized view.

In order to send requests to the Proxy Server the HTTP protocol is used, although the library could be easily adapted to use the sockets protocol. The following figure shows a code example of this library:

```
MobileHTTPConnection mc=new MobileHTTPConnection(ConnectionData);
mc.DBConnectionRequest();
while (!mc.DBOpened()){}
int ticket=mc.DBStatement(SQLStatement);
MobileHTTPStatementResult result=mc.DBResultRequest(ticket);
while (result.setStatus()==MobileHTTPStatementResult.WAITING){
    result=mc.DBResultRequest(ticket);
}
```

Figure 5: Code example of theJ2ME library to access a remote DB.

Another important issue to be considered was the amount of data send from the proxy server to the mobile client. One option was to use XML on data format, but it requires high end devices to parse the information, even when the device has enough memory and processor, it may affect the battery life. For this reason we did not use XML to sent data from the Proxy application to the mobile client. We defined our own format for data transfer.

3.4 A Backup/Restore mechanism

MIDP applications use record stores as their default storage mechanism. The Record Management System is a virtual layer that every vendor has to implement, and the use of this layer has an important limitation for developers, because they cannot access the device file system (if the device has a file system).

The first implementation of the J2MEMicroDB works using only the Record Management System provided by J2ME.The limitations of working only with record stores are the following:

- When first installing the mobile client application and loading the application data, the only way to provide the mobile client application with data is sending the data from the server using a wireless connection. Considering that there may be sudden disconnections and the cost of data transfer this approach may be limited.
- There is no way to do a partial backup on the mobile device, because there is no way of knowing where the record stores are located.

In this first implementation of the J2MEMicroDB the installation and recuperation process depends on the network connection. Another issue to be considered is the dependence on the DBMS to store application data. In some scenarios, when the mobile client application is simple, it may not manage relational data. Instead of storing data on a local database, the mobile application may be using files to store data. From the backup point of view, if no relational structure is used, we would need to convert these objects to XML and send them to a server using Webservices to back them up.

The second implementation of the J2MEMicroDB uses the File Connection Optional Package (FCOP) of the Generic Connection Framework [11] provided by J2ME to store either objects or relational tables. It provides: 1) the ability to backup data on a folder located on a memory card, 2) the Restore function to recover the backup data from the memory card and 3) storage for unlimited amount of data on the memory card. On the other hand, the limitation of these functions is that they are not available in the devices where there is no implementation of the File Connection protocol or not operating system or hardware support.

From the design point of view, the use of a layer-based architecture made the work simple. The PDAStore abstract class was extended to include the new functions.

4. APPLICATIONS OF J2MEMICRODB

Recent experiences and research in the field of mobile learning show the potential pedagogical benefits that a mobile extension of a web-based classroom can provide [12]. The J2MEMicroDB library is going to be used to access a Moodle LMS server [13]. Moodle is a well known Learning Management System with an Open Source community of more than 220.000 members. The main goal of the project called MoodleforMobiles was the design and implementation of a prototype of mobile client to access some information from the Moodle LMS (i.e. new internal mails).

The system architecture was the three-tier architecture described in section 3.1 with some small changes. One of these changes was the change of the J2EE proxy application server by a PHP application that could communicate with the Moodle server. The second major change was that the remote server is a Moodle Server instead of any DBMS. In the Moodle Server a new block that allows students to suscribe/unsubscribes to some course activities has been developed.

On the client-side, the mobile application that is currently being developed uses the J2MEMicroDB API to query the Moodle database. Theº data retrieved from the Moodle database are stored on the mobile device for offline access.

Apart from the context of e-learning, the J2MEMicroDB library has been selected as a caching mechanism for a commercial application: a mobile media player.

In order to improve his business of selling multimedia contents, a telecommunication operator has decided to develop a media player based on java. The selection of J2ME is based on the need to customize the media player with meta-information. This is not possible on media players integrated on the mobile phone.

The main goal is to manage meta-information about music on the server as well as on the mobile terminal. J2MEMicroDB has been selected to manage this meta-information.

The current version of the media player accesses a musical catalog using WAP. The main limitation of this solution is that the access using WAP is very slow. Keeping part of the catalog on a database on the mobile terminal permits a quicker access to the meta-information. The synchronization option of J2MEMicroDB allows the periodic actualization of this catalog.

5. CONCLUSIONS AND FUTURE WORK

Since the first release of J2MEMicroDB one year ago, this software library has become stable enough to support wireless applications embedded in mobile devices. These new upcoming applications will work on the currently used mobile devices, and

will be a very useful field of research and experimentation before the wireless data flat rate becomes widespread among the mobile users.

The further developments in J2MEMicroDB must go in tree directions as third part developers have suggested:

- Recode the SQL engine according to the JDBC interfaces, either to access the local (mobile) data and the server side data.
- Develop a feature to send and retrieve object stores from and to a server. Most likely this feature needs to be implemented using marshalling/unmarshaling techniques and complying with distributed object sharing standards.
- Add cryptography features to local data storage and communication channels.

A collection of best practices and code examples will be published soon.

6. ACKNOWLEDGMENTS

Our thanks to Dr. Miquel Barceló, Dr. Enric Mayol for their guidance and inspiration. Saul Cheung and Jordi López of ASUS IBÉRICA for technical advice and hardware for lab testing, and to Dr Xavier Franch and our researchers colleagues from the GESSI research group in UPC .

Our special thanks to Núria Lara, Jose Antonio Rodriguez and Néstor Giménez for her dedication to this project and their exceptional skills as developers. Finally we want to thank to all the students of the post degree course on "Mobile Application development" in FPC-UPC, and their coordinators Angels Tejada and Pilar Martinez.

This work has been partially supported by the Spanish project TIN2004-07461-C02.

7. REFERENCES

[1] Kumar, V. Mobile Database Systems, John Wiley & Sons, Inc. (2006), Hoboken, New Jersey.

[2] JDBC for CDC/FP Optional Package Specification v.10. [Online]. Available at: http://java.sun.com/javame/reference/apis/jsr169/

[3] Keogh, J. J2ME: The Complete Reference, McGraw-Hill/Osborne (2003), Berkeley.

[4] IBM toolbox for J2ME. [Online]. Available at: http://publib.boulder.ibm.com/infocenter/iseries/v5r4/index.jsp?topic=/rzahh/page1.htm

[5] TJopen. [Online] Available at: http://jt400.sourceforge.net

[6] PointBase Micro. [Online]. Available at: http:/http://www.pointbase.com/

[7] Oracle Database Lite Edition. [Online]. Available at : http://www.oracle.com/technology/products/lite/index.html

[8] SyBase Anywhere. [Online]. Available at : http://www.sybase.com/

[9] HSQL Database Engine. [Online]. Available at : http://www.hsqldb.org/

[10] Alier, M. Casado, P. Casany, M.J. JDMEMicroDB: A new Open Source Lightweight Database Engine for J2ME Mobile Devices. *In proceedings of the Multimedia and Ubiquitous Engineering (MUE'07)* (Seoul, Korea, April 26-28, 2007). Ed. IEEE Computer Society ISBN 978-0-7695-2777-2 on page(s). 247-252.

[11] FileConnection Optional Package 1.0 Specification Rev. 1.00- Final Release. [Online]. Available at: http://www.j2medev.com/api/fileconnection/index.html

[12] Sakkopoulos, E. Lytras, M. Tsakalidis, A. Adaptive Mobile Web Services Facilitate Communication and Learning Internet Technologies. *In proceedings of: Education, IEEE Transactions on*. Volume: 49, Issue: 2 On page(s): 208- 215. ISSN: 0018-935.

[13] Alier, M., Casado, P., Casany. M.J. A Mobile Extension of a Web Based Moodle Virtual Classroom. In *Proceedings of the e-Challenges Conference (e-Challenges 07)* (The Hague, The Netherlands, October 24-26-2007). www.eChallenges.org