

J2MEMicroDB: a new Open Source lightweight Database Engine for J2ME Mobile Devices

Marc Alier, Pablo Casado, M^a José Casany

Universitat Politècnica de Catalunya

UPC, C/Jordi Girona Salgado 1-3, Barcelona, E-08034, Spain

Tel: +34 93 4137885, Fax: + 34 93 4137833, Email: malier@lsi.upc.edu

UPC, C/Jordi Girona Salgado 1-3, Barcelona, E-08034, Spain

Tel: +34 93 4137885, Fax: + 34 93 4137833, Email: pcasado@lsi.upc.edu

UPC, C/Jordi Girona Salgado 1-3, Barcelona, E-08034, Spain

Tel: +34 93 4137888, Fax: + 34 93 4137833, Email: mjcasany@lsi.upc.edu

Abstract

J2ME is the development platform for mobile devices with larger support and availability in the market right now. Due to the minimalist definition of this architecture it does not support APIs for data persistence management, like object serialization, and the relational database access; neither to store data inside the mobile device nor to access a remote host.

This paper presents J2MEMicroDB, an Open Source development that implements Object Serialization, local relational database engine and a remote database access protocol that allows the access to any JDBC database.

In the design and implementation of J2MEMicroDB specific requirements and limitation of mobile devices have been considered. Even some performance improvements have been developed, like BTree indexing structures which improves significantly the efficiency, as the cross-platform presented benchmarking proves.

1. Introduction

The mobile industry is becoming more data-oriented. As a result, companies must determine how to deliver compelling applications to ensure that data services fulfill their potential.

Important issues to be considered to create these mobile applications are, in the first place, usability and accessibility. Mobile applications must be interactive and usable, because otherwise they won't succeed. They must also be multimedia applications, which may

access different types of contents. They must provide users with a quick response as well, because the mobile user is a "one-hand" user. This means that the user may be doing other things while using the mobile phone, so that his/her attention is not focused on the mobile device. And finally, mobile applications must be available online as well as offline (when the network is temporarily unavailable).

This last issue is especially important when dealing with the European network connection costs and network availability. The wireless connection cost is not low and there is a lack of high availability connections. Due to these limitations some application logic must be placed on the mobile device. It is also necessary that the mobile application may be able to work offline, because eventually there may not be network coverage. This is also important, because when the mobile application works offline there is no connection cost. In this scenario some information must be stored on the mobile device, so there must be some persistent data management mechanisms on the mobile application.

In order to choose the platform for the development of advanced mobile application, the platforms shown in figure 1 have been considered.

We wanted to develop advanced applications with sophisticated features that may be able to run on different types of devices, with a relatively low development cost.

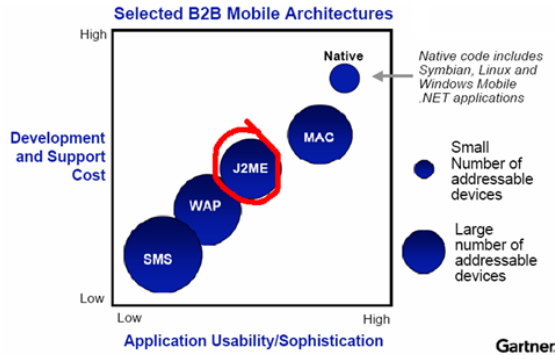


Figure 1: Development platforms.

One of the new technologies that can match the previous requirements is java. The group of mobile devices that support java applications has grown in the last years, becoming a de facto standard with a real presence in the market. The following figure shows how the sales of java-enabled mobile devices have grown in the last years.

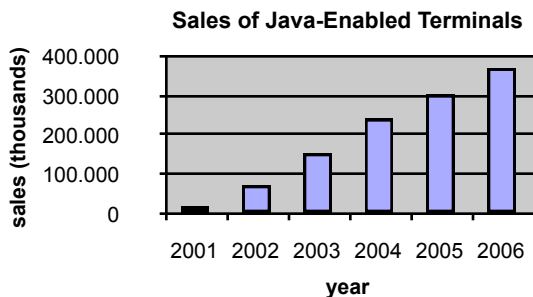


Figure 2. Sales of Java-Enabled Mobile Terminals Worldwide.

These java-enabled devices include a smaller specification of the java virtual machine (called the Kilobyte Virtual Machine (KVM) because its size is measured in Kilobytes not in Megabytes), in order to run java application on mobile devices. These java technology specific for mobile devices is called J2ME (Java 2 Micro Edition). Sun Microsystems presented in 1999 this new java technology designed for limited devices, like smartphones or PDAs (Personal Data Assistant). The features of mobiles devices, such as limited process capabilities, limited memory, reduced display or limited bandwidth, made necessary a new java edition [1].

J2ME is organized using a four layer architecture, composed of the following layers:

- Different virtual machines, one for each group of mobile devices.

- Different configurations: Connected Device Configuration (CDC) and Limited Connected Device Configuration (CLDC). CLDC includes the KVM.
- Different profiles. The MIDP (Mobile Information Device Profile) is a specific profile for the CLDC configuration. The MIDP 2.0 was presented in 2003 and is implemented in almost every mobile phone released nowadays (or at least it can be installed, like on PalmOS or Windows Mobile devices).
- Optional packages.

J2ME is a very recent technology, so it only provides developers with the very basic tools. In order to succeed, J2ME needs that vendors implement virtual machines for any mobile device. While mobile phone vendors usually integrate java in their products, PDA vendors usually prefer other alternatives, and it is usually necessary to install third-part virtual machines like IBM J9 for Palm OS and Windows Mobile [2].

Among the java applications available for mobile devices we focused on mobile applications that require database access, either on the mobile device for local and offline storage, or online access to a remote host database. In order to expand business applications to the mobile scope, it is necessary to store and retrieve persistent information on the mobile device as well as access remote information stored on a remote DBMS host from the mobile device.

The storage of information on the mobile device is necessary because a J2ME application can stop suddenly due to different reasons, such as, an incoming call phone or because the device runs out of battery.

1.1. Database support in J2ME

As strange as it sounds J2ME does not provide developers with any means to handle databases. JDBC (Java Database Connectivity) is the generic mechanism for accessing databases from J2SE (Java 2 Standard Edition) and J2EE (Java 2 Enterprise Edition) applications [3]. But why isn't there any JDBC connection for J2ME? The answer is simple, because the classes that implement a JDBC connection are too heavy for mobile devices, therefore they cannot be executed in such limited devices. This is definitely a handicap for the developers of mobile information retrieval applications.

Besides the storage mechanisms on the mobile device are quite limited. There is only a low level API to handle persistence. In the foundation profile, of the CDC configuration, plain random access files are used to store data. In the MIDP profile (versions 1.0, 2.0 and the recently released MIDP 2.1) of the CLDC configuration, only record stores are provided. These record stores require that the developer uses a low level API, which seems contradictory considering that java is an object oriented language.

1.2. Database development tools for mobile devices

Nowadays there aren't available DBMS for any mobile device. There are microDBMSs specifically created for some mobile devices such as Palm OS [4], Symbian [5] 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 non existence 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 [6]. There is also an Open Source version of this package called JOpen [7], which provides developers with a small driver JDBC 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 [8]. CodeBase for J2ME is a high speed database engine for J2ME. It allows developers to create fully functional database applications [9]. Oracle lite is a small version the Oracle DBMS software that provides access to remote databases and to data stored on a local database [9]. Finally Sybase Anywhere 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.

1.3. Development goal

The state of the art study revealed that there is a need for an Open Source high level data management and database access set of tools. That is the goal of our project called J2MEMicroDB.

J2ME was selected as the platform for our development, because java application can run on any mobile device that includes a KVM. Besides we decided to develop our project by the Open Source Standard Parading, because a persistent database engines must be as standard as possible, but if there is not standard solution, developers must have access to the code. The access to the code provides developers with some advantages:

- No (completely) dependence on vendors when assistance is required.
- Lower development cost.
- Lower risks.

- From the university research point of view, share knowledge, maximizes use and impact.

2. J2MEMicroDB

J2MEMicroDB is the name of the Open Source project that has been started in the UPC (Universitat Politècnica de Catalunya) in order to develop a J2ME solution to extend the functions of the J2ME persistent API, and allow access from mobile applications to remote and local databases.

2.1. Goals

J2MEMicroDB is a database engine for mobile devices that has the following features:

- Is lightweight, a very important feature because it has to run on devices with limited storage capabilities.
- Is distributed under a Free and Open Source license like GPL (General Public License).
- Is efficient, because provides reduced time to access the information requested.
- Implements a relational database on a mobile device.
- Implements a persistence engine to easily store and retrieve Objects.
- Accesses a remote DBMS host through a web service and stores remote information on materialized views on the mobile device.

The current implementation of J2MEMicroDB runs on CLDC devices using the MIDP 1.0 profile or above.

2.2. J2MEMicroDB architecture

In this section we present the layer architecture of the J2MEMicroDB engine.

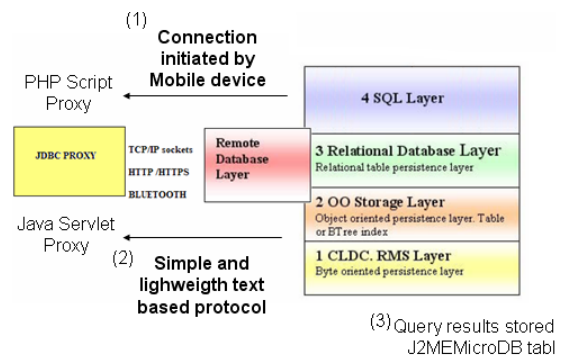


Figure 3. J2MEMicroDB architecture.

Figure 3 shows the layer architecture we developed in the J2MEMicroDB project. The lower layer (RMS layer) is provided by J2ME.

In the first step of the project the limitations of the J2ME API for CLDC devices with MIDP 1.0 profile were studied. One of the most important limitations of the J2ME API is its persistent storage capability. Among the packages of J2ME for CLDC devices with MIDP profile, there is one, javax.microedition.rms, which is used for storing and retrieving information from mobile devices. This package offers a set of classes and interfaces in order to manage persistent storage on a mobile device [1].

The javax.microedition.rms package is based on Record Stores. A Record Store is the equivalent of a simple file. A Record Store stores a set of records in binary format. Although this package allows developers to store and retrieve information to/from files on mobile devices, it has some limitations:

- It is not possible to directly store objects in a Record Store. Data is stored in a Record Store as a sequence of bytes. That is an important limitation for java developers because they usually need to manage objects in their applications.

- Very limited search capabilities in the Record Store. Each Record stored in a Record Store has an identifier (called recordId). The searches of data are bases on this (recordId).

Due to these limitations we developed a Object Oriented (OO) storage layer above the RMS layer. This layer provides four types of files (Direct access, Sequenced Access, Key value access, Key value sequenced access) to store objects in a Record Store, and improve the access and search of data among the objects. The most important benefit of this layer is the possibility of object serialization.

In the development of this layer we faced the problem of providing programmers with a “easy to use” OO API. To do so we had to implement a mechanism to translate the objects the programmer manages, to the data stored in the RMS file provided by J2ME. Figure 4 shows a schema of this translation mechanism. Let us imagine that the programmer of the OO layer creates a contact file. The data of every contact are encapsulated in an object which as an assigned label. The programmer adds three registers to the file and assigns the following labels to each contact: Anna, Mike and Thomas. As shown in the upper part of figure 4, the programmer adds an object with the data of each contact as well as the label. This label will be used to retrieve the data associated to a contact. In order to translate the objects and labels to the real data stored in the RMS file, we used two indexing mechanisms. The first indexing mechanism (Index Table) consists of a table, which stores the physical position of a record in the RMS file related to a label. The second indexing mechanism, used to

improve the performance of the file in the search processes, uses a BTree [10] instead of a table to store physical positions of records and their related labels. The performance of these indexes is compared in section III.

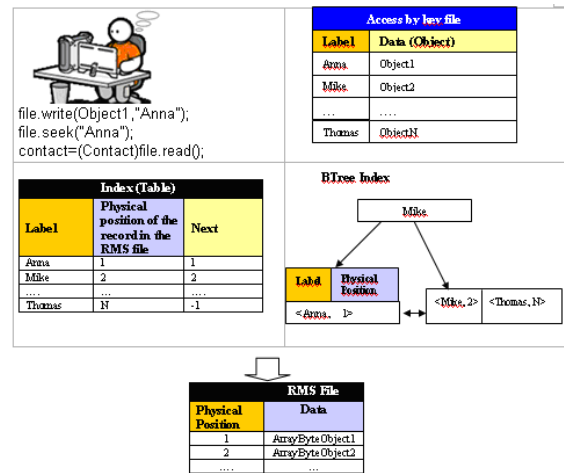


Figure 4. OO Storage Layer operation.

The first indexing mechanism is used when the persistent storage capacities of the device are very limited while the second indexing mechanism is used when more persistent storage capacities are available to improve the searches.

With this layer the developer is able to store and retrieve objects from a file.

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.

And, finally, the fourth layer gives a limited SQL interpreter to query the local database as well as methods to access a remote database, query it and store data on the mobile device as materialized views. This part of the API is described in detail in section 2.3.

With the set of classes and interfaces provided by J2MEMicroDB a J2ME developer can easily handle the persistence of objects in J2ME. J2MEMicroDB provides developers with a relational database engine that can run on any J2ME mobile device based on the CLDC configuration and MIPD 1.0 profile or above.

2.3. Send queries from your mobile device to your DBMS host

The ability to manage data locally on the mobile device is necessary to develop advanced application but not enough. There is a need to access a remote database on a host using any kind of network connection:

- GSM, GPRS, 3G, WIFI, through TCP/IP sockets.

- GSM, GPRS, 3G, WIFI, through the HTTP or HTTPS protocol.
- Bluetooth.

The J2MEMicroDB also provides access to remote DBMS servers. In particular, the mobile client that uses this API is able to:

- Establish secure connections with the remote host.
- Execute SQL queries in the remote host. The results can be stored locally on the mobile device as materialized views.
- Execute SQL statements (INSERT, DELETE and UPDATE, etc) in the remote host.

In order to access a remote DBMS host from a JME application we had to face the problem described in section 1.1. It is not possible to connect directly the J2ME application with the remote host using JDBC. So we defined a 3-layer architecture composed of the following items:

- The J2ME application or mobile client that has to access the remote DBMS host.
- A proxy server, a web service which accepts connection requests from the J2ME client and redirects them to the DBMS host. There are two current versions of this application: a J2EE application with all the java functionalities including JDBC and a PHP program used to access a Moodle server.
- The remote DBMS host that accepts requests from the proxy server via JDBC.

The Proxy server application is a bridge between the J2ME application and the DBMS host. Using this layer architecture the only thing we had to do was establish the communication between the J2ME client and the proxy server. After that, the proxy application can connect to any SGBD with the proper JDBC Driver (like ORACLE, IBM's DB2 or MySQL which we used in our tests).

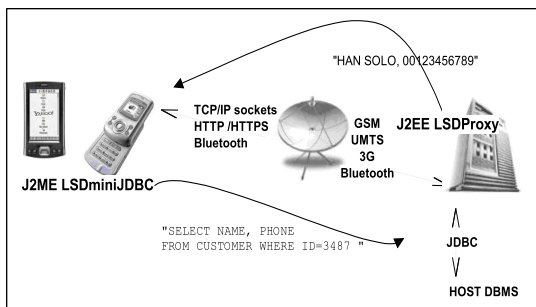


Figure 5. Connection to DBMS host schema.

We had to establish an asynchronous communication between the J2ME client and the Proxy server because the connection may be severed suddenly.

3. Benchmarkings

So far we have tested the performance of the second (OO Storage Layer) and third (Relational Database Layer) layers described in section 2.2.

In order to test the performance of the tables that store persistent data on the mobile device we designed a J2ME application called J2MEMicroDBTests.

J2MEMicroDBTests is based on J2MEUnits [11] which is the version of the JUnit [12] framework for J2ME.

J2MEMicroDBTests implements 3 tests performed on a single table database. The first test is used to insert 100 or 1000 rows into a relational table. Afterwards, the search of records by key are tested. We search 100 or 1000 record based on a random alphanumeric key. Finally, the 100 or 1000 records of the table are updated. The tests estimate the required time to insert the 100 rows, to search them and update them. Finally the total time of the process is calculated.

The tests have been performed on on these PDAs: PalmTungsten TX, Palm Tungsten C and HP iPAQ h6340. The results of those tests are shown on the following tables:

Table 1. Test results for 100 rows with no BTree.

	Tungsten TX (63,8Mb free memory)	Tungsten C (50Mb free memory)	Tungsten C (34,5Mb free memory)	HP iPAQ h6340 (57M free memory)
	IBM J9	IBM J9	IBM J9	IBM J9
Inserting 100 rows	1020	560	1150	16
Searching 100 aleatory keys	300	290	280	9
Performing 100 updates	1370	620	570	137
Total time	6190ms	2260ms	3130ms	18993r

Table 2. Test results for 1000 rows with no BTree.

	Tungsten TX (63,8Mb free memory)	Tungsten C (50Mb free memory)	Tungsten C (34,5Mb free memory)	HP iPAQ h6340 (57Mb free memory)
	IBM J9	IBM J9	IBM J9	IBM J9
Inserting 1000 rows	14510	12140	12570	2755
Searching 1000 aleatory keys	8450	8910	8740	1769
Performing 1000 updates	13360	5750	5700	896368
Total time	51800ms	34890ms	35820ms	1007351ms

As expected the tests results based on the table index described on section II, subsection B, are slower than the tests performed on a BTree index especially in the searches.

Table 3. Test results for 100 rows using BTree.

	Tungsten C (34,5Mb free memory)	HP iPAQ h6340 (57Mb free memory)
	IBM J9	IBM J9
Inserting 1000 rows	850	1678
Searching 1000 aleatory keys	200	957
Performing 1000 updates	680	12472
Total time	2640ms	17900ms

Table 4. Test results for 1000 rows using BTree.

	Tungsten C (34,5Mb free memory)	HP iPAQ h6340 (57Mb free memory)
	IBM J9	IBM J9
Inserting 100 rows	7710	19979
Searching 100 aleatory keys	2000	8667
Performing 100 updates	6610	878910
Total time	19460ms	970131ms

Besides, the results show that the tests run on a mobile terminal with higher amount of free memory are faster than the ones performed on the same devices but with less free memory.

4. Future related work

The next step has been the application of the J2MEMicroDB to develop a new mobile application.

We wanted to create a new mobile client which accesses a traditional web application, the forums of Moodle [13]. Moodle is a Learning Management System (LMS), distributed freely under General Public License (GPL).

In this project, first, we had to establish the communication mechanism to query the Moodle database. After that, we retrieve this information and send it to the mobile client. Finally the mobile client stores this information locally in a relational database. The persistent storage mechanisms allow users to access the information of the forums of Moodle offline.

An optional feature implemented in J2ME is the capability to access the mobile device file system. This allows the programmer to take advantage of storage devices with large space, such as SSD Memory cards. Another step in our development will be to implement a backup-restore feature to save the data in J2MEMicroDB, and finally to implement the physical storage layer in the SSD when the feature is available.

5. Conclusions

This project has succeeded in creating a tool for the developers of mobile application to develop database enabled and connected applications. This technology can be used to develop mobile applications that feed from existing web applications and webservices layers, so very little development must be done in the server side.

Since the J2ME application can store and manage its own data, the mobile application can work offline, so the mobile user can control the access to online information. This can reduce the connection costs.

Further work on this field will consist of developing real applications and keep in touch with the community of J2ME developers, providing them with documentation and technical support, as well as analysis of the feedback reported to explore future opportunities for improvement.

6. Acknowledgements

Our thanks to Dr. Miquel Barceló, Dr. Enric Mayol for their guidance and inspiration. Thanks also to Jordi Marca of Palm, 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 Nuria Lara, Jose Antonio Rodriguez 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.

10. References

- [1] James Keogh. J2ME The Complete Reference. McGraw-Hill/Osborne.
- [2] Windows Mobile. <http://www.microsoft.com/spain/windowsmobile/default.mspx>
- [3] Sun Microsystems, Inc (1996,1997). JDBC Guide: Getting started. JDK Online Documentation.
- [4] Palm OS. <http://www.palm.com/>
- [5] Symbian. <http://www.symbian.com>
- [6] IBM toolbox for J2ME. <http://publib.boulder.ibm.com/infocenter/iserics/v5r4/index.jsp?topic=/rzahh/page1.htm>
- [7] TJopen. <http://jt400.sourceforge.net>
- [8] PointBase Micro. <http://www.pointbase.com/>
- [9] CodeBase for J2ME. <http://www.codebase.com>
- [10] R. Ramakrishnam. Database Management Systems. Boston: McGrawHill.
- [11] JUnit project. <http://www.junit.org/index.htm>
- [12] JUnits Project for J2ME <http://j2meunit.sourceforge.net/doc.html>

