Think small with J2ME

Java platform offers opportunities on small, networked devices

Soma Ghosh (sghosh@entigo.com)
Senior Application Developer, Entigo
November 2001

The Java 2 Platform, Micro Edition (J2ME) offers great tools for developers, porting the Java platform's network-centric and platform-agnostic worldview down to memory- and processor-limited devices. Soma Ghosh explains the basics of the J2ME world, showing you the building blocks of the platform and demonstrating a sample application.

Personalized and intelligent information appliances are necessities in our life today. These appliances, which include cell phones, two-way pagers, smart cards, personal organizers, and palmtops, tend to be special-purpose, limited-resource, network-connected devices, and not the general-purpose desktops we have known until now. To specifically address this vast consumer space, the Java 2 Platform, Micro Edition (J2ME) provides a plethora of innovative Java technologies.

Transition of the Java VM: From desktop to microdevice

The microdevices that J2ME targets have 16- or 32-bit processors and a minimum total memory footprint of approximately 128 KB. They conform to a *Connected Limited Device Configuration* (CLDC) while maintaining the Java tradition of anytime, anywhere code portability, deployment flexibility, safe network delivery, and code stability. The prerequisite for the J2ME CLDC is a stripped-down JVM, called the *K Virtual Machine* (KVM). The KVM is designed for small-memory, resource-constrained, network-connected devices.

Another J2ME configuration, the *Connected Device Configuration* (CDC), targets advanced consumer electronic and embedded devices such as smart communicators, advanced "smart" pagers, smart personal digital assistants (PDAs), and interactive digital television set-top boxes. Typically, these devices run a 32-bit microprocessor/controller and have more than 2 MB of total memory for the storage of the virtual machine and libraries. The CDC contains the C Virtual Machine (CVM). In this article, we will focus on CLDC and KVM architectures. For more information on the CDC and CVM, see the Resources section below.

The KVM has been adapted to the features of small-footprint devices in the following ways:

- VM size and class libraries have been reduced to a standard of 50 to 80 KB of object code
- Memory utilization has been reduced to a standard of tens of kilobytes

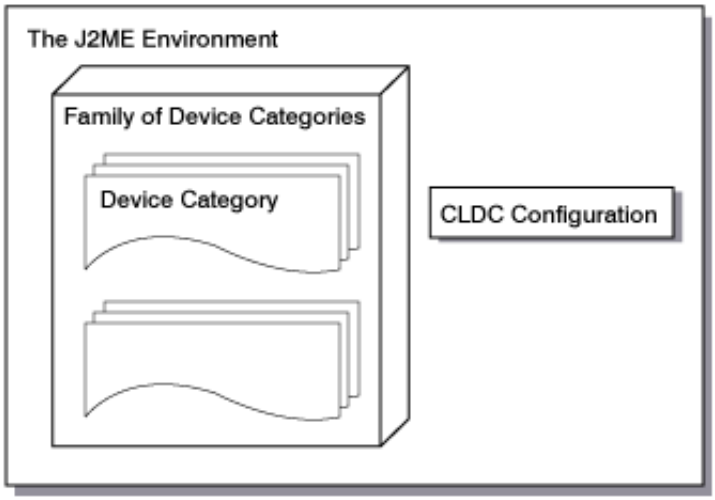
- Performance is effective on devices with 16- and 32-bit processors
- Architecture is highly portable, with a small amount of machine- and/or platform-specific code
- Multithreading and garbage collection are system independent
- Components of the virtual machine can be configured to suit particular devices, thus increasing flexibility

J2ME architecture and configuration

The J2ME architecture is based on *families* and *categories* of devices. A category defines a particular kind of device; cellular telephones, simple pagers, and organizers are separate categories. A family of devices is made up a of a group of categories that have similar requirements for memory and processing power. Together, cellular phones, simple pagers, and simple personal organizers make up a single family of small-footprint devices.

Figure 1 defines the relationship between the families and categories of devices in the context of J2ME.

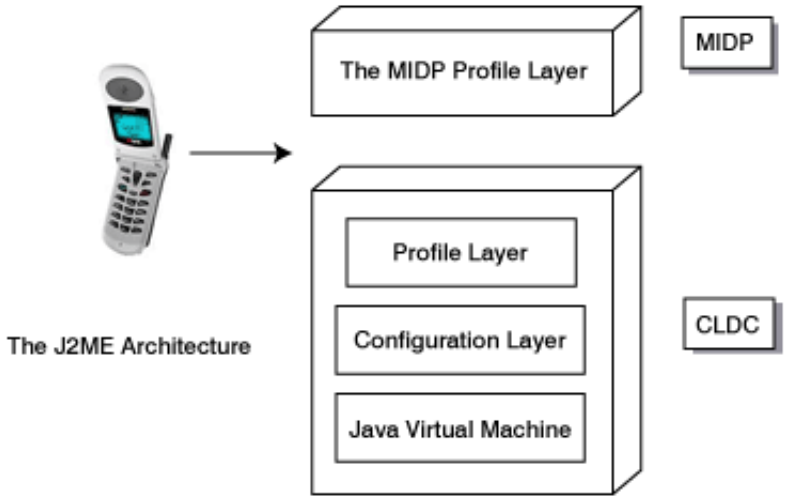
Figure 1. Families and categories of devices



In order to support the kind of flexibility and customizable deployment demanded by the family of resource-constrained devices, the J2ME architecture is designed to be modular and scalable. This modularity and scalability is defined by J2ME technology in a complete application runtime model, with four layers of software built upon the host operating system of the device.

Figure 2 shows the J2ME architecture.

Figure 2. The J2ME architecture



- **Java Virtual Machine layer:** This layer is an implementation of a Java Virtual Machine that is customized for a particular device's host operating system and supports a particular J2ME configuration.
- **Configuration layer:** The configuration layer defines the minimum set of Java Virtual Machine features and Java class libraries available on a particular category of devices. In a way, a configuration defines the commonality of the Java platform features and libraries that developers can assume to be available on all devices belonging to a particular category. This layer is less visible to users, but is very important to profile implementers.
- **Profile layer:** The profile layer defines the minimum set of application programming interfaces (APIs) available on a particular family of devices. Profiles are implemented upon a particular configuration. Applications are written for a particular profile and are thus portable to any device that supports that profile. A device can support multiple profiles. This is the layer that is most visible to users and application providers.
- **MIDP layer:** The Mobile Information Device Profile (MIDP) is a set of Java APIs that addresses issues such as user interface, persistence storage, and networking.

The Java Virtual Machine layer, configuration layer, and profile layer together constitute the Connected Limited Device Configuration (CLDC). The MID Profile and CLDC provide a standard runtime environment that allows new applications and services to be dynamically deployed on end-user devices.

Programming J2ME with MIDP APIs: The building blocks
The combination of CLDC and MIDP provides a complete environment for creating applications on cell phones and simple two-way pagers.

The core of a MID Profile is a MIDlet application. The application extends the `MIDlet` class to allow the application management software to control the MIDlet, retrieve properties from the application descriptor, and notify and request state changes.

All MIDlets extend the `MIDlet` class -- the interface between the runtime environment (the application manager) and the MIDlet application code. The MIDlet class provides APIs for invoking, pausing, restarting, and terminating the MIDlet application.

The application management software can manage the activities of multiple MIDlets within a runtime environment. In addition, the MIDlet can initiate some state changes by itself, and notify the application management software of those changes.

The whole set of MIDP API classes can be broken down into two categories:

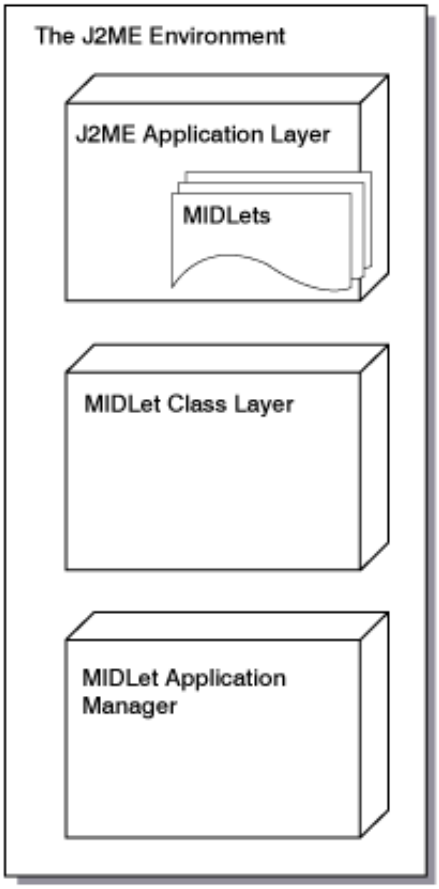
- **MIDP APIs for the user interface:** These APIs are designed so that interaction with the user is based around a succession of screens, each of which presents a reasonable amount of data to the user. Commands are presented to the user on a per-screen basis. The APIs allow the application to determine what screen to display next, what computation to perform, and what request to make of a network service.

- **MIDP APIs for handling the database:** These APIs organize and manipulate the devices database, which comprises information that remains persistent across multiple invocations of the MIDlet.

The underlying CLDC API is used to handle strings, objects, and integers. A subset of the Java 2 API is also provided to handle I/O and network communications.

Figure 3 shows the building blocks of J2ME.

Figure 3. The building blocks of J2ME



The relationship between the standard and micro edition Java APIs is shown in Figure 4.

Figure 4. The relationship between the J2ME and J2SE APIs

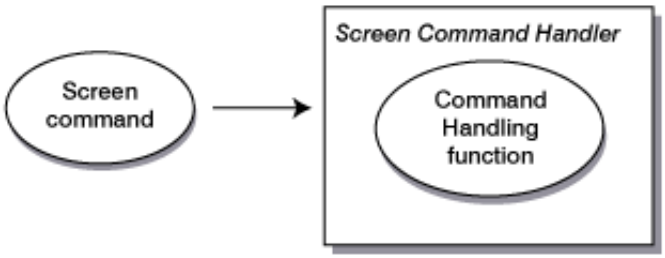
Event handling in J2ME
Event handling in J2ME, in contrast to event handling on the desktop version of the Java platform, is based around a succession of screens. Each screen carries a certain small amount of data.

Commands are presented to the user on a per-screen basis. The `Command` object encapsulates the name and information related to the semantics of an action. It is primarily used for presenting a choice of actions to the user. The resulting command behavior is defined in a `CommandListener` associated with the screen.

Each `Command` contains three pieces of information: a *label*, a *type*, and a *priority*. The label is used for the visual representation of the command; the type and priority are used by the system to determine how the `Command` is mapped onto a concrete user interface.

Figure 5 shows the event handling mechanism in J2ME.

Figure 5. Handling user events in J2ME



Designing the user interfaces
Although it maintains a constrained profile, the MIDP API provides a complete set of UI elements. The following are some of the most important ones:

- An `Alert` acts as a screen to provide information to the user about an exceptional condition or error.
- A `Choice` implements a selection from a predefined number of choices.
- A `ChoiceGroup` provides a group of related choices.
- A `Form` acts as a container for the other UI elements.
- A `List` provides a list of choices.
- A `StringItem` acts as a display-only string.
- A `TextBox` is a screen that allows the user to enter and edit text.
- A `TextField` allows the user to enter and edit text. Multiple `TextField`s can be placed in a `Form`.
- A `DateField` is an editable component for presenting date and time information. A `DateField` can be placed in a `Form`.
- A `Ticker` acts as a scrollable display of text.

A complete list of UI elements is available in the MID Profile API documentation that accompanies the J2ME Wireless Toolkit (see Resources below for more information).

Managing the device database
The MIDP provides a set of classes and interfaces to organize and manipulate a device's database: `RecordStore`, `RecordComparator`, and `RecordFilter`. A `RecordStore` consists of a collection of records, which remain persistent across multiple invocations of the MIDlet. Comparing records in a `RecordStore` or extracting sets of records from a `RecordStore` is functionality provided by `RecordComparator` and `RecordFilter` interfaces.

Developing J2ME applications
The previous sections have given an overview of J2ME. In this section, we will acquaint ourselves with the practical details of the platform through development of a real-world application on a phone interface.

A sample application: Phone calendar
One of the notable features in J2ME is its date manipulation functionality in a constrained environment. The `DateField` UI Item offered by J2ME is an editable component for presenting calender information (i.e., date and time). In this section, we will develop a J2ME application that displays a scrolling calendar on a cell phone UI, using `DateField` and `Date` functions.

A phone calendar application

```
// Import of API classes
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;

//A first MIDlet with simple text and a few commands.
public class PhoneCalendar extends MIDlet
 implements CommandListener, ItemStateListener {

//The commands
private Command exitCommand;

//The display for this MIDlet
private Display display;

// Display items e.g Form and DateField
Form displayForm;
DateField date;

public PhoneCalendar() {
  display = Display.getDisplay(this);
exitCommand = new Command("Exit", Command.SCREEN, 1);
  date = new DateField("Select to date", DateField.DATE);

}

// Start the MIDlet by creating the Form and
// associating the exit command and listener.
public void startApp() {
displayForm = new Form("Quick Calendar");
  displayForm.append(date);
  displayForm.addCommand(exitCommand);
displayForm.setCommandListener(this);
displayForm.setItemStateListener(this);
display.setCurrent(displayForm);
}

public  void itemStateChanged(Item item)
{
 // Get the values from changed item
}

// Pause is a no-op when there is no  background
// activities or record stores to be closed.
public void pauseApp() { }

// Destroy must cleanup everything not handled
// by the garbage collector.
public void destroyApp (boolean unconditional) { }

// Respond to commands. Here we are only implementing
// the exit command. In the exit command, cleanup and
// notify that the MIDlet has been destroyed.
public void commandAction (
```

```
Command c, Displayable s) {
if (c == exitCommand) {
destroyApp(false);
  notifyDestroyed();
}
}


}
```

The `PhoneCalendar` MIDlet as defined above extends an `ItemListener` and a `CommandListener`. It gives the MIDlet the capability of tracking an item change on the screen and responding to user commands. The user interface initiative by this application begins by defining a display for the phone screen and attaching a `Form` to it. The `Form` acts as a container and can hold a number of user interface items. The `commandAction()` function acts a command handler in J2ME and defines the actions to be taken for a certain command.

Deploying J2ME
You can download an emulator from Sun that allows you to test J2ME applications on your desktop system. If you'd rather shun all that graphical overhead, you can also deploy J2ME on your command line.

Deploying in an emulated environment
Deploying and running a J2ME application in an emulated environment involves the installation and configuration of an emulator. The J2ME Wireless Toolkit provides an emulated environment for development and deployment of Java applications on top of resource-constrained devices. Here's how to get yourself up and running:

1. Install the J2ME Wireless Toolkit (see Resources). The installer program will guide you with necessary instructions. Choose a standalone mode for running these examples. Choose an integrated mode in case you want to integrate it with an IDE.

2. Create a new project through the user interface of KToolbar. Indicate a class name.

3. Place the class name from Step 2 in the `C:\[J2ME Installation directory]\apps\[Project Name]\src` directory.

4. Build the project.

5. Choose `DefaultGrayPhone` as a default device from J2ME Wireless Toolkit -> Default Device Selection.

6. Run the project.

The toolkit also provides an option to package the project into a jar file and a jad file. A double-click on the jad file will deploy the application indicated by the jar file.

Deploying on the command line
A number of command-line options are also available.

1. Create the classfile:

```

C:\J2ME\apps\AppointmentPhoneCalendar>
 javac _ tmpclasses _ootclasspath
C:\J2ME\lib\midpapi.zip -classpath tmpclasses;
classes src\*.java
```

2. Create a manifest file, `manifest.mf`:

```
MIDlet-1: AppointmentPhoneCalendar,
AppointmentPhoneCalendar.png,
AppointmentPhoneCalendar
MIDlet-Name: AppointmentPhone Calendar
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
MicroEdition-Configuration: CLDC-1.0
MicroEdition-Profile: MIDP-1.0
```

3. Create a jar file:

```
C:\J2ME\apps\AppointmentPhoneCalendar>jar cfm .\bin\
AppointmentPhoneCalendar.jar
manifest.mf  -C classes . _ res .
```

4. Create a jad file:

```
MIDlet-1: AppointmentPhoneCalendar,
AppointmentPhoneCalendar.png,
AppointmentPhoneCalendar
MIDlet-Jar-Size: 4490
MIDlet-Jar-URL:
F:\J2ME\apps\AppointmentPhoneCalendar\bin\
AppointmentPhoneCalendar.jar
MIDlet-Name: AppointmentPhoneCalendar
MIDlet-Vendor: Sun Microsystems
MIDlet-Version: 1.0
```

5. Run the jad file:

```
C:\J2ME\bin> emulator -Xdescriptor:
C:\J2ME\apps\AppointmentPhoneCalendar
\bin\AppointmentPhoneCalendar.jad
```

The bottom line

J2ME is a significant wireless move from existing software models to the portable, network-centric Java Virtual Machine. The flexibility in development and deployment of J2ME applications will efficiently cater to the increasing requirements of the wireless world. Stay connected!

Resources

- Participate in the discussion forum on this article by clicking **Discuss** at the top or bottom of the article.
- Stay connected to the wireless world with the developerWorks Wireless zone.
- The developerWorks Java technology zone keeps you up to date on events in the Java arena.
- This article discusses the mechanism for relating peer-to-peer and enabling all wireless points to get along.
- Check out the latest developments at IBM's Pervasive Computing site.
- Build Java apps with IBM's Visual Age for Java.

Some documentation from Sun:

- Visit the homepage of J2ME to learn what J2ME is all about.

- Read details of the J2ME architecture and configurations (in PDF format).
- See more documentation on J2ME.
- Read about the Connected Limited Device Configuration and Connected Device Configuration platforms for J2ME.
- Visit the MIDP homepage to learn about the MIDP APIs.
- Download the J2ME Wireless Toolkit to run and test your J2ME applications. You can also browse through the MIDP API documentation that comes with the toolkit.
- Browse through a range of articles on the Java Wireless initiative.

About the author

A graduate in computer science and engineering, Soma Ghosh has developed a wide range of e-commerce and networking Java applications over the past six years. She believes that wireless commerce represents the near future of the industry, and has recently been drawn to wireless initiatives and away from existing desktop models. She is currently a senior application developer with Entigo, a pioneer in B2B sell- and service-side e-commerce products and solutions. Contact Soma at sghosh@entigo.com.