CHAPTER 2

Overview of Java 2 Platform, Micro Edition (J2ME™)

2.1 Java 2 Platform

Recognizing that one size does not fit all, Sun Microsystems has grouped Java technologies into three editions, each aimed at a specific area of today's vast computing industry:

- *Java 2 Platform, Enterprise Edition (J2EE™)* for enterprises needing to serve their customers, suppliers and employees with scalable server solutions.
- *Java 2 Platform, Standard Edition (J2SE™)* for the familiar and well-established desktop computer market.
- *Java 2 Platform, Micro Edition (J2ME™)* for the combined needs of:
  - consumer and embedded device manufacturers who build a diversity of information devices,
  - service providers who wish to deliver content to their customers over those devices,
  - content creators who want to make compelling content for small, resource-constrained devices.

Each Java platform edition defines a set of technologies that can be used with a particular product:

- Java virtual machines that fit inside a wide range of computing devices,
- libraries and APIs specialized for each kind of computing device,
- tools for deployment and device configuration.

Figure 2.1 illustrates the Java 2 Platform editions and their target markets, starting from the high-end platforms on the left and moving towards low-end platforms on the right. Basically, five target markets or broad device categories are identified. Servers and enterprise computers are supported by Java 2 Enterprise Edition, and desktop and personal computers by Java 2 Standard Edition. Java 2 Micro Edition is divided broadly into two categories that focus on high-end and low-end consumer devices. Java 2 Micro Edition is discussed in more detail later in this chapter. Finally, the Java Card™ standard focuses on the smart card market.

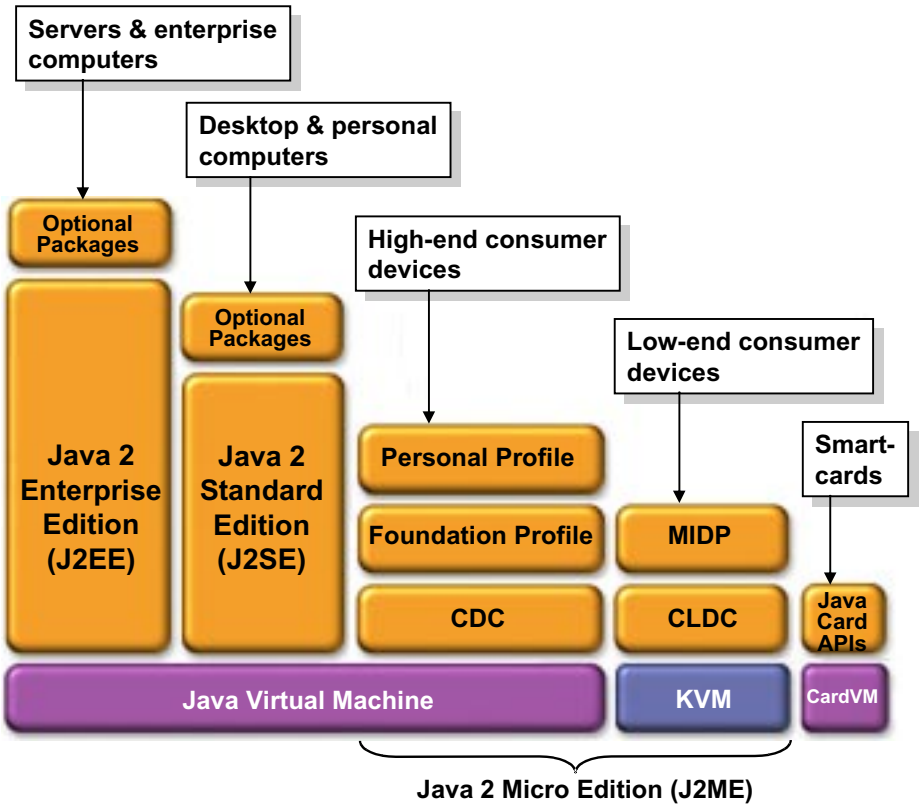

Figure 2.1 Java 2 Platform editions and their target markets

2.2 Java 2 Platform, Micro Edition (J2ME)

Java 2 Platform, Micro Edition (henceforth referred to as *Java 2 Micro Edition* or J2ME) specifically addresses the large, rapidly growing consumer space, which covers a range of devices from tiny commodities, such as pagers, all the way up to the TV set-top box, an appliance almost as powerful as a desktop computer. Like the

larger Java editions, Java 2 Micro Edition aims to maintain the qualities that Java technology has become known for, including built-in consistency across products, portability of code, safe network delivery and upward scalability.

The high-level idea behind J2ME is to provide comprehensive application development platforms for creating dynamically extensible, networked devices and applications for the consumer and embedded market. J2ME enables device manufacturers, service providers and content creators to capitalize on new market opportunities by developing and deploying compelling new applications and services to their customers worldwide. Furthermore, J2ME allows device manufacturers to open up their devices for widespread third-party application development and dynamically downloaded content, without losing the security or the control of the underlying manufacturer-specific platform.

At a high level, J2ME is targeted at two broad categories of products:

- *High-end consumer devices*. In Figure 2.1, this category is represented by the grouping labeled *CDC* (Connected Device Configuration). Typical examples of devices in this category include TV set-top boxes, Internet TVs, Internet-enabled screenphones, high-end wireless communicators and automobile entertainment/navigation systems. These devices have a large range of user interface capabilities, total memory budgets starting from about two to four megabytes and persistent, high-bandwidth network connections, often using TCP/IP.
- *Low-end consumer devices*. In Figure 2.1, this category is represented by the grouping labeled *CLDC* (Connected, Limited Device Configuration). Cell phones, pagers, and personal organizers are examples of devices in this category. These devices have very simple user interfaces (compared to desktop computer systems), minimum memory budgets starting at about 128 kilobytes, and low bandwidth, intermittent network connections. In this category of products, network communication is often not based on the TCP/IP protocol suite. Most of these devices are usually battery-operated.

The line between these two categories is fuzzy and becoming more so every day. As a result of the ongoing technological convergence in the computer, telecommunication, consumer electronics, and entertainment industries, there will be less distinction between general-purpose computers, personal communication devices, consumer electronics devices, and entertainment devices. Also, future devices are more likely to use wireless connectivity instead of traditional fixed or wired networks. In practice, the line between the two categories is defined more by the memory budget, bandwidth considerations, battery power consumption and

physical screen size of the device, rather than by its specific functionality or type of connectivity.

Because of strict manufacturing cost constraints, the majority of high-volume wireless devices today such as cell phones belong to the low-end consumer device category. Therefore, this book focuses only on the CLDC and MIDP standards that were specifically designed for that category of products.

2.3 Key Concepts of the J2ME Architecture

While connected consumer devices such as cell phones, pagers, personal organizers and TV set-top boxes have many things in common, they are also extremely diverse in form, function, and features. Information appliances tend to be special-purpose, limited-function devices. To address this diversity, an essential requirement for the J2ME architecture is not only small size but also modularity and customizability.

In general, serving the information appliance market calls for a large measure of flexibility in how computing technology and applications are deployed. This flexibility is required because of

- the large range of existing device types and hardware configurations,
- the different usage models employed by the devices (key operated, stylus operated, voice operated),
- constantly improving device technology,
- the diverse range of existing applications and features,
- the need for applications and capabilities to change and grow, often in unforeseen ways, in order to accommodate the future needs of the consumer.

The J2ME architecture is intended to be modular and scalable so that it can support the kinds of flexible deployment demanded by the consumer and embedded markets. To enable this, the J2ME environment provides a range of Java virtual machine technologies, each optimized for the different processor types and memory footprints commonly found in the consumer and embedded marketplace.

For low-end, resource-limited consumer products, the J2ME environment supports minimal configurations of the Java virtual machine and Java libraries that embody just the essential capabilities of each kind of device. As device manufacturers develop new features in their devices, or service providers develop new and exciting applications, these minimal configurations can be expanded with additional libraries that address the needs of a particular market segment. To support

this kind of customizability and extensibility, two essential concepts are defined by the J2ME environment:

- *Configuration*. A J2ME configuration defines a minimum platform for a "*horizontal*" category or grouping of devices, each with similar requirements on total memory budget and processing power. A configuration defines the Java language and virtual machine features and minimum class libraries that a device manufacturer or a content provider can expect to be available on all devices of the same category.
- *Profile*. A J2ME profile is layered on top of (and thus extends) a configuration. A profile addresses the specific demands of a certain "*vertical*" market segment or device family. The main goal of a profile is to guarantee interoperability within a certain vertical device family or domain by defining a standard Java platform for that market. Profiles typically include class libraries that are far more domain-specific than the class libraries provided in a configuration. One device can support multiple configurations.

Configurations and profiles are discussed in more detail below. Both configurations and profiles use the capabilities of the Java virtual machine (JVM), which is considered to be part of the configuration. The virtual machine usually runs on top of a *host operating system* that is part of the system software of the target device. The high-level relationship between the different software layers—the JVM, configuration, profiles and the host operating system—is illustrated in Figure 2.2.

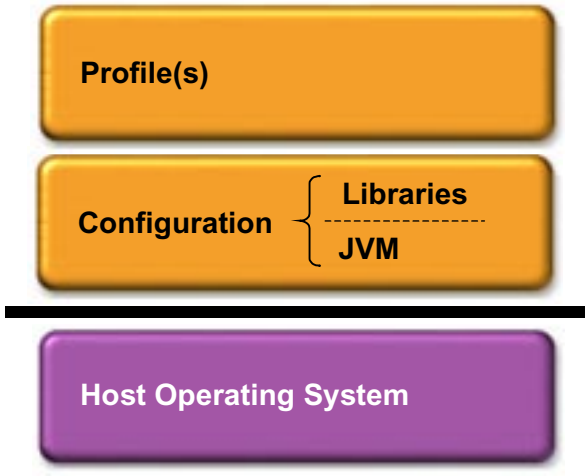

Figure 2.2 Software layers in a J2ME device

J2ME configurations and profiles are defined through the Java Community Process (JCP). For further information on the Java Community Process, refer to the Java Community Process web site.[1]

2.3.1 Profiles

Application portability is a key benefit of Java technology in the desktop and enterprise server markets. Portability is an equally critical element in the consumer device space. However, application portability requirements in the consumer space are very different from portability requirements demanded by the desktop and server markets. In most cases consumer devices differ substantially in memory size, networking and user interface capabilities, making it very difficult to support all devices with just one solution.

In general, the consumer device market is not so homogeneous that end users can expect or require universal application portability. Rather, in the consumer space, applications should ideally be fully portable within the same device family. For example, consider the following types of consumer devices:

- cellular telephones,
- washing machines,
- electronic toys.

It seems clear that each of these represents a different market segment, device family or application domain. As such, consumers would expect useful applications to be portable within a device family. For example:

- A discount broker's stock trading application is generally expected to work on different cell phones, even though the phones are from different manufacturers.
- It would be annoying if a highly useful grape-juice-stain-removing wash cycle application available on the Internet runs on an old brand-X washer, but not a new brand-Z washer.
- A child's birthday party could be less enjoyable if the new toy robot doesn't "talk to" or "play games with" the new electronic teddy bear.

On the other hand, consumers do not expect the stock trading application or an automobile service program to run on the washing machine or the toy robot. In

---

1. `http://java.sun.com/aboutJava/communityprocess`

other words, application portability *across* different device categories is not necessarily very important or even meaningful in the consumer device space.

In addition, there are important economic reasons to keep these device families separate. Consumer devices compete heavily on cost and convenience, and these factors often translate directly into limitations on physical size and weight, processor power, memory size and power consumption (in battery-powered devices). Consumers' wallets will usually favor devices that perform the desired functions well, but that do not have added cost for unnecessary features.

Thus, the J2ME framework provides the concept of a *profile* to make it possible to define Java platforms for specific vertical markets. A profile defines a Java platform for a specific vertical market segment or device category. Profiles can serve two distinct portability requirements:

- A profile provides a complete toolkit for implementing applications for a particular kind of device, such as a pager, set-top box, cell phone, washing machine, or interactive electronic toy.
- A profile may also be created to support a significant, coherent group of applications that might be hosted on several categories of devices. For example, while the differences between set-top boxes, pagers, cell phones and washing machines are significant enough to justify creating a separate profile for each, it might be useful for certain kinds of personal information management or home banking applications to be portable to each of these devices. This could be accomplished by creating a separate profile for these kinds of applications and ensuring that this new profile can be easily and effectively supported on each of the target devices along with its "normal" more device-specific profile.

It is possible for a single device to support several profiles. Some of these profiles are very device-specific, while others are more application-specific. Applications are written "for" a specific profile and are required to use only the features defined by that profile. Manufacturers choose which profile(s) to support on each of their devices, but are required to implement all features of the chosen profile(s). The value proposition to the consumer is that any application written for a particular profile will run on any device that supports that profile.

In its simplest terms, a profile is a contract between an application and a vertical market segment. All the devices in the same market segment agree to implement all the features defined in the profile. And the application agrees to use only those features defined in the profile. Thus, portability is achieved between the applications and the devices served by that profile. New devices can take advantage of a large and familiar application base. Most importantly, new compelling

applications (perhaps completely unforeseen by the original profile designers and device manufacturers) can be dynamically downloaded to existing devices.

At the implementation level, a profile is defined simply as a collection of class libraries that reside on top of a specified configuration and that provide the additional domain-specific capabilities for devices in a specific market segment.

In our example above, each of the three families of devices (cell phones, washing machines and intercommunicating toys) would be addressed by a separate J2ME profile. The only one of these profiles in existence at the current time is the MIDP, designed for cell phones and other two-way communication devices.

2.3.2 Configurations

In the J2ME environment, an application is written "for" a particular profile, and a profile is "based upon" or "extends" a particular configuration. Thus, all of the features of a configuration are automatically included in the profile and may be used by applications written for that profile.

A configuration defines a Java platform for a "horizontal" category or grouping of devices with similar requirements on total memory budget and other hardware capabilities. More specifically, a configuration:

- specifies the Java programming language features supported,
- specifies the Java virtual machine features supported,
- specifies the basic Java libraries and APIs supported.

The J2ME environment is designed so that it can be deployed in more than one configuration. Each configuration specifies the Java language and virtual machine features and a set of libraries that the profile implementer (and the applications using that profile) can safely assume to be present on all devices when shipped from the factory. Profile implementers must design their code to stay within the bounds of the features and libraries specified by that configuration.

In its simplest terms, a configuration defines a "lowest common denominator" platform or building block for device manufacturers and profile implementers. All the devices with approximately the same amount of memory and processing power agree to implement all the features defined in the configuration. And the profile implementers agree to use only those features defined in the configuration. Thus, portability is achieved between the profile and the devices served by that configuration.

In our example above, each of the three profiles (for cell phones, washing machines and electronic toys) would most likely be built upon the same configura-

tion, the CLDC. This configuration provides all the basic functionality to serve the needs of each of these, and perhaps many more, profiles.

To avoid fragmentation, there are a very limited number of J2ME configurations. Currently, (see Figure 2.1) only two standard J2ME configurations are available:

- *Connected, Limited Device Configuration (CLDC)*. This configuration focuses on *low-end consumer devices*. Typical examples of CLDC target devices include personal, mobile, battery-operated, connected information devices such as cell phones, two-way pagers, and personal organizers. This configuration includes some new classes, not drawn from the J2SE APIs, designed specifically to fit the needs of small-footprint devices.
- *Connected Device Configuration (CDC)*. This configuration focuses on *high-end consumer devices*. Typical examples of CDC target devices include shared, connected information devices such as TV set-top boxes, Internet TVs, and high-end communicators. This configuration includes a much more comprehensive set of Java libraries and virtual machine features than CLDC.

Figure 2.3 illustrates the relationship between CLDC, CDC and Java 2 Standard Edition (J2SE). As shown in the figure, the majority of functionality in CLDC and CDC has been inherited from Java 2 Platform, Standard Edition (J2SE). Each class inherited from the J2SE environment must be precisely the same or a subset of the corresponding class in the J2SE environment. In addition, CLDC and CDC may introduce a number of features, not drawn from the J2SE, designed specifically to fit the needs of small-footprint devices.

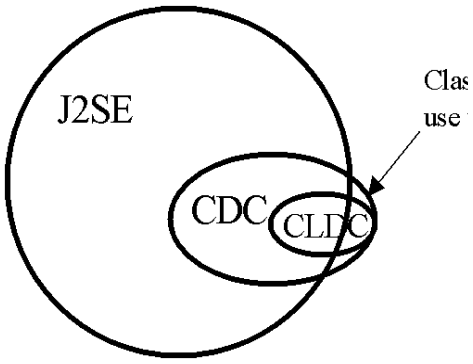

Figure 2.3 Relationship between J2ME configurations and Java 2 Standard Edition

The most important reason for the configuration layer in the J2ME environment is that core Java libraries needed across a wide variety of Java platform implementations are usually intimately tied with the implementation of a Java virtual machine. Small differences in the specification of a configuration can require a number of significant modifications to the internal design of a Java virtual machine, and can require a substantial amount of additional memory footprint. Such modifications would be very expensive and time-consuming to maintain. Having a small number of configurations means that a small number of virtual machine implementations can serve the needs of both a large number of profiles and a large number of different device hardware types. This economy of scale provided by the J2ME environment is very important to the success and cost-effectiveness of devices in the consumer and embedded industry.

2.4 The K Virtual Machine (KVM)

CLDC and MIDP commonly run on top of Sun's K Virtual Machine (KVM). KVM is a compact, portable Java virtual machine specifically designed for small, resource-constrained devices. The high-level design goal for KVM was to create a new Java virtual machine with the following characteristics:

- small, with a static memory footprint of the core of the virtual machine starting from about 60 kilobytes, depending on compilation options and the target platform,
- clean, well-commented and highly portable,
- modular and customizable,
- as complete and fast as possible without sacrificing the other design goals.

The "K" in KVM stands for "kilo." It was so named because its memory budget is measured in tens of kilobytes (whereas desktop systems are measured in megabytes or even gigabytes). KVM is suitable for 16/32-bit microprocessors with a total memory budget of no more than a few hundred kilobytes. This typically applies to digital cellular phones, pagers, personal organizers, portable audio/video devices and small retail payment terminals.

The minimum total memory budget required by a KVM implementation is about 128 kilobytes, including the virtual machine, minimal libraries and some heap space for running Java applications. A more typical implementation requires a total memory budget of 256 kilobytes, of which at least 32 kilobytes is used as runtime heap space for applications, 60 to 80 kilobytes is needed for the virtual

machine itself, and the rest is reserved for class libraries. The ratio between volatile memory (*e.g.*, DRAM) and non-volatile memory (*e.g.*, ROM or Flash memory) in the total memory budget varies considerably depending on the implementation, the device, the configuration, and the supported profiles. A simple KVM implementation without system class preloading support needs more volatile memory than a KVM implementation with system classes (or even applications) preloaded into the device.

The actual role of KVM in the target devices can vary significantly. In some implementations, KVM is used on top of an existing native software stack to give the device the ability to download and run dynamic, interactive, secure Java content on the device. In other implementations, KVM is used at a lower level to also implement the lower-level system software and applications of the device in the Java programming language.

For further information on KVM, refer to the KVM product web site (`http://java.sun.com/products/kvm`).