Programming mobile devices with J2ME

E. Roldán, E. Montón, S. Guillén
ITACA Institute
edrolher@upvnet.es, *emonton@upvnet.es*, *sguillen@upvnet.es*

Abstract

Since its beginning, J2ME is a very used language programming to develop applications for mobile devices. Its particular architecture is prepared to work in lots of device kinds, with more or less resources. However, J2ME have some objections that have to be had into account.

1. Introduction

Java 2 Micro Edition (J2ME) is a programming language aimed at the development of applications for mobile devices, such as mobile phones or PDAs. It is based on Java language, with a light version of J2SE´s API in order to execute applications in low resource devices.

Java is a programming language used in the development of applications for several machines, with different resources and capabilities. Due to this, Java programming is divided in three groups:

- Java 2 Standard Edition (J2SE): For personal computers.
- Java 2 Enterprise Edition (J2SE): Adds to J2SE features oriented to the corporative development.
- Java 2 Micro Edition (J2ME): The most lightweight distribution of Java, aimed at the development of applications for very reduced devices. This distribution is the most portable of all.

J2ME has an architecture with different layers and different possibilities by layer, in order to work as better as possible in most of reduced resources devices. J2ME´s architecture is composed by a first layer, named Configuration Layer, where each kind of configuration is prepared for a kind of devices; a second layer, Profile Layer, where each profile is prepared for a family of devices, and finally, a third layer with the different application programming interfaces for a determinate device.

As J2ME is very portable, the families of devices than can use J2ME applications are a lot, such as mobile phones, PDAs… but also notebooks and PCs too.

2. Architecture

The J2ME´s architecture is divided in three layers:

- Configuration Layer
- Profile Layer
- Application Layer

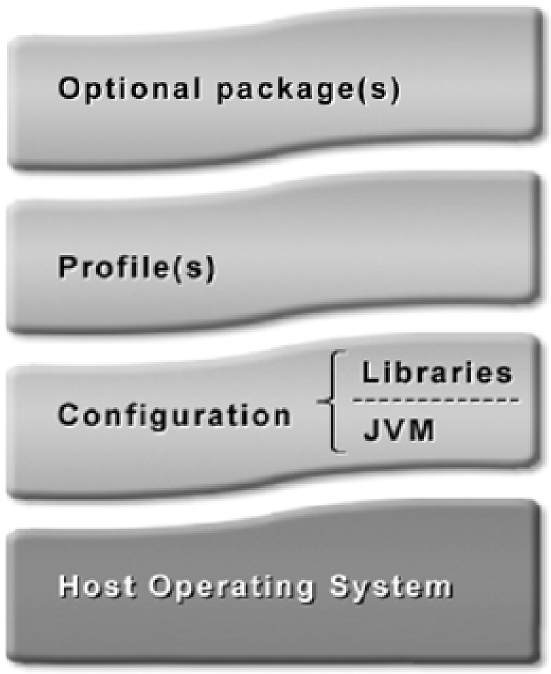


Figure 1. J2ME Architecture

2.1. Configuration layer

The first layer encountered upper the Operating System is the Configuration Layer. There is the Java Virtual Machine (JVM) and the basic set of libraries to make programs. In J2ME, we have the below configurations.

2.1.1. CLDC 1.0

Connected Limited Device Configuration. Used in very limited resources devices (128-512KB of memory). It works with the KVM (Kilobyte Virtual Machine) and it is not compatible with J2SE. Due to the limited resources included in this configuration, it doesn't allow the use of basic characteristics, as neither the float or double types nor the JNI interface. The classes that this layer provides are:

- java.lang.*
- java.io.*
- java.util.*
- java.microedition.io.*

2.1.2. CLDC 1.1

It is the new version of the CLDC configuration. It includes new features as floating point type and weak references.

2.1.3. CDC

Connected Device Configuration. Used in devices with more resources than in the case of CLDC (2MB of memory). It works with the CVM (Compact Virtual Machine) and is compatible with J2SE. In fact, this configuration uses a reduced set of the JDK 1.3 libraries. Some implementations of this configuration allow the use of the JNI. The classes that this configuration provides are:

- Java.io.*
- Java.lang.*
- Java.math.*
- Java.net.*
- Java.security.*
- Java.text.*
- Java.util.*
- Java.microedition.io.*

2.2. Profile layer

The next layer of the architecture is the Profile layer. This layer gathers a set of libraries oriented to a determined application field. It determines the application's life cycle, application's user interface…

There is available a lot of profiles, but the most important profiles are two: the MIDP (used usually on mobile phones) and the Personal Profile (used usually on PDAs).

2.2.1 MIDP

Mobile Information Device Profile. This profile works with the CLDC configuration. It is oriented to devices with the next characteristics:

- Low computational and memory capacities.
- Low connectivity (9600bps).
- Low graphic capacity.
- Data input alphanumeric very reduced.
- 128KB of memory for MIDP components.
- 8KB of memory for persistent application data storage.
- 32KB of memory for the Java stack in runtime.

This profile provides the next classes:

- javax.microedition.lcdui.*
- javax.microedition.rms.*
- javax.microedition.midlet.*
- javax.microedition.io.*
- java.io.*
- java.lang.*
- java.util.*

The version 1.0 of this profile gives two APIs in order to create a user interface: the high level API and the low level API.

The high level API allows the creation of forms and to put several components on it, in the same way than the AWT libraries in J2SE. The components available are: Choice groups, gauges, tickers, date fields, text fields, image items, text items and string items. Instead of forms, other predefined screens, as a list, a textbox and an alert screen can be used.

The low level API provides the control all the potentiality of the screen, accessing to each pixel individually. In this case, the Canvas class allows the access to the screen and it is available a Graphics object to draw on that. In addition, the API allows a total screen control, allowing a total keyboard control, returning the keycode of the key pressed, for all the keys presents in the device. Using this API, it is possible to create new components for the high level API. The disadvantages of this profile basically that making user interfaces with it is more complicated that using the high level API. Furthermore, the user interfaces made with this API are more dependents of the device that the high level API, because not all the devices have the same screen size or the same buttons.

Apart of the user interface, the profile gives classes in order to make network connections, using the connections systems that the device has, like GPRS or Wi-Fi. In the version 1.0 of the MID Profile, only the HTTP 1.1 protocol is implemented, whereas in the MIDP 2.0 are implemented the sockets, datagrams and HTTPS protocols as well as the CommConnection, an interface to access to serial ports, and the StreamConnectionNotifier, that allows the configuration of the mobile device as a server to receive incoming connections. The problem of this configuration in this type of devices is that usually the IP address assigned to the device normally is not fixed.

In order to store data in persistent mode, the MIDP provides the RMS system, that is, the Record Management System that allows the storage of data in an index table. Each midlet (MIDP application) can create its own RMS zone, and only the midlets that are in the same suite can share a RMS zone. The RMS data is stored in binary files into restricted area of the device's memory map. The RMS operations are atomics, synchronized and serialized.

At this moment, there are two MIDP versions, the MIDP 1.0 and the MIDP 2.0. The 1.0 version has the features described before, and the 2.0 version includes the next new features:

- Secure networking
- Multimedia
- Form enhancements
- Game API
- RGB images
- Code Signing and Permissions
- Push Registry

The MIDP applications have a special life circle. The application has three states: Active, Paused and Destroyed. When the application starts, the startApp() method is called. When an incoming call arrives, the application goes to the Paused state, calling to the pauseApp() method in order to close open connections or open files. When the call finish, then startApp() is called again and the application goes to the Active state. Finally, when the application is going to be closed, the destroyApp() method is called and the application goes to Destroyed state.

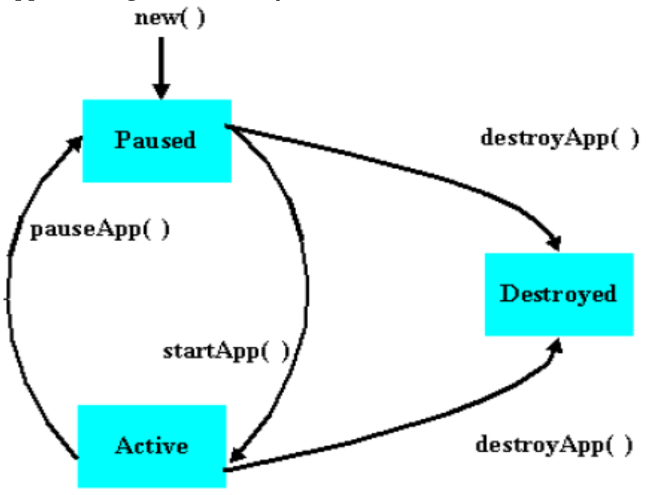


Figure 2. MIDP Applications Life Circle

2.2.2 Personal Profile

This profile works with CDC configuration, and is oriented to devices that require full GUI or Internet applet support, such as PDAs. The user interface is based on the AWT libraries, and the Personal Profile is based on a subset of JDK 1.3, being compatible with J2SE applications.

In most of cases, this profile support JNI (Java Native interface), and then it is possible the installation of own APIs, even if that API need to access to hardware resources. This profile allows the development of applications, applets and Xlets.

2.3. Top layer

The last layer of the J2ME architecture is the application layer. This layer contains your application and all the application program interfaces (APIs) that you can use in your application. The most used APIs are:

- JSR-82: Bluetooth API.

This API allows application developers the use of the Bluetooth interface of the device. With this API it is possible to search devices and services, as well as the exchange of information with others devices. However, some implementations of this API don't implement the OBEX protocols, being necessary to implement them in order to be able to communication by means of this system.

- JSR-120: Wireless Messaging API

This API allows application developer to send and receive SMSs. Is very used in mobile phones

- JSR-205: Wireless Messaging API 2.0

The new version of the JSR-120 API. It provides delivery and reception of MMS, in addition to SMS messages.

- JSR-135: Multi-Media API

The most used API in mobile phones. This API allows you application to manage the multimedia resources of the mobilde device, like the speaker, the microphone and the camera. It is necessary to highlight that some implementations doesn't includes the access to the camera, or includes the access but only to take snapshots, but not video. In addition, the quality of the images captures is in general very low

- JSR-172: Web Services API

This API allows application developer to use web services from a server. Actually, this API is only a SOAP implementation, because if it is not possible to access to the network, it is not possible access to web services, That is, this API doesn't control any hardware resource, only implements a protocol.

- JSR-72: File Connection and PIM API

This API allows application developer to access to physical files in your device, both for reading and for writing. This API provides access to determinate resources like the device's contact agenda.

Others existing APIs are: RMI, USB, Speech, Location, SIP, Mobile 3D Graphics, Data Sync, Encryption, XML parsers...

This layer contains the libraries to be used by the application. For example, libraries make by the same developer or by others in order to encrypt, to implement a protocol, to compress... Some available libraries are the kSOAP libraries, which allow application developer to access to web services or the Derby database, which is a 100% Java database implementation fully compatible with J2ME Personal Profile.

3. Development Environments

In order to create a J2ME application it is necessary an editor to edit the source code, a preverifier to preverify the classes, a compiler to compile the java source and a simulator to simulate our applications. These tools can be obtained individually or in a integrated environment.

Below, two example environments to create MIDP applications are described:

3.1. J2ME Wireless Toolkit

This free tool provides the compiler, the preverifier and the simulator. It is only necessary an editor to edit the source code and allows the creation of MIDlets.

3.2. Sun One Java Studio

This tool is an Integrated Development Environment (IDE) that provides all the necessary to create MIDlets. Nowadays, this utility is integrated with the NetBeans IDE in the plug-in named Mobility Pack.

In the case of Personal Profile applications we there are following environments:

3.3. WebSphere Studio Device Development

Created by IBM based on Eclipse, this environment allow developers to create application, applets and Xlets in order to be run in a device with the J9 Virtual Machine, the JVM created by IBM for some devices like Pocket PCs.

3.4. J2SE Development Kit

It is possible to develop an application for J2SE but only using the J2ME Personal Profile classes subset. Therefore, that application could be run in a J2ME Personal Profile environment without problems.

4. Devices

The principal advantage of J2ME is its portability. it is possible to run J2ME applications in any device where you can put a JVM. Then, J2ME application can operate in lots of devices like the followings:

4.1. Mobile phones

Some manufacturers that include a JVM in some of their mobile phones are Nokia, Alcatel, LG, Samsung, Sanyo, Siemens, Sony Ericsson...

4.2. PDAs

Both Pocket PCs and PALMs, but the flexibility is higher in the case of Pocket PCs than in PALMs.

4.3. PCs

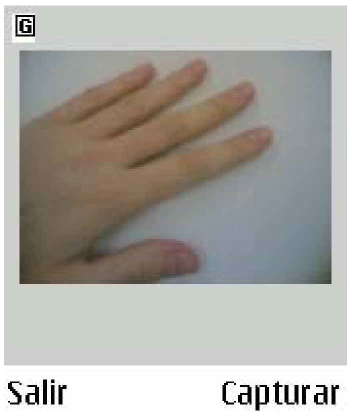
J2ME is designed to run in reduced resources devices, but applications can run in desktop or notebook computers perfectly.

5. Practical cases

5.1. DERMA

This application, carried out in the TSB group, allows the user to capture photographs and fill forms in order to send that information to a server using the GPRS network.

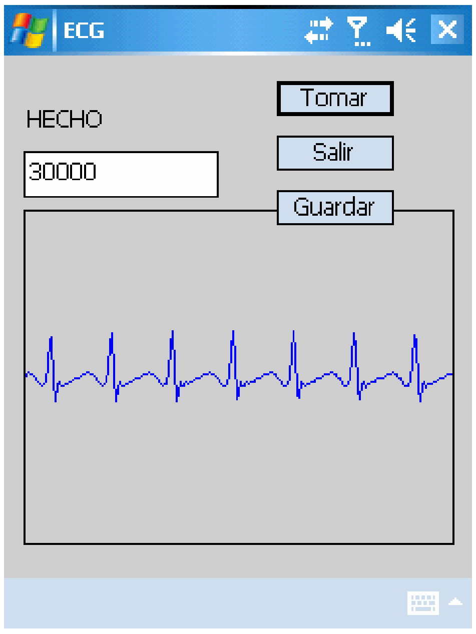
The application was made using the MultiMedia API and MIDP 2.0 in a Nokia 6630 mobile phone.



The principal problem was the low snapshot quality of the MMAPI, because this API only allows applications to capture photographs with the size of the phone screen, and in the option given to use bigger photographs, the API captures the photograph with the screen size and then resize it digitally. Then, the quality of the photograph is very low.

5.2. LyraPDA

This application, done in the TSB group too, allows the user to have information about several patients, with personal information and vital signal measures.



The application has a database integrated, uses web services to communicate with the server, and an encryption system to store and to send the data. In addition, it can control the monitoring device using a Bluetooth link that allows the measurement of the patient's pulse, oxygen saturation, blood pressure and ECG.

In order to do this, the application was carried out over a Pocket PC PDA (QTEK 9100) using the IBM J9 Virtual Machine with the Personal Profile. To control the Bluetooth resources, it was needed to include an external Bluetooth API (that was possible because J9 Personal Profile allows the JNI), and the use of the database was possible because the inclusion of a a JDBC package driver.

In the communication, the use of a library with an implementation of the SOAP protocol and other with the HTTPS protocol was necessary, in order to access web services using an encrypted connection.

The problem encountered was the memory management, because the application fills the memory faster than the garbage collector can free it, and if the garbage collector is called manually, it is not able to free all the memory used by unreferenced objects.

6. Conclusions

In conclusion, we can define J2ME like a language programming that is an alternative to develop applications for reduced resources devices, with the advantages that is easy to learn and fast to develop applications, with lots of APIs designed to help us in the most common tasks. However, this language has some disadvantages that need to be had into account. The access to the hardware resources is restricted to the APIs that the programmer can use. The memory management is done by the JVM and sometimes it is not as good as would be. Some APIs are not 100% implemented, like the MultiMedia API. And finally, if we cannot access to the JNI, we are restricted to the Java sandbox. If these constraints are not restrictive to implement one particular application, J2ME is the best option and the easiest way to develop that application.

7. References

[1] S. Gálvez, L. Ortega, "Java a Tope",
Electronic Edition.

[2] J. de M. Hernández, "Introducción a J2ME"
www.todosymbian.com

[3] "Introduction to mobile Java technology"
http://developers.sun.com/techtopics/mobility/getstart/

[4] J. Knudsen, "What is new in MIDP 2.0",
http://developers.sun.com/techtopics/mobility/midp/articles/midp20/

[5] A. Froufe, P. Jorge, "J2ME: Manual de Usuario y Tutorial",
Ed. Ra-Ma.

[6] "Datasheet Java 2 MicroEdition"
http://java.sun.com/j2me/j2me-ds.pdf