Editor: Sumi Helal ■ University of Florida ■ helal@cise.ufl.edu

Pervasive Java

Sumi Helal, University of Florida

EDITOR'S INTRO

Welcome to the first installment of Standards, Tools, & Best Practices. This department will report on the latest in standards activities and development tools as they pertain to pervasive computing. The intended audience includes practitioners, technologists, and systems researchers who want to base their prototyping activities on industry standards and tools. I plan to discuss several exciting topics including

- platform standardization
- mobile services and their discovery protocols
- sensor networks
- wearable computers and their applications
- peer-to-peer programming over ad hoc networks
- standards and applications for smart homes
- commercial mobility-support middleware (data synchronizers, Web, databases, email, and so forth)
- mobile commerce

I welcome suggestions and contributions for additional topics.

In this first issue, I highlight *pervasive Java*—a surge of wireless and mobile platform standardization activities led by Sun Microsystems. Driven by a fast-pace proliferation of portable devices and appliances with embedded computers, pervasive Java (also called *wireless Java* and *mobile Java*) attempts to bridge the gap between disparate devices and platforms, and profitable business and application development. *—Sumi Helal*

Today, mobile devices and gadgets come with more than just a cool and cutting-edge design; they come equipped with small brains and a common language, making them smarter and more powerful than ever before. Perhaps they're a little too smart—after all, they can adapt and morph into each other and become *many-in-one* super devices. Mobile phones with PDA capabilities and PDAs packaged as mobile phones have started to invade the marketplace. Furthermore, it won't be long before we can shoot a presentation right off our mobile phones or use a mobile phone as a wireless mouse to more easily surf the Web on a laptop. Appliances are also gaining amazing intelligence, thanks to embedded computers and tiny communication interfaces. Point-of-sale equipment, gas pumps, automobile dashboards, and digital cameras are examples of today's smart appliances. Refrigerators with Web pads, microwaves that download recipes, and stereo systems that mute when the phone rings are among the many smart appliances on the horizon.

The impressive leap in this technology led to a rapid proliferation of many kinds of portable and embedded devices. This proliferation signaled the beginning of a new and exciting era of computing. It also quickly underpinned the new requirements we must meet to mobilize this technology. The most critical requirement was the need for a common yet flexible computing and communication environment that could be fitted for—and shared by—devices of different makes and capabilities.

Realizing this need, and recognizing that one size does not fit all, Sun Microsystems introduced the Java 2 Platform, Micro Edition, a set of specifications that pertain to Java on small devices, ranging from pagers and mobile phones to set-top boxes and car navigation systems. Since its introduction, J2ME has evolved according to the Java Community Process, a formalized process and an open organization of international Java developers and licensees whose charter is to develop and revise Java technology specifications, reference implementations, and technology compatibility kits.

The J2ME specifications are divided into configurations that are specific to different device categories. Each configuration is specialized into a set of profiles, which are specifications of partic-

1536-1268/02/$17.00 © 2002 IEEE

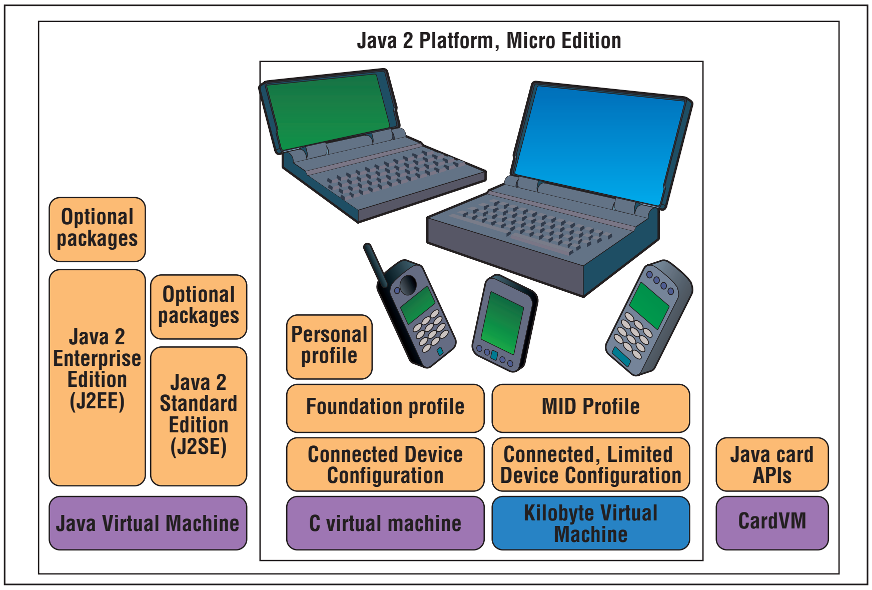

Java 2 Platform, Micro Edition
Optional packages
Java 2 Enterprise Edition (J2EE)
Optional packages
Java 2 Standard Edition (J2SE)
Java Virtual Machine
Personal profile
Foundation profile
MID Profile
Connected Device Configuration
Connected, Limited Device Configuration
C virtual machine
Kilobyte Virtual Machine
Java card APIs
CardVM

Figure 1. One Java, three editions.

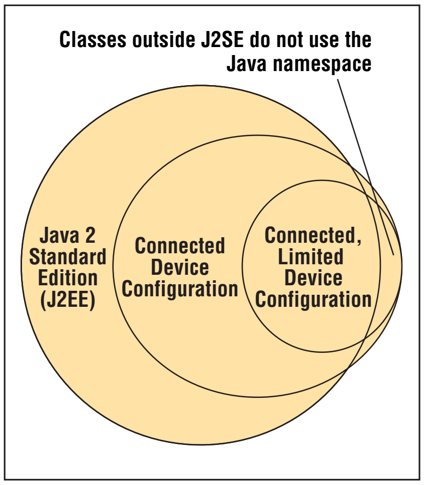

Classes outside J2SE do not use the Java namespace
Java 2 Standard Edition (J2EE)
Connected Device Configuration
Connected, Limited Device Configuration

Figure 2. The relationship between Java 2 Platform, Micro Edition configurations and the Java 2 Standard Edition.

ular types of devices within a device category. Here, I introduce J2ME and its configurations and profiles, emphasizing mobile phones and PDAs that use the Mobile Information Device Profile. In the next issue, I'll survey available J2ME development toolkits and MIDP devices, including mobile phones and PDAs.

JAVA 2 PLATFORM, MICRO EDITION

In June 1999, Sun Microsystems introduced J2ME, targeted for consumer electronics, portables, and embedded devices. This was part of a reorganization effort of the Java technology into Enterprise, Standard, and Micro editions (see Figure 1). To support the kind of flexibility and customizable deployment that the portables and embedded marketplace demand, the J2ME architecture is composed of three modular and scalable layers: *Java Virtual Machine*, *Configurations*, and *Profiles*.

The JVM layer implements a JVM customized for a particular device's host operating system and that supports a particular J2ME configuration.

The configuration layer defines a minimum set of JVM features and core Java class libraries available on a particular category of devices. These devices represent a particular market segment and can be thought of as the lowest common denominator of the Java platform features that a developer can assume will be available on all devices.

The profile layer defines the minimum set of application programming interfaces available on a particular group of devices, which are developed on the underlying configuration. Profiles serve two main purposes: device specialization (APIs that capture or exploit particularities of the device interface and capability) and device portability (APIs that behave consistently on any device supporting the profile). Applications written for a particular profile should therefore port to any device that conforms to that profile. Generally, a device can support multiple profiles on which different applications are built.

The configuration that defines small, mobile devices is called the Connected, Limited Device Configuration. Examples of CLDC devices are mobile phones and pagers. These devices will have memory between 160 and 512 Kbytes and use the Kilobyte Virtual Machine (KVM). A CLDC device uses either a 16- or 32-bit processor and a low-bandwidth wireless network connection. The only profile currently developed for the CLDC configuration is the MIDP.

The Connected Device Configuration is considered a fixed type of device that is always connected but still relatively resource poor. An example would be set-top boxes such as a satellite TV receiver or WebTV. The CDC configuration is a superset of the CLDC, which ensures their compatibility. Figure 2 shows the relationship between the various configurations.

Here, I highlight the main features of the KVM–CLDC configuration and the MIDP. (For more information, see the Sun Microsystems white paper, "Java 2 Platform Micro Edition (J2ME) Technology for Creating Mobile Devices," at http://java.sun.com/products/cldc/wp/KVMwp.pdf.)

KVM

The KVM is a new, smaller runtime environment for resource-constrained devices. It is in the range of 40 to 80

GLOSSARY

CDC	Connected Device Configuration
CLDC	Connected, Limited Device Configuration
J2ME	Java 2 Platform, Micro Edition
J2SE	Java 2 Platform, Standard Edition
JAD	Java Application Descriptor
JAM	Java Application Manager
JAR	Java Archive
JVM	Java Virtual Machine
KVM	Kilobyte Virtual Machine
MIDP	Mobile Information Device Profile

Kbytes—hence the "K" in KVM. Developing the KVM originated as a project, known as the Spotless System, to create an execution engine for the Palm PDA. The Spotless System development team soon discovered that the runtime environment's size is derived mainly from its runtime libraries. They then extracted classes that were too bloated or not as critical to the system. The primary features eliminated were

- The Java Native Interface, because native functionality is implementation dependent
- The user-defined class loader, because CLDC will have a class loader that users can't override, replace, or reconfigure
- Reflection, to eliminate RMI, serialization, or any other features reliant on reflection
- Thread groups and daemon threads (although the system supports multithreading)
- Finalization of class instances
- Weak references
- AWT (Java Abstract Window Toolkit), which the developers replaced with limited user interface classes, developed from the javax.microedition.lcdui package
- The floating-point type (although the system enables long integers)

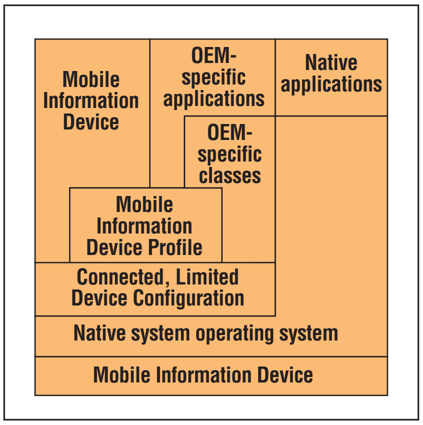

Mobile Information Device
OEM-specific applications
Native applications
OEM-specific classes
Mobile Information Device Profile
Connected, Limited Device Configuration
Native system operating system
Mobile Information Device

Figure 3. Relationships between Kilobyte Virtual Machine, Connected Limited Device Configuration, and Mobile Information Device Profile layers.

A reference implementation of KVM written in the C language is available. Most of the code is common to all existing and future implementations. A relatively small amount of platform-dependent code is isolated in a few files. Porting KVM to a particular device amounts to modifying only these dependency files. A special startup mechanism of KVM and its applications is included in this reference implementation. This is needed because, unlike a general-purpose machine, small devices do not have a command-line or special interface to start software atop the native operating system. The Java Application Manager serves as an interface between the native operating system and KVM. JAM downloads and launches a J2ME application on top of KVM by connecting to an HTTP server (using a serial line interface or some other means, such as "over the air" in the case of mobile phones) to download the application (called *MIDlet*) and a metadata file that describes it. JAM uses the metadata file to uncompress and install the MIDlet.

CLDC

The CLDC configuration defines a standard Java platform for small, resource-constrained, connected devices and enables the dynamic delivery of Java applications and content to those devices. The small footprint configuration (limited to about 128 Kbytes) is designed to run on many small devices, making minimal assumptions about the native system software available. CLDC is implemented as a set of additional classes contained in a separate package (the java.io, java.lang, java.util, and javax.nicroedition.io packages). This facilitates CLDC porting to different platforms.

CLDC requires that a JVM be able to identify and reject invalid class files. The standard verification technique, such as that used by J2SE (Standard Edition), is expensive in terms of memory use. To reduce client-side verification overhead, CLDC uses a dual pre-verification and verification process to push part of the verification process into the development platform. The developer must pre-verify the server or desktop prior to downloading the CLDC MIDlet application into the mobile device. The pre-verification process generates information (known as *stackmap* attributes) that should be downloaded to the mobile device along with the application classes. To streamline the packaging of this pre-verification information and to enable dynamic downloading of third-party applications and content, CLDC requires implementations to support the distribution of Java applications using compressed Java Archive (JAR) files. Whenever a Java application intended for a CLDC device is created, it must be formatted into a JAR file, and class files within a JAR file must contain the stackmap attributes. The MIDP also imposes a similar data requirement on third-party MIDlets.

MIDP

The MIDP is a set of Java APIs that, together with the CLDC, provides a complete J2ME application runtime environment targeted at mobile information devices, such as mobile phones, PDAs, and two-way pagers. It is the only profile currently fully specified for CLDC (another profile that specializes only on PDAs—the PDA profile is currently being developed). The MDIP provides API classes related to interface, persistence storage, networking, and application model. For instance, the MIDP specifications provide an API for persistent storage in the form of a *RecordStore*. This is a Java object that can be instantiated and used to store data objects as raw bytes. The profile also provides a set of requirements for device manufactures to use as guidelines if they want their devices to be CLDC and MIDP branded. For example, the MIDP minimum display requirements are a 96 × 54 pixel screen-size, a display depth of 1 bit, and a pixel shape (aspect ratio) of approximately 1:1. Another MIDP minimum requirement is memory, which is 128 Kbytes of RAM, 8 Kbytes of application-created data, and 32 Kbytes of Java heap.

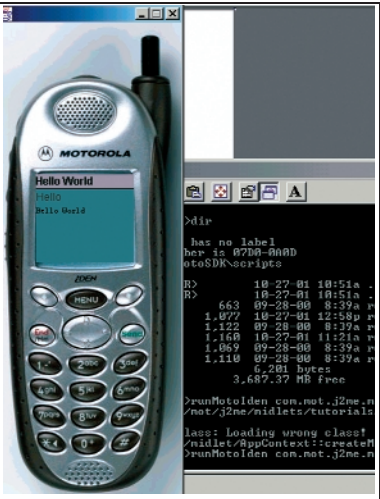
To enable the distribution of third-party MIDlets, developers must generate a metadata file. Once they package an application (classes and all auxiliary components) into a JAR file, the developers must generate an associated Java Application Descriptor (JAD) and ship it along with the JAR file. Toolkits exist to aid developers with the packaging and to help them generate the JAR and JAD files. JAD contains information that the JAM will use to properly verify and configure the MIDlet application at loading time and that the MIDlet will use during execution time.

MIDP has been created through the Java Community Process (see the Java Specification Request JSR37 at http://jcp.org/jsr/detail/37.jsp). The goal of the consortium that created the MIDP (which consisted mainly of device manufacturers) was to create an open third-party application development environment for mobile information devices. Device manufacturers may include additional original equipment manufacturer classes (such as APIs optimized for packet data communication, gaming, or animated images). MIDlet applications created or preinstalled on the device by the manufacturers may use these additional classes and APIs. Third-party developers, however, may not access the native device operating system, or any OEM classes or applications. Figure 3 shows the architectural view of the relationships between the KVM, CLDC, and MIDP layers.

Hello World MIDlet

Figure 4 shows a Hello World MIDlet. The MIDlet contains the constructor method as well as the methods startApp(), pauseApp(), and destroyApp(). It calls the first method when the application starts or restarts after a pause state. It calls the pauseApp() during the phone's idle or paused mode and calls destroyApp() right before it is unloaded. The JAM will create the Hello World MIDlet, and then the public, no-argument constructor is called. The JAM will then call theMIDlet's startApp() method, which loads the display to the instance of the Form class mainscreen. This instance creates the Form used to hold the StringItem instance, strItem, properly initialized to display "Hello World." By appending strings and many other components to the form, the information is displayed into the Java phone.

```
/*
 *  HelloWorld.java
 */

package com.mot.j2me.midlets.tutorials;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

/*
 * A simple Hello World midlet
 */
public class HelloWorld extends MIDlet {

    private Form mainscreen;
    private Display myDisplay;

    HelloWorld1(){
       myDisplay         = Display.getDisplay(this);
       mainscreen         = new Form("Hello World");
       StringItem strItem = new StringItem("Hello", "Hello
World");
       mainscreen.append(strItem);
    }

    public void startApp() throws
MIDletStateChangeException{
         myDisplay.setCurrent(mainscreen);
    }
    protected void pauseApp() {
    }
    protected void destroyApp(boolean unconditional)
            throws MIDletStateChangeException {
    }
}
```



```
>dir

 has no label
her is 07D0-0A0D
otoSDK\scripts

R>           10-27-01  10:51a .
R>           10-27-01  10:51a .
      663    09-28-00   8:39a r
    1,077    10-27-01  12:58p r
    1,122    09-28-00   8:39a r
    1,160    10-27-01  11:21a r
    1,069    09-28-00   8:39a r
    1,110    09-28-00   8:39a r
             6,201 bytes
          3,687.37 MB free

>runMotoIden com.mot.j2me.m
/mot/j2me/midlets/tutorials

lass: Loading wrong class!
/midlet/AppContext::createM
>runMotoIden com.mot.j2me.m
```

Figure 4. The Hello World MIDlet.

Pervasive Java is a significant technology development that is truly enabling mobile and pervasive computing. It is transforming mobile devices and appliances from just "cool" gadgets into essential players and integrated elements in the pervasive computing world. By making mobile devices and appliances smarter and by training them to speak the same language (allowing baby devices to speak weaker dialects of the language), we've come far closer to what a few pioneers in this field envisioned years ago. The stage is now set to realize this vision, and the opportunities are numerous to elevate and push the frontiers of research and development, in pursuit of a better, more effective use of computers in our lives.

Sumi Helal is an associate professor in the Computer and Information Science and Enginering Department at the University of Florida. Contact him at helal@cise.ufl.edu; www.cise.ufl.edu/~helal.