chapter 2

Introducing Nokia Developer Platforms

The Nokia Developer Platforms allow developers to write scalable applications across a range of Nokia devices.

The mobile handset industry has seen fast-paced innovation in the last several years. Nokia alone has been announcing more than a dozen new devices every year. That is great news for consumers, since Nokia offers choices. But for mobile application developers, it is tough to make sure that applications work correctly on all handsets. The Nokia Developer Platforms aim to solve this problem by standardizing developer APIs among Nokia phones. Each Developer Platform supports a standard set of technologies on a series of Nokia devices. In 2004, more than 100 million Developer Platform devices will be sold worldwide.

Key technologies supported on Nokia Developer Platforms are open industry standards. In particular, Java technology plays a crucial role. Client-side and server-side Java technologies can be used to develop applications for all Developer Platform devices. That helps 3 million existing Java developers to enter this exciting new market. In this chapter, we discuss the big pictures and architectures behind the Nokia Developer Platforms as well as the technical specifications of the most popular Series 40 and 60 Developer Platforms. From a Java developer's perspective, we cover the four technology pillars on the Series 40 and 60 Developer Platforms: Wireless Markup Language (WML), and Extensible Hypertext Markup Language (XHTML) browsers, Multimedia Message Services (MMS), Java 2 Micro Edition (J2ME), and Symbian C++. Strengths and weakness of each technology are addressed. Key topics in this chapter include

- **Open Standard Mobile Technologies:** explains the synergy between open standards and mobile technologies.
- **Nokia Developer Platform Architecture:** covers the basic architecture, device characteristics, and supported technologies on each Nokia Developer Platform.
- **Pervasive Client Technologies:** discusses the thin-client application paradigm using browser and MMS technologies. They are available on all Nokia Developer Platforms.
- **Managed Smart-Client Technology:** introduces Java technology for smart-client development on Series 40, 60, 80, and 90 devices.

- **Tightly Integrated Smart-Client Technology**: introduces the Symbian C++ technology for native smart-client applications on Series 60, 80 and 90 devices.
- **Get Connected**: gives a brief overview of services from Forum Nokia that help developers, operators, and business leaders to take advantage of the Nokia Developer Platforms.

In this chapter, we cover the technologies from a bird's-eye view. Development tools, API tutorials, design patterns, and best practices are covered in later chapters. In a sense, the rest of this book is to elaborate the concepts discussed in this chapter and put them into practical terms through real-world code examples.

Open Standard Mobile Technologies

Mobile commerce and mobile entertainment present users and developers with tremendous opportunities. But in order to realize those promises, the enabling technologies must keep up with the customer demands. We need continued innovations in both device hardware and software. The successes of the PC and the Internet industries have taught us that standardization and open platforms are the keys to sustainable innovations. In the mobile space, open standards are crucial to both device manufacturers and software developers.

- For developers, standards-based technologies lower the barrier of entry for development and reduce the time and effort required to learn new proprietary APIs and tools. Developers can easily optimize standard-technology-based applications for several different devices.
- For device manufacturers, standards-based technologies allow them to reach out to developer communities. A large portfolio of innovative third-party applications is crucial to the market success of any new device.

However, traditionally, mobile device manufacturers have been slow to embrace open standards. Closed platforms are often considered more secure and more efficient for small consumer devices. Proprietary solutions are developed to take advantage of special hardware optimizations. But that practice has hindered the independent developer's ability to write applications for these smart devices. As the computing power of mobile devices increases exponentially according to Moore's law, a smart phone today can easily have more processing power and memory than a 10-year-old desktop PC. The need for innovative software outweighs the benefits of proprietary optimizations. Today, all major mobile device manufacturers have their own open standards strategies. Nokia is leading the way with the Nokia Developer Platforms.

The Nokia Developer Platforms allow developers to write applications for almost all Nokia devices using open standard technologies. Such platform-enabling technologies include the following.

- Java 2 Micro Edition (J2ME) is a smart-client platform developed by the Java Community Process (JCP), which includes Nokia and all other major wireless handset vendors. J2ME specifications define the programming language, the virtual machine, and programming APIs. It is available on all Nokia Developer Platform devices.
- WML and XHTML are markup languages for authoring Web pages. They are standardized by the World Wide Web Consortium (W3C). Dynamic Web pages can be served by Java-enabled application servers via the HTTP network. All Nokia Developer Platform phones have WML or XHTML browsers.
- MMS is the standard way to deliver multimedia content asynchronously to mobile devices. The Third-Generation Partnership Project (3GPP) defines an open XML/SOAP API (MM7) to access MMS service-center servers in wireless carrier networks. We can use Java Web services toolkits to send and receive MMS messages to handsets from the desktop or server computers. All Nokia Developer Platforms support sending and receiving MMS messages.
- Digital Rights Management (DRM) enables content publishers to provision copyrighted material with special metadata that prevents the receiving device from copying and forwarding it to third parties. The content is typically downloaded from the HTTP network or via MMS. Nokia's DRM solutions are based on the Open Mobile Alliance (OMA) standard.
- The OMA Client Provisioning solution enables developers and wireless operators to send device configuration settings to supported Nokia Developer Platform phones over the air.
- Symbian OS is an open standard mobile operating system developed by a group of leading mobile handset manufacturers, each owning a stake in Symbian. It is the operating system for all Nokia high-end smart phones and enterprise and mobile media devices. The Symbian C++ native programming API can be used to develop applications for Symbian devices.

The audience of this book is primarily Java developers who are interested in developing end-to-end applications for Nokia devices. Throughout the book, we cover, in detail, the use of both client-side and server-side Java technologies to develop smart-client or server-driven mobile applications. In this chapter, we introduce Nokia Developer Platforms from a Java developer's perspective (Figure 2–1).

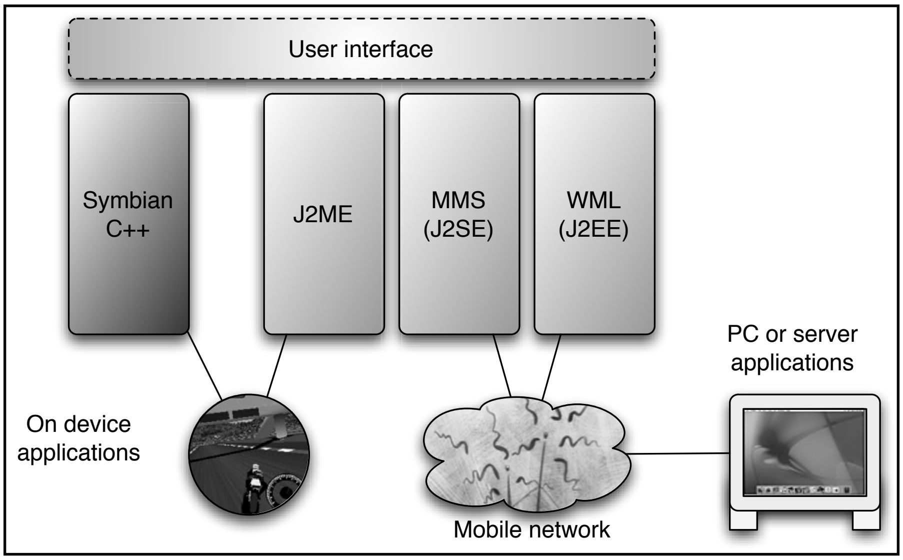


Figure 2–1 Nokia Developer Platform technologies from a Java developer's point of view.

Nokia Developer Platform Architecture

Standardized and open technologies enable developers to develop and optimize portable applications across different devices using the same APIs and tools. However, mobile devices are often used in specific application areas with very different requirements. Not all technologies are available on all devices. As a result, a monolithic Developer Platform does not work. Nokia divides its developer platform into several series, each targeting a specific device market segment. Hence, the Developer Platforms are not just about enabler technologies. They are about devices and user experiences as well.

NOTE

Nokia recommends a "develop-and-optimize" approach to building applications. You first write applications for the key technology enabler (such as Symbian/C++ or Java MIDP), which you select for the application's requirements, your expertise/preference, and the desired market.

From here, you can target specific Developer Platform versions and develop an application against the Developer Platform specifications. The key to leveraging the Nokia Developer Platforms and minimizing device-specific development is to remain as abstract as possible for as long as possible when developing mobile applications.

The next step is to optimize applications for the different user interfaces on a given Developer Platform (file size, screen size, key mapping, etc.) and then finally to take any device-specific hardware limitations or issues into account (such as file size limitations or processor speeds).

Currently, Nokia supports four Developer Platforms. The devices covered under each platform are the following.

- **Series 40 Developer Platform** includes mass-market phones with LCD screens and multimedia capabilities. It is the biggest platform in terms of both revenue and number of users.
- **Series 60 Developer Platform** includes smart phones and mobile game decks based on Symbian OS v6, v7, and beyond.
- **Series 80 Developer Platform** includes high-end enterprise devices based on Symbian OS v7 and beyond, with a full stack of enterprise communication software.
- **Series 90 Developer Platform** includes high-end mobile media devices with advanced multimedia (audio and video) features. Those devices are based on Symbian OS v7 and beyond.

Developer Platforms are not static. They have to evolve to keep up with innovations in device technologies. The Series 40 and 60 Developer Platforms 1.0 mainly apply to devices released before 2004; the Developer Platforms 2.0 apply to most devices that came out in and after 2004.

In this book, we focus on the Series 40 Developer Platform 2.0 and cover important aspects of the Series 60, 80, and 90 Developer Platforms. This approach encourages Series 40 developers to design applications compatible with higher series devices and provides a path for Series 40 developers to extend their skills. Now, let's look at the technical specifications of those platforms.

NOTE

Nokia Developer Platforms are independent from the user interface. In fact, one of the major strengths of Developer Platforms is that they allow Nokia to implement multiple UI flavors on a common set of device technologies. However, currently, most devices in a series have very similar UI designs. Throughout this book, when we discuss UI designs for a particular series, we refer to the typical and most popular UI design for devices in this series.

Series 40 Developer Platform

The Series 40 Developer Platform targets mass-market consumer devices with hundreds of millions of users. Series 40 devices are very important to developers due to their large market penetration. On the other hand, they also

02Yuan.qrk 11/18/04 8:14 PM Page 29

present the biggest challenge to developers due to their limited size and resource constraints. In this section, we first look at the enabler technologies that make up this platform. Then we check out the device characteristics and user interfaces of the current Series 40 devices.

Software Stack

The basic technology stack on a Series 40 device is illustrated in Figure 2–2. At the bottom, there are device hardware and Nokia's proprietary operating system (Nokia OS). The Nokia OS is closed to developers outside of Nokia. On top of the Nokia OS, all Series 40 devices support a common set of native client applications:

- Telephony applications such as speed dialing, call logs, and mobile messaging clients.
- Personal information management (PIM) applications, including calendar, to-do lists, and phonebook.
- Synchronization applications that synchronize the PIM database with desktop PCs via the Nokia PC suite.
- Application installation and management utilities, including over-the-air (OTA) download, wallpaper, and ringtone managers.

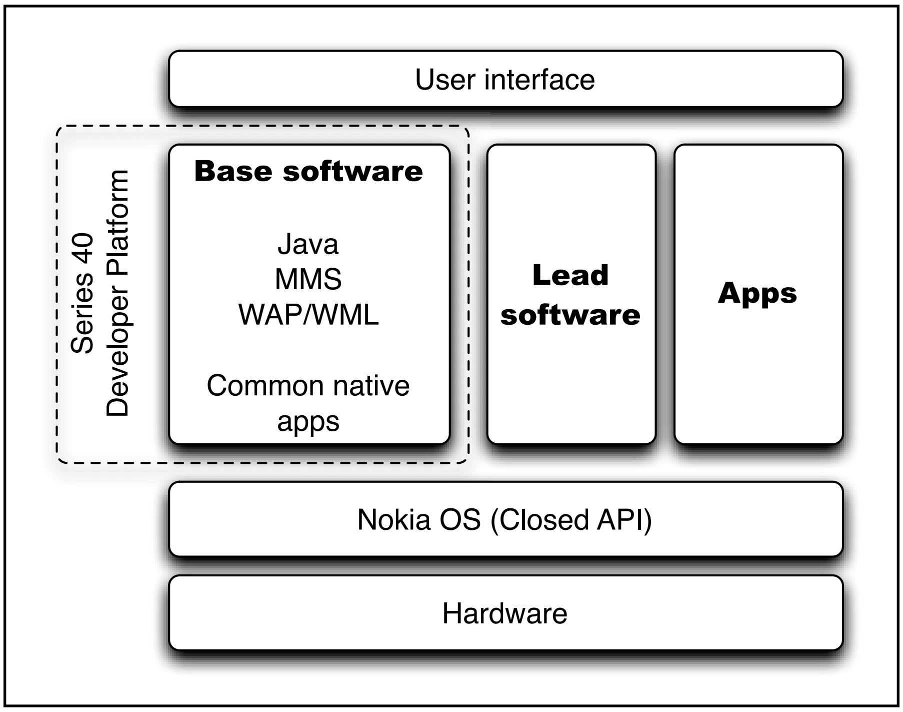


Figure 2–2 Software stack on Series 40 devices.

The common native applications are not customizable by third-party developers and hence not as interesting to the readers of this book. For developers, the key value proposition of the Nokia Developer Platforms is the support for technologies that enable third-party applications on the device. The Series 40 Developer Platform supports the following enabler technologies and APIs: the J2ME MIDP and its optional packages, WML, XHTML Mobile Profile, MMS, OMA DRM (forward lock), and OMA client provisioning technologies. The common native applications and the open API implementations constitute the base software in the Series 40 Developer Platform.

Nokia and wireless operators can also differentiate device offerings by installing "lead software," which are device-specific technologies or native applications. For example, the Nokia 6800 messaging phone for GSM networks extends Series 40 Developer Platform 1.0 with a very capable native email client; the Nokia 6255 imaging phone for Code Division Multiple Access (CDMA) networks extends Series 40 Developer Platform 2.0 with JSR-184 (Mobile 3D API).

Device Characteristics

A typical Nokia Series 40 device features a 128 by 128 LCD display with 4,096 colors. Some devices have 96 by 65 or 128 by 160 LCD screens and other color depths. It typically displays five lines of text plus headers. The keypad has the traditional alphanumeric keys, a four-way scroll key, the Send/End keys, and two or three generic soft keys. The device displays images in common file formats, receives AM/FM radio station signals, records voice messages, and plays Musical Instrument Digital Interface (MIDI) polysynthetic ringtones. Series 40 devices have multiple connectivity protocol support built into their hardware and OS.

- Series 40 devices support 2G and 2.5G wireless networks compatible with mobile operators throughout the world. Some work over GSM and GRPS, while others support CDMA networks.
- Some Series 40 devices support EDGE networks and 3G UMTS networks for fast wireless data transfer.
- All Series 40 devices support one or several of the following local network connectivity protocols: Bluetooth, USB, or Infrared Data Association (IrDA).

Device extensions such as cameras, full alphabetic keyboards, and MP3 players are available on selected Series 40 device models that target specific market segments. Figure 2–3 also shows the Nokia 7210 and 6230 devices, which are the first devices for the Series 40 Developer Platforms 1.0 and 2.0 respectively. The Nokia 6230 device supports a VGA camera, MP3 playback, and add-on MultiMedia Card (MMC) memory cards. The figure also shows a Nokia 6800 messaging phone (full keyboard) and a Nokia 3300 music phone (deck key layout and MP3 support).

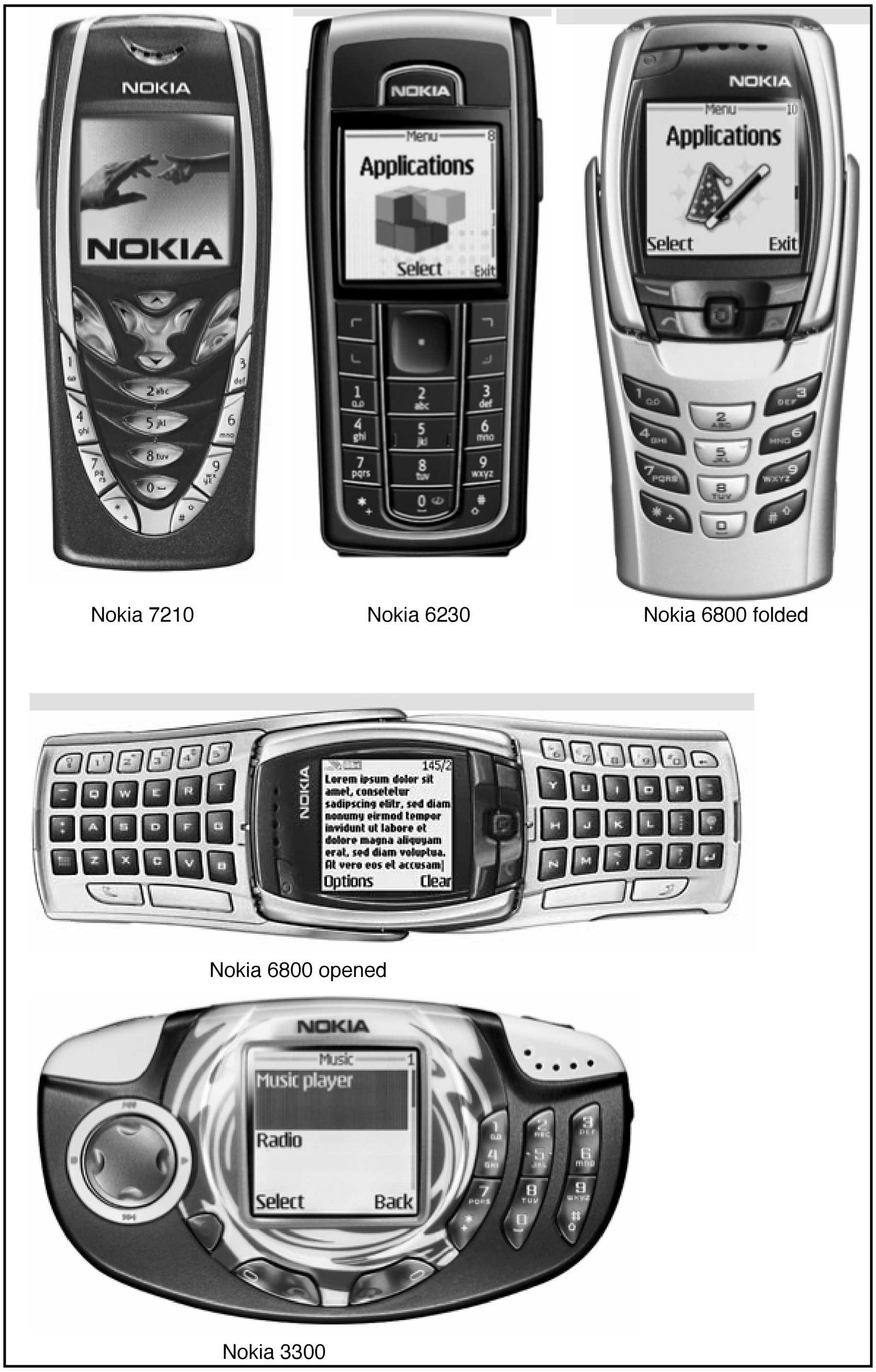


Figure 2–3 Important Series 40 Developer Platform devices.

TIP

We do not print the detailed physical characteristics and application limitations for each individual device in this book. For the most updated information about individual devices, please refer to the device specification document from Forum Nokia, available at *http://forum.nokia.com/devices.*

User Interface

The user interface on Series 40 devices is based on view-switch screens. It is designed specifically for one-hand operations.

1. An idle Series 40 device displays its home screen. After the user presses the Menu soft key, the device shows its top-level menu, which consists of a series of screens, each representing a different native application (e.g., the Web browser or messaging client) or content folder. The content folder could contain media files (i.e., pictures in the Gallery folder) or installed applications (i.e., Java MIDlets in the Applications folder). The user can navigate through the top-level menu items using the arrow navigation keys. The menu content and presentation of each menu item screen are determined by Nokia and the wireless operator. Developers cannot change them from J2ME applications.
2. When we select a top-level menu item by pressing the Select key, the next screen is a list menu. Each menu item takes up one line. For a native application, the list menu consists of available actions. For a content folder, the list menu shows content files, installed applications, or subfolders.
3. A Series 40 application typically consists of multiple screens. Application and navigation actions are assigned to each screen for the users to select. These actions are typically mapped to the soft keys. If there are more than two options, the left soft key becomes an Options key, which opens a full-screen selection list when pressed.

The above menu hierarchy is illustrated in Figure 2–4. Figure 2–5 shows the use of the Options soft key. The user interface on Series 40 devices is very screen-centric. Attempt to interact with individual UI elements (e.g., menu, selection list, options, or editable text box) often brings up a separate screen (see Figure 2–6 for examples). In Chapter 4, “MIDP User Interface,” we cover how to program the UI elements shown in Figure 2–6. This UI design is a proven success on small phone screens. Hundreds of millions of existing Nokia users are already familiar with it.

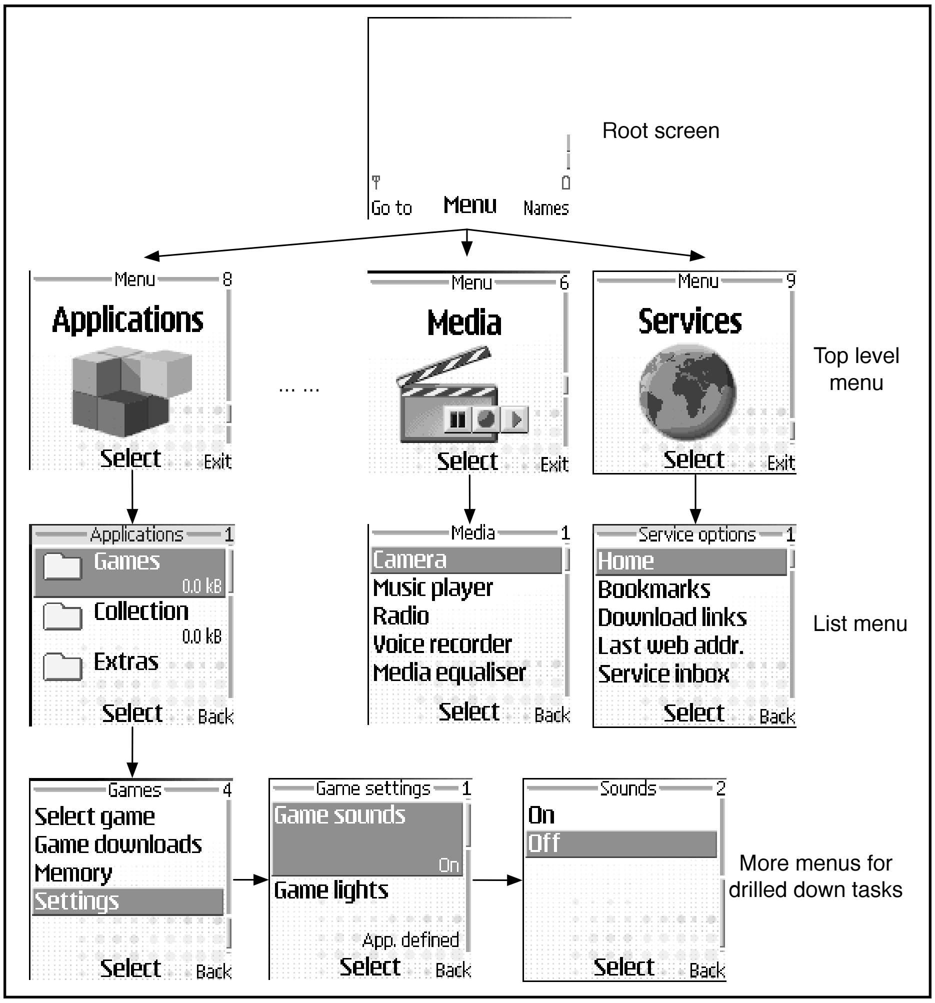


Figure 2–4 The UI menu for devices in the Series 40 Developer Platform.

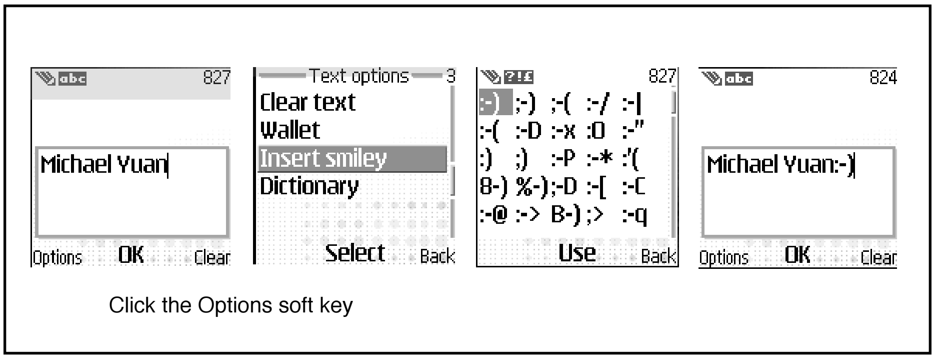


Figure 2–5 The use of the Options soft key in Series 40 devices.

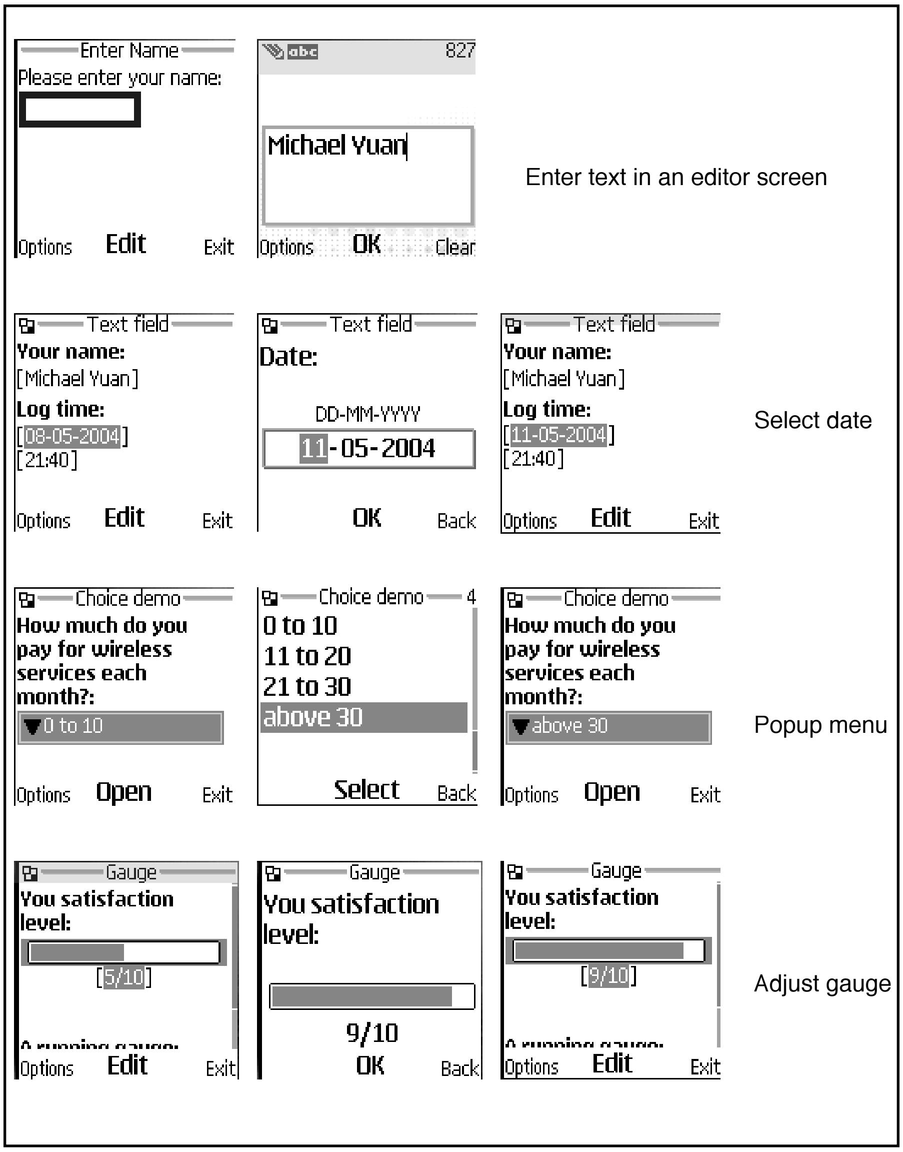


Figure 2–6 Interact with individual elements on a Series 40 device.

Series 60 Developer Platform

The Series 60 Developer Platform targets the world's best-selling smart phones produced by seven (as of April 2004) different vendors, including Nokia. More than 10 million Series 60 smart phones will be sold in 2004. There are a lot of overlaps between the Series 40 and 60 Developer Platforms. In this section, we focus on the enhancements brought by the Series 60 devices.

02Yuan.qrk 11/18/04 8:14 PM Page 35

NOTE

The Series 60 Developer Platform is different from the Series 60 Platform. The latter is a licensable product from Nokia. It is licensed to seven other device makers. Fourteen (as of April 2004) Series 60 smart phones have been launched. Series 60 Developer Platform is the platform for the developers. This book deals with Developer Platforms.

All Series 40 core native applications and most lead software are available on Series 60 devices. A significant difference between the Series 40 and 60 Developer Platforms is that Series 60 devices are based on Symbian OS instead of the proprietary Nokia OS. Developers can access the OS functionalities directly using Symbian C++ language and APIs. Users can install Symbian C++ applications into the device via OTA downloading or via a flash memory card. For example, most commercial games for the N-Gage game deck are written in Symbian C++. We give a brief introduction to Symbian OS later in this chapter. Figure 2–7 shows the software stack on Series 60 devices.

A Nokia Series 60 device typically has a 176 by 208 LCD screen capable of displaying 65,536 (16-bit) colors. More devices with other UI configurations will come in the future. Compared with a standard Series 40 keypad, a Series 60

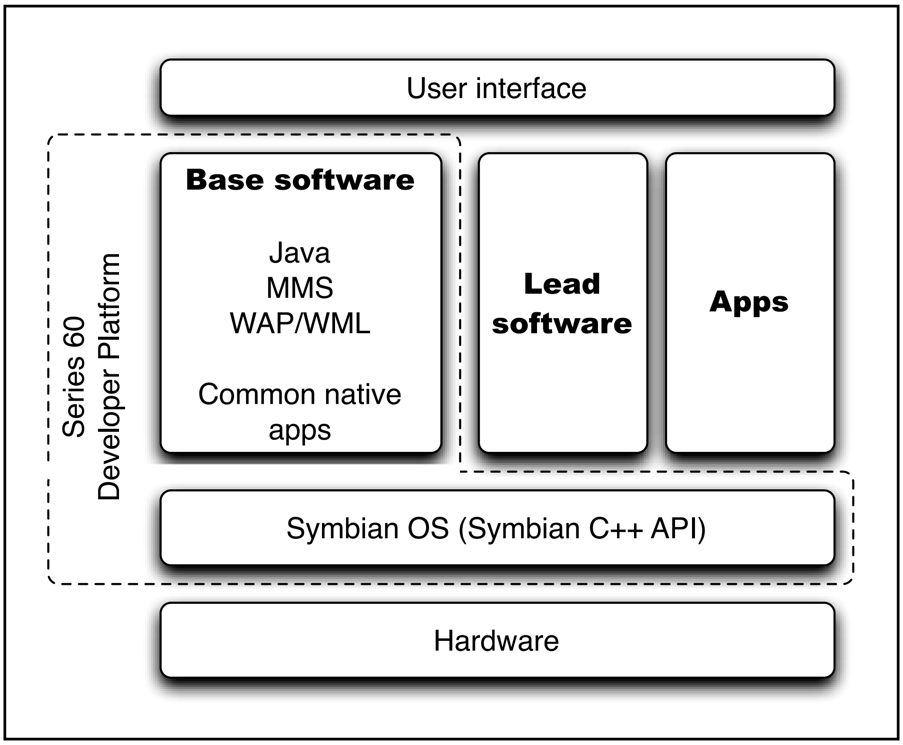


Figure 2–7 Software stack on Series 60 devices.

keypad has several additional keys, including an Application key, a Clear key, and an Edit key. A Series 60 device plays the Audio/Modem Riser (AMR) voice tones as well as other Series 40 audio formats. We can expand the data storage space of Series 60 devices using add-on flash cards. As a result, Series 60 devices can support large downloadable applications up to 4MB. The Nokia 3650 and 6600 smart phones are the first devices for the Series 60 Developer Platforms 1.0 and 2.0, respectively. A particularly interesting Series 60 device is the Nokia N-Gage mobile game deck. It is optimized for connected mobile games. Figure 2–8 shows the important Series 60 devices.

Compared with the typical user interface on a Series 40 device, a Series 60 device looks more like a minicomputer or PDA. The top-level menu and submenus can be displayed in a grid of icons or in a selection list. It supports popups (e.g., menus, option lists, and alerts) and directly editable widgets.

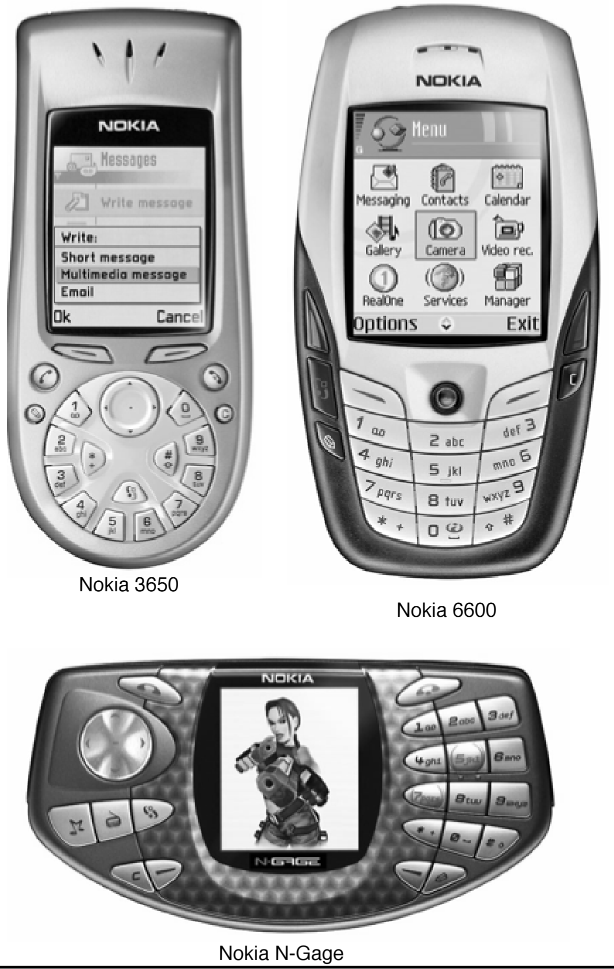


Figure 2–8 Important Series 60 Developer Platform devices.

Series 80 Developer Platform

The Series 80 Developer Platform is based on Symbian OS v7 and above. It is designed to support business productivity applications. A significant Series 80 addition to the Developer Platform base software is the support for J2ME Personal Profile. The J2ME Personal Profile is a more powerful Java environment than the MIDP, which is also supported on Series 80. The Personal Profile allows us to run enterprise mobile middleware, including many from IBM, on Series 80 devices. J2ME Personal Profile is not covered in this book. Series 40 and 60 MIDP, WAP, and MMS applications should work well on Series 80 devices. The Series 80 Developer Platform includes an array of enterprise-oriented lead software, including email client, messaging client, and VPN software.

The Series 80 Developer Platform was introduced in February 2004 with the Nokia 9500 Communicator device. It is a platform for enterprise devices. The Nokia 9500 Communicator features two user interfaces. An external 128 by 128 LCD screen and alphanumeric keypad are very similar to the UI design on Series 40 devices. But when opened, the device reveals a 640 by 200 large LCD screen and a full alphabetic keyboard. There are four soft keys along with the large LCD (see Figure 2–9). The Nokia Communicator 9500 user interface is clearly designed for two-hand operations. More UI designs will be available for Series 80 devices in the future.

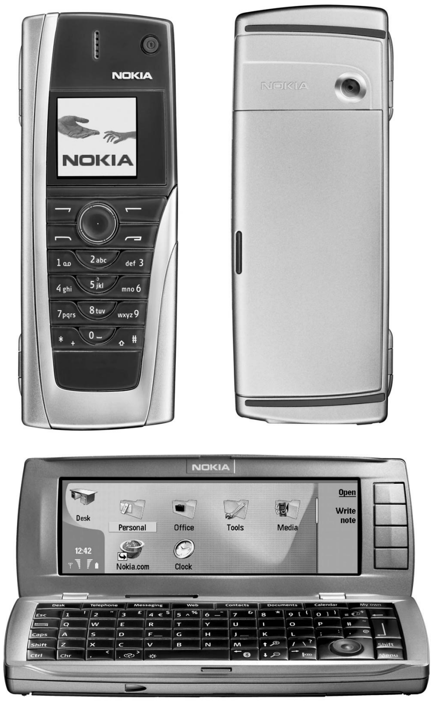


Figure 2–9 The Nokia Communicator 9500 is the first Series 80 Developer Platform device.

Series 90 Developer Platform

The Series 90 and 80 Developer Platforms are similar. The Series 90 is based on Symbian OS v7.0 and is primarily designed to support multimedia applications. For MIDP, WAP, and MMS applications, the multimedia enhancements and pen-based input methods are transparently available to developers. But for Symbian C++ developers, the Series 90 exposes more APIs to manipulate multimedia contents and UI events. Most Series 40 and 60 applications should run correctly on Series 90 devices with little or no change.

The Series 90 Developer Platform was introduced in late 2003. A typical Series 90 device features a 320 by 240 color display with 16-bit colors. It supports many audio and video playback formats and could allow users to watch TV programs or movies on the device. Series 90 devices feature a major UI upgrade from the Series 60: they support pen-based input methods (see Figure 2–10).

Other Nokia Device Series

In addition to Developer Platform devices, Nokia makes other devices. These devices are either legacy devices being phased out or devices that do not offer a significant opportunity for third-party developers. These are not the focus of this book.

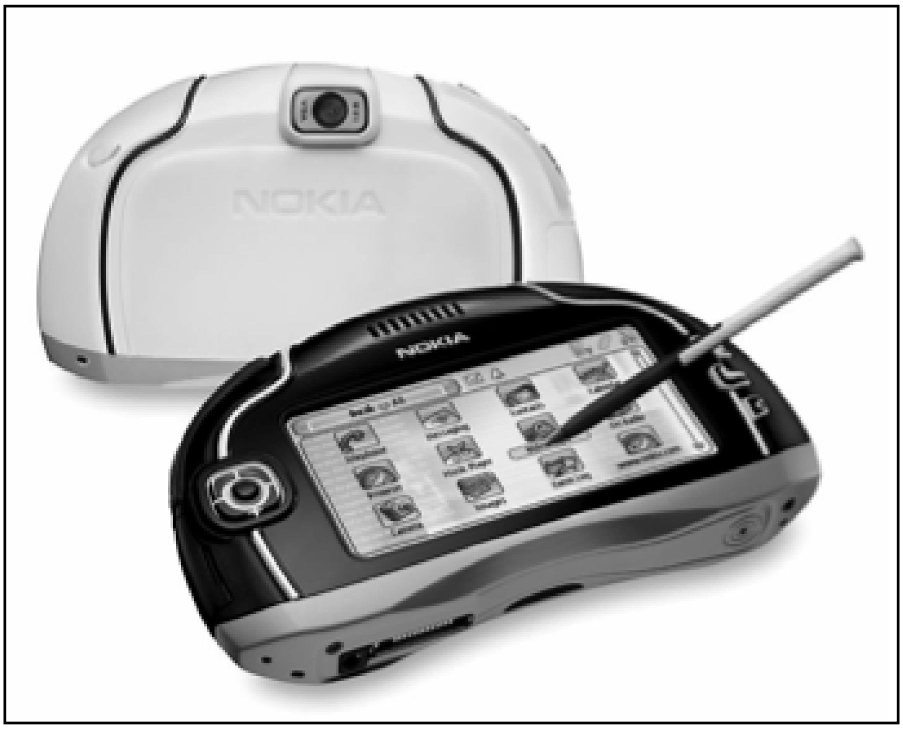


Figure 2–10 An example Series 90 Developer Platform device with pen-based user interface.

Pervasive Client Technologies: WAP and MMS

WAP browser and MMS messaging client are two technology pillars supported by all Nokia Developer Platforms. We cover the basics of those two technologies in this section.

Introducing WAP

A WAP browser works pretty much the same way as the HTML Web browser on a desktop PC. The user interacts with the remote application server by following dynamic links and submitting forms. The handset renders the content provided by the server. All the application logic is processed on the server side.

Although mobile and desktop browser applications share the same application model, the actual network architecture and markup languages are different. We check out those differences in the next several sections.

Network Architecture

While an HTML Web browser can make direct HTTP connections to the server, the WAP browser must go through a gateway server to connect to the general TCP/IP Internet. The WAP infrastructure is illustrated in Figure 2–11. The gateway converts data packets from the wireless network to TCP/IP format and then forwards them onto the wired Internet, and vice versa.

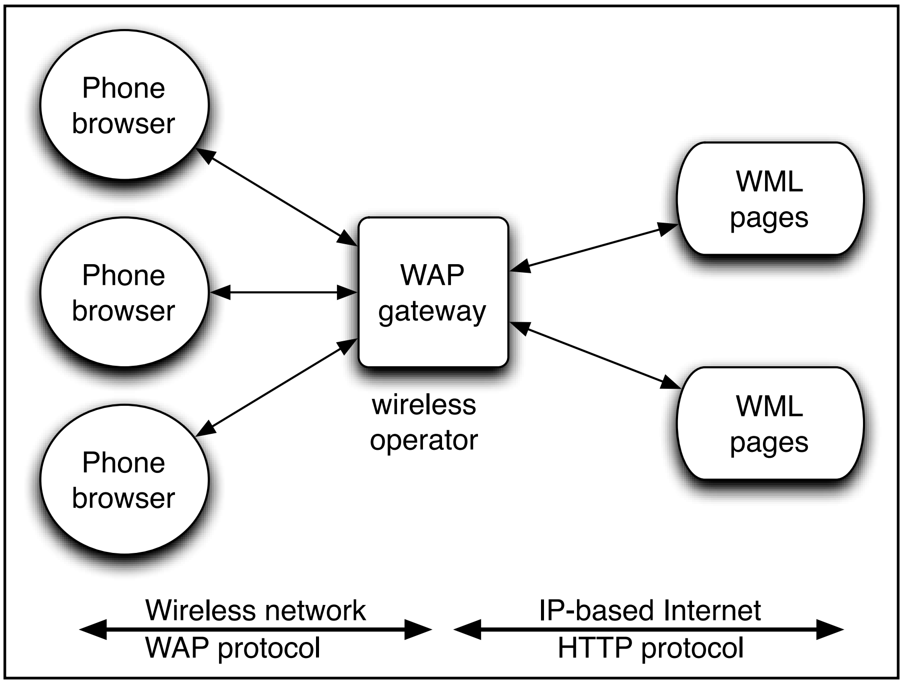


Figure 2–11 The WAP network infrastructure.

From the Web application developer's point of view, however, the gateway is almost completely transparent. All the developer needs to do is set up a normal HTTP server to serve the markup pages and other media objects. HTTP headers, including cookies and authentication credentials, pass through the gateway transparently. The gateway also handles encrypted HTTPS connections automatically.

WML

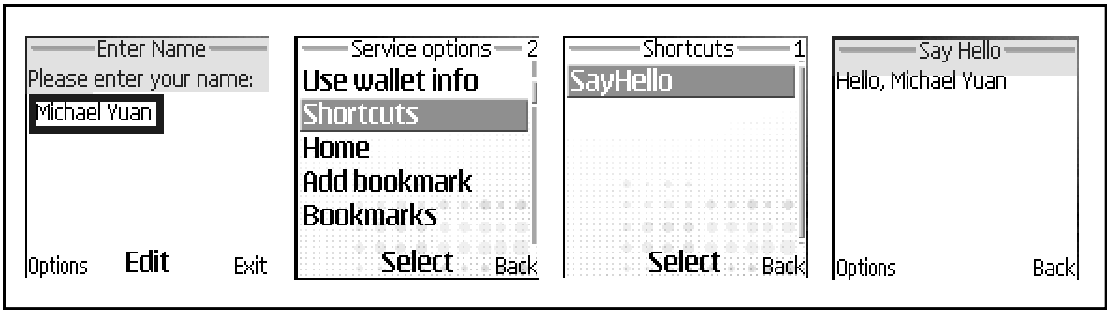
For developers, the biggest difference between an HTML Web application and a WAP wireless application is the different markup languages. Most mobile browsers support the Wireless Markup Language (WML), and all Nokia Series 40 and 60 devices support the WML specification. A core element in WML is `<card>`. Unlike HTML, where one page corresponds to one screen, one WML download page can contain a deck of cards denoted by the `<card>` tag. Each card corresponds to one screen and mobile device, and the user can navigate between cards using internal reference links. The cards help to break long content into several screens without requiring multiple round trips to fetch them one by one from the server. For example, the WML snippet below shows a deck of WML cards in one page, and Figure 2–12 shows how it looks on a device. The `<do>` tag maps a text label to a soft key. When the user presses on the soft key, the browser navigates to the page or card URL specified in the enclosed `<go>` tag.

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
          "http://www.wapforum.org/DTD/wml_1.2.xml">
<wml>
  <card id="Name" title="Enter Name">
    <do type="accept" label="SayHello">
      <go href="#Hello"/>
    </do>
    <p>Please enter your name:
      <input type="text" name="name"/>
    </p>
  </card>

  <card id="Hello" title="Say Hello">
    <p>Hello, $(name)</p>
  </card>
</wml>
```

XHTML MP

The XHTML markup language is developed by the W3C to replace HTML. It is HTML-defined as an XML document with cleaner and stricter syntax. Series 60



Please enter your name:
Michael Yuan

Hello, Michael Yuan

Figure 2–12 A deck of WML cards displayed on a cell phone screen.

devices and some Series 40 devices feature dual-mode WAP browsers that support both WML and XHTML. The browser conforms to the XHTML Mobile Profile (MP) specification, which contains a subset of most widely used XHTML tags. A key benefit of the dual-mode browser is that it allows users to access the vast amount of Web content out there on the wired Internet. The XHTML browser also supports WAP cascading style sheets (CSS) for styling.

Details about the WAP infrastructure, applications, markup languages, and Nokia device browsers can be found in Chapter 15, "Browser Applications."

Introducing MMS

An MMS message is analogous to an email message on the wired Internet. It contains a text body and any number of multimedia file attachments. The MMS client in Nokia Series 40 and 60 devices supports all popular attachment types, including JPEG, GIF, PNG, and MIDI. Some devices support advanced formats such as AMR TrueTone audio and 3GPP mobile video clips. You can send an MMS message to any MMS-enabled phone or ordinary email address. The message is delivered as follows:

1. The sender composes a message and sends it to the carrier's Multimedia Messaging Service Center (MMSC).
2. The MMSC forwards the message to the recipient carrier's MMSC or email server via the wired Internet.
3. The message is delivered to the recipient's phone or email inbox.

As we can see, the MMSC is central to the MMS architecture. We can write applications that connect to the MMSC directly over the wired Internet and send automated messages to a large number of users (see Figure 2–13).

An interactive MMS application functions like an automated email information service. It works as follows:

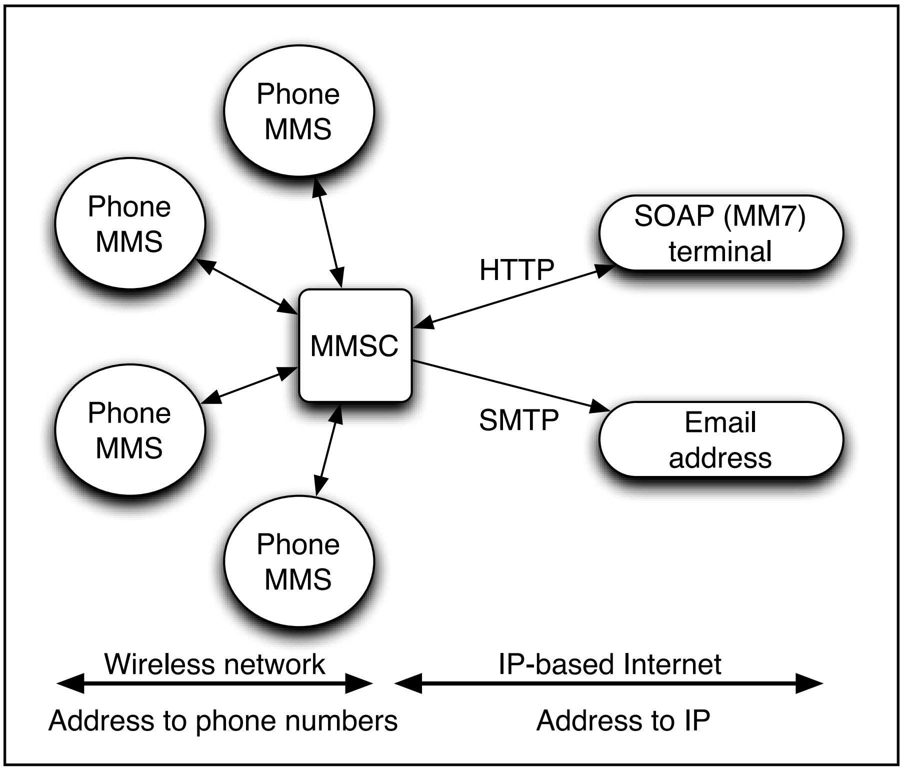


Figure 2–13 The MMS network architecture.

1. The user requests an application action by sending messages to the server.
2. The server returns the results via messages delivered to the phone.
3. The user then makes a further request by replying to that message.

This process goes on until the user stops replying to the message, thereby ending the session (see Figure 2–14).

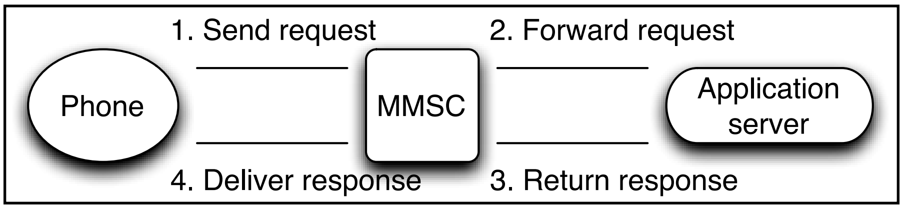


Figure 2–14 The MMS application interaction diagram.

NOTE

A major difference between WAP and MMS applications is that MMS applications are not "instantaneous." The message can be queued at the MMSC and scheduled for delivery later. The asynchronous messaging model trades the real-time performance for reliability. Temporary network problems do not cause the application to fail, since the message can be automatically scheduled for a later delivery time when the network recovers. If the delivery fails after a certain amount of time, the user can get a notification message.

SMIL

In addition to the text and multimedia components, the MMS message can also include a presentation component written in a special XML format called Synchronized Multimedia Integration Language (SMIL), which is also a W3C standard. A SMIL document contains time sequence instructions on how to display the attached multimedia components. The following SMIL example code instructs the MMS client to display image `demo.gif` and text `demo.txt` simultaneously on different parts of the screen for four seconds. At the same time, the client should play the `demo.midi` audio file.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE smil PUBLIC "-//W3C//DTD SMIL 2.0//EN"
      "http://www.w3.org/2001/SMIL20/SMIL20.dtd">
<smil xmlns="http://www.w3.org/2001/SMIL20/Language">
  <head>
    <layout>
      <root-layout width="320" height="240"
                   title="Demo"/>
        <region id="Image" width="150" height="60"
                                 left="0" top="0"/>
        <region id="Text" width="150" height="35"
                               left="0" top="70"/>
    </layout>
  </head>
  <body>
    <par dur="4s">
      <img src="demo.gif" region="Image"/>
      <text src="demo.txt" region="Text"/>
      <audio src="demo.midi"/>
    </par>
  </body>
</smil>
```

Not all devices support the SMIL component in MMS messages. Some earlier Series 40 devices ignore the SMIL attachment altogether but still allow the user to access other attachments in the MMS message. More details of the MMS infrastructure, applications, and SMIL are available in Chapter 14, "Multimedia Messaging Service."

The Thin-Client Application Paradigm

The WAP and MMS applications both run on servers. The handsets merely render the content and capture user interaction. This is commonly known as the *thin-client* application paradigm. It is a proven success in the Internet-based applications. Key advantages of this thin-client application model include the following:

- **The clients are pervasively available.** WAP browsers are almost universally supported by all device manufacturers and network carriers. The SMS and MMS messaging services are also widely available throughout the world. Several factors contribute to the pervasiveness of those technologies:
  - Since the device only handles presentation, it does not require much processing power. WAP browsers and messaging clients can be implemented on small, low-end devices with high sales volumes and long battery lives.
  - Since WAP has been around for a long time, most wireless data networks are well equipped to handle WAP traffic reliably. That makes thin-client applications available all over the world.
  - WML, XHTML, SMIL, and MIME attachments are standard technologies with a huge installed base worldwide. Most compatibility problems have been worked out over the years.
- **Thin-client applications and developers are readily available.**
  - The Web application and email application models are well known to today's Internet developers. They can easily migrate their skills to the new wireless arena.
  - A large number of Web applications are available today. It is relatively easy to make changes to their presentation layer so that they generate WML pages instead of HTML pages.
- **Thin-client applications are installed and deployed on the server end.** There are no complex and costly provisioning process, license management, security update, and so forth.

However, a crucial disadvantage of the thin-client paradigm is that it requires the mobile device to be always connected. Today's wireless data networks are slow, unreliable, and expensive. They cover only limited areas. Those limitations have severely hindered the adoption of thin-client applications. To get around the network problem, we have to rely on the other two pillars in the Nokia Developer Platforms: J2ME and Symbian C++.

Managed Smart-Client Technology: J2ME

J2ME brings rich and high-availability applications to occasionally connected mobile devices. It is universally supported by all versions of Nokia Developer Platforms as well as all other major mobile handset manufacturers. In this introductory chapter, we have a high-level overview of J2ME.

A Brief History of Java

The Java technology is emerging as one of the most important enablers for mobile applications. In 2007, Java handset shipments will reach more than 450 million, constituting 74 percent of all handset shipments. Java mobile devices will soon surpass Wintel PCs and become the dominant information access clients. Nokia is a major player in the Java landscape. The technical benefits of Java include:

- **Crossplatform:** This is very important in the diverse mobile device market. For example, the same J2ME MIDP application runs on all Nokia Developer Platform devices with relatively small amount of modification, representing huge cost savings for developers.
- **Robust:** Since Java applications run in a managed environment, the bytecode is verified before execution, and unused objects are reclaimed by garbage collectors (see the tip). Even if a Java application does crash, it is contained within the virtual machine. It will not affect other applications on the device.
- **Secure:** The Java runtime provides advanced security features through a domain-based security manager and standard security APIs.
- **Object oriented:** The Java language is a well-designed, object-oriented language with vast library support. There is a vast pool of existing Java developers.
- **Wide adoption at the backend:** It is relatively easy to make Java clients work with Java application servers and messaging servers. Due to the wide adoption of Java 2 Enterprise Edition (J2EE) on the server side, J2ME is the leading candidate for end-to-end mobile applications.

TIP

The garbage collector periodically travels the directed graph of allocated objects and frees up all objects that cannot be reached via a valid reference. That could create certain conditions for memory leaks. For example, if a long-lived object holds references to short-lived objects, even after the short-lived objects are no longer used, their memory cannot be freed because they are still reachable in the linked graph from the long-lived object. As a result, we should be extremely careful when adding object references to collections held in long-lived objects such as the root UI window, the MIDlet object itself, or Singleton objects. Since the heap space on mobile devices is limited and the garbage collector takes time to run, it is generally considered a best practice to minimize object creation and reuse objects as much as possible.

For memory-intensive applications, it is sometimes hard for the garbage collector to keep up. Failures to free stale objects in time could cause out-of-memory errors. To correct this problem, you can manually invoke the garbage collector by calling the `System.gc()` method in your code. It asks the JVM runtime to make best effort to reclaim memory space before it returns.

From WORA to Java Everywhere

For early Java, the term *crossplatform* has a strict meaning: the same bytecode application should run without modification on any computer that has a Java runtime. The original vision is that Java-based software agents could roam over the network automatically. That not only requires bytecode compatibility but also runtime library compatibility. But as Java evolves, it is used in many different application scenarios. The single class library approach no longer fits the needs.

Recognizing that one size does not fit all, the Java 2 Platform is divided into three editions. The Java 2 Standard Edition contains the basic JVM and core class libraries; the Java 2 Enterprise Edition provides additional class libraries and tools for enterprise server applications; the Java 2 Micro Edition consists of stripped-down virtual machines (called KVMs—Kilobyte Virtual Machines) that can run on devices with kilobytes of memory. For mobile devices, a subset of the standard edition class library and new libraries for mobile-specific tasks. It is clear that a Java bytecode application written for an enterprise server will not run crossplatform on a PDA device without modification.

In June 2003, during the eighth JavaOne conference in San Francisco, Sun Microsystems brought up a new slogan for Java: “Java Everywhere.” The emphasis is no longer on direct portability of bytecode applications. The focus now is to provide the same language, consistent architectures, and similar APIs across all computing platforms. Java Everywhere allows developers to port their skills to new application arenas.

The J2ME Architecture

The separation of J2EE, J2SE, and J2ME is a step in the right direction. However, a single monolithic J2ME is still too inflexible for mobile devices. There is a huge variety of mobile devices, designed for different purposes and with different features. For example, applications on an automobile-mounted system are much more complex than those on a cell phone. Even among similar devices, such as high-end and low-end cell phones, portability can cause underutilization of resources on one device and strain on another. Device manufacturers and developers need fine-grained API differentiation among devices, not the "lowest common denominator."

To balance portability with performance and feasibility in the real world, J2ME contains several components known as configurations, profiles, and optional packages (Figure 2–15). Each valid combination of a configuration and a profile targets a specific kind of device. The configurations provide the most basic and generic language functionalities. The profiles sit on top of configurations and support more advanced APIs, such as a graphical user interface (GUI), persistent storage, security, and network connectivity. The optional packages can be bundled with standard profiles to support specific application needs.

NOTE

Even with J2ME, device-specific optimization is still a major challenge in mobile application development. A single Java code base cannot account for the different screens, CPUs, memory sizes, Java API libraries and even JVM implementation bugs, found on different devices. As we discussed, Nokia reduces the required optimization work by developing relatively consistent handsets within each Developer Platform. The focus of this book is to help the readers understand the J2ME characteristics of Nokia Series 40 devices and then develop applications optimized for those devices. In Chapter 12, "Developing Scalable Applications," we cover how to scale J2ME applications across different devices within and beyond the Nokia Series 40 Developer Platform.

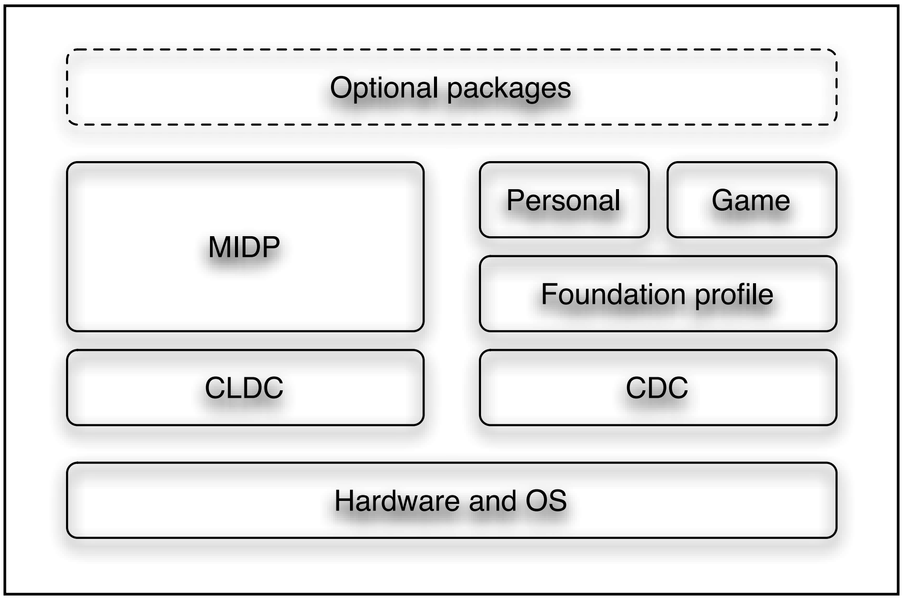


Figure 2–15 The J2ME architecture.

The two most important J2ME configurations are as follows.

- **The Connected Limited Device Configuration (CLDC)** is for the smallest wireless devices with 160KB or more memory and slow 16/32-bit processors. The CLDC has limited math, string, and I/O functionalities, and lacks features such as the Java Native Interface (JNI) and custom class loaders. Only a small subset of J2SE core libraries is supported by the CLDC virtual machines (KVMs). The most recent version of the CLDC is version 1.1. It was developed by the JSR 139 and released in March 2003.
- **The Connected Device Configuration (CDC)** is for more capable wireless devices with at least 2MB of memory and 32-bit processors. Unlike the CLDC, the CDC supports a fully featured Java 2 virtual machine and therefore can take advantage of most J2SE libraries. The CDC 1.0 was developed by the JSR 36, and it became available in March 2001. The new CDC 1.1 is currently being developed by the JSR 218 and is expected before the end of year 2004.

Important J2ME profiles include the following. The **Mobile Information Device Profile (MIDP)** is built on top of the CLDC to provide support for smart phones; the **Foundation Profile** is built on top of CDC to provide support for networked embedded devices; the **Personal Basis Profile (PBP)** and **Personal Profile (PP)** are built on top of the CDC and the Foundation Profile to provide support for GUI-based powerful mobile devices such as high-end PDA devices. The standard UI library in the current PBP and PP editions is the Java AWT (Abstract Widget Toolkit).

On the CDC and Personal Profile stack, important optional packages include the following: the RMI (Remote Method Invocation) Optional Package (JSR 66) supports remote object sharing between Java applications; the JDBC (Java DataBase Connectivity) Optional Package (JSR 169) provides a uniform interface to access structured query language (SQL) databases from Java applications; the Advanced Graphics Optional Package (JSR 209) aims to add Swing and Java 2D API libraries into the CDC/PP stack.

Although CDC and PP have their places in the mobile market, they are not nearly as popular as the MIDP. All major mobile device manufacturers, including Nokia, are committed to support MIDP. In the next section, we take a deeper look at MIDP and its optional packages.

NOTE

The concept of open interfaces is core to the Java technology. It works as follows: For a given computing task, a set of standard APIs is defined by a standards committee. Individual vendors then provide competing libraries that implement those APIs. The application code using the API is complete-ly decoupled from the specific implementation provider. That approach minimizes the developer's learning cost and improves code portability. Yet, it also protects the freedom of choosing vendors. The Java Community Process (JCP) is an effort to develop standard Java API specifications.

JCP Executive Committees (ECs) consist of industry-leading companies. Anyone in the general public can submit a new Java Specification Request (JSR) for a new API. The appropriate EC decides whether to accept this new JSR. Once approved, the JSR lead can recruit more companies or individuals to develop the API specification together. Every specification goes through multiple stages of community and public reviews before it becomes an official Java standard.

MIDP and Its Optional Packages

The most important and successful J2ME profile is the CLDC-based MIDP. The MIDP targets the smallest devices, such as smart phones. It is already deployed on millions of handsets, including all Nokia Series 40 and 60 devices. Hence, the MIDP is a key technology that all Nokia developers need to learn.

An MIDP application consists of a suite of MIDlets. Each MIDlet can be independently installed, started, paused, and stopped by the application management software (AMS) on the device. The AMS can be controlled by the user, using the phone keypad. The MIDlet API specification provides a set of abstract life-cycle methods that hook into the AMS. Developers must implement these methods to specify the runtime behavior of each MIDlet. The code for a minimal MIDlet that displays a `Hello World` string on the screen is as follows. The details of the code are explained in Chapter 3, "Getting Started."

```
package com.buzzphone.hello;

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

public class HelloMidlet extends MIDlet {

  Form form;

  // Called by the AMS when the MIDlet is instantiated
  public HelloMidlet () {
    form = new Form ("Hello");
  }

  // Called by the AMS when it starts the MIDlet
  protected void startApp() {
    form.append ("Hello World");
  }
  // Called by the AMS when it stops the MIDlet
  protected void destroyApp(boolean unconditional) {
    destroyApp(false);
    notifyDestroyed();
  }

  // Called by the AMS when it pauses the MIDlet
  protected void pauseApp() {
  }
}
```

As of late 2003, most mobile phones in the market support the MIDP 1.0 specification. However, the MIDP 1.0 lacks some important features, such as security and advanced UI controls. As a result, device vendors often supply their own MIDP extensions to provide advanced custom features. Vendor-specific extensions undermine the portability of J2ME applications. Many problems with the MIDP 1.0 have been fixed in the MIDP 2.0, which came out of JCP in August 2002. The Nokia Developer Platforms 2.0 for Series 40 and 60 mandate MIDP 2.0 and optional packages on new Nokia phones.

Table 2–1 lists MIDP-compatible optional packages. Most of the MIDP optional packages run on CDC profiles as well. Nokia supports the optional packages at different levels:

- **Device available:** The optional package is already factory-installed on some Nokia devices.
- **Coming soon:** Device implementation of the optional package is currently being developed by Nokia engineers. It will be available on new devices soon.
- **Specification:** The optional package specification is still being developed by the JCP. Nokia supports that specification by contributing to the expert group.
- **No plan:** Nokia currently has no plan to support this optional package on its devices.

NOTE

The Java Technology for the Wireless Industry (JTWI) specification is a guidance and roadmap document for Java handset manufacturers and developers. It specifies the minimal software and hardware requirements for Java smart phones that can be marketed with the JTWI logo. A JTWI-compatible handset must support MIDP, the Wireless Messaging API, and the Mobile Media API.

Table 2–1 MIDP Optional Packages

Name	JSR	Nokia Support	Description
File I/O and PIM	75	Device available	This optional package has two modules: the file I/O module supports access to file systems on a PDA device; the PIM module allows the MIDP application to integrate with the device's native PIM clients.
Mobile Media	135	Device available	Provides audio and video capture and playback APIs. The exact supported media formats vary by devices. It is covered in detail in Chapter 9.
Wireless Messaging	120/205	Device available	Provides an API for the MIDP application to send and receive SMS and MMS messages. It is covered in detail in Chapter 8.
Location	179	Coming soon (1H2005)	Supports location tracking for devices. The location information can come either from a GPS device module or from the network carrier.
Web Services	172	Coming soon	Provides XML APIs for generic XML parsing as well as SOAP Web Services clients.
Bluetooth	82	Device available	Supports access to Bluetooth data channels and protocol libraries from an MIDP application. It is covered in detail in Chapter 10.
Security and Trust	177	Coming soon (1H2005)	Allows MIDP applications to interact with the phone's embedded security module such as the SIM card for GSM phones.
3D Graphics	184	Device available	Provides an API to display 3D scenes on a mobile device. A lightweight mobile 3D data format for the art works is also defined.
Content Handler	211	Coming soon (1H2005)	Allows devices to associate MIME types with MIDlet applications. Media files with certain MIME types will be automatically opened by the associated MIDlet.
Scalable 2D Vector Graphics	226	Coming soon (1H2005)	Provides capability to render 2D vector images in the SVG (Scalable Vector Graphics) format.

Table 2–1 MIDP Optional Packages (continued)

Name	JSR	Nokia Support	Description
SIP (Session Initiation Protocol)	180	Coming soon (1H2005)	Provides support for SIP-based communication. It will allow data to be pushed to mobile devices.
Presence and IM	165/187	No plan	Supports presence and instant messaging applications based on the SIP.
Data Sync	230	Specification	Supports synchronizing PIM databases over the network. This optional package also provides APIs to process most common PIM data formats.

The Smart-Client Paradigm

Microbrowser-based thin-client technologies were instrumental in bringing mobile Internet to masses in the early days of mobile commerce. But WAP-based mobile commerce has never taken off due to the poor usability on the client side. The new generation of smart-client and mobile middleware technology (e.g., J2ME and Microsoft's .NET Compact Framework) promises to bring feature-rich clients to mobile applications. The benefits of smart clients over thin clients include the following:

- **Smart clients have richer and more pervasive user interfaces.** In particular, the judicial use of threads can drastically improve user perception of the application performance.
- **Smart clients can be more easily personalized.** Extreme personalization is one of the most touted benefits of the freedom (mobile) economy.
- **On-device data storage reduces network traffic, especially unnecessary round trips.** It enables transactions; supports the "offline" mode when the network is temporarily unavailable; and hence improves overall performance, reliability, and availability of mobile applications.
- **Smart clients can leverage device extensions.** For example, a smart-client program can talk with the device's built-in (or attached) GPS module and barcode scanners. A smart client can also integrate with device-specific software (e.g., email and messaging clients) to improve the user's workflow.
- **Smart clients support more powerful and flexible security schemes.** They enable content-based security and distributed single sign-on.

- **Smart clients support advanced integration technologies.** They are easy to plug into existing corporate infrastructure. Supports for asynchronous messaging and XML Web Services are crucial for reliable and maintainable mobile solutions.

Tightly Integrated Smart-Client Technology: Symbian C++

The Symbian OS is a sophisticated 32-bit operating system designed specifically for mobile devices. It consumes few resources and yet has a modular, object-oriented C++ architecture. It is based on preemptive multitasking and supports threading and asynchronous processing. It was anticipated that Symbian devices could run for years without being switched off, so reliability and stability were key design goals for the OS.

The Symbian C++ API provides complete access to services, such as messaging and multimedia, as well as device and OS functionality that is not available through the use of J2ME. Symbian OS is an open developer platform available on Nokia Series 60 and higher devices as well as on devices manufactured by other Symbian OS licensees.

The Evolution of Symbian OS

Symbian devices are proliferating because of Symbian's position as an open operating system for data-enabled mobile phones. Currently, Symbian has the most partners and licensees of any mobile OS, including Nokia, Sony Ericsson, Motorola, Siemens, Fujitsu, Samsung, Sanyo, and others.

The operating system began as software for PDAs from a company called Psion. Symbian was formed in 1998 to evolve this OS primarily for phones. These high-end phone handsets are now known as smart phones. The Symbian OS is layered to support different device designs while retaining core functionality across all products. Three families of product lines emerged:

- **Keypad-based:** These are designed for one-handed operation and do not have a touch screen. They are currently the most common type of Symbian devices, and Series 60 exemplifies this design. Nokia created the Series 60 platform on top of Symbian OS and licenses it to other manufacturers such as Siemens, Samsung, Panasonic, and Sendo. This gives consumers more choice while allowing them to exchange data, use compatible software, and switch smart phones without having to learn a new interface.
- **Pen-based:** These phones include a stylus for touch-screen operation. There are now two lines of pen-based Symbian handsets: UIQ phones and the Series 90 Developer Platform. The Sony Ericsson P800 was the first device

02Yuan.qrk 11/18/04 8:14 PM Page 54

with a UIQ user interface. Nokia does not make UIQ devices but has introduced the Series 90 Developer Platform, which represents the latest in mobile technologies.

- **Keyboard-based:** These phones, such as the Nokia 9500 Communicator, are the most similar to handheld personal organizers. They have a full keyboard as well as a touch screen for pen-based input.

In the future, user input will not be the primary distinction between these designs. There will be some convergence of product features, and the addition of new features will add different distinctions between product lines.

Symbian OS Architecture

Symbian OS API (Figure 2–16) contains hundreds of C++ object classes grouped in subsystems. We can also group these subsystems in layers.

In general, we can think of the Symbian architecture in four layers or groupings: the application utility layer, the GUI framework and services, communications, and base system APIs.

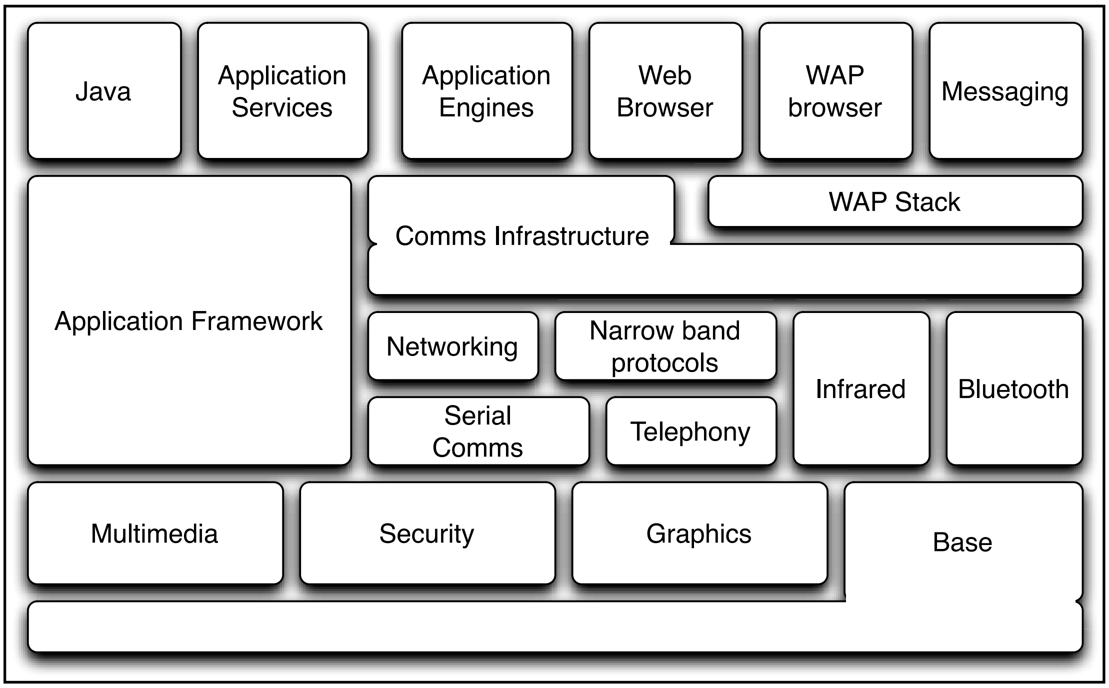


Figure 2–16 The Symbian OS API architecture.

- **The application utility layer:** This includes a variety of application-oriented utilities. Application engines give access to the data from built-in PIM applications, such as contacts and calendar schedules. This allows third-party applications to integrate with core applications easily. Other application services include specialized data management and data exchange.
- **The GUI framework and services:** The framework APIs give structure to third-party applications and provide for UI handling. These include UI controls and lower level APIs for multimedia handling of sounds and graphics. Symbian platforms such as Series 60 and UIQ extend the UI frameworks to provide for different UI designs. When developing Symbian applications, it's best to separate the UI and application logic. This limits the amount of code that needs to be ported between platforms.
- **Communications:** There's a broad stack of communication-related APIs. At a high level, there are messaging and browsing utilities. Beneath that is support for networking interfaces such as Bluetooth, infrared (IrDA), and USB; protocols such as TCP/IP, HTTP, and WAP; and of course mobile telephony services.
- **Base system APIs:** The base APIs encompass class libraries for data structures, file and memory access, date and time, and other basic system APIs.

Although it is more effort to develop Symbian C++ applications, there are compelling reasons to do so. As natively compiled C++ applications, Symbian applications can run much more quickly than J2ME applications. Depending upon the requirements of the solution, a Symbian application may be the only choice available. Symbian provides extensive APIs that give access to almost all the functionality in a handset now, whereas not all the MIDP 2.0 optional packages are available yet.

Get Connected

The success of Nokia Developer Platforms will be ultimately tested by developer adoption. Nokia provides valuable services to wireless operators, developers, content owners, and business managers who want to leverage the Developer Platforms to reach hundreds of millions of device users.

Leading Platforms

The core value behind the Nokia Developer Platforms is the large volume of shipped devices. By March 2004, more than 40 Developer Platform devices had been launched. More than 100 million Developer Platform device units will be shipped in 2004. As the mobile handset market leader, Nokia's commitment to Developer Platforms allows developers and content owners to connect to the volumes via the minimum learning curves. It also eases the decision-making process for business managers who need to identify which handsets to support.

Developer Resources

Forum Nokia, the developer arm of Nokia, provides superb support for the Developer Platforms. Forum Nokia publishes software development tools, documentations, and white papers. The white papers cover a wide range of topics from technical tutorials to best practices to business case studies. Developers can access the latest devices and mobile service servers via the Forum Nokia loaner device and developer hub services. Forum Nokia also provides technical support via telephone and Internet discussion forums. It has more than 1.35 million registered users, more than 460,000 tool and document downloads every month, and more than 17,000 unique visitors everyday. Forum Nokia allows developers to connect to Developer Platform–related answers.

In early 2004, Nokia launched the Forum Nokia PRO service. For a small annual fee, companies can gain early access to tools, the latest prototype devices, confidential documents, and proprietary technical support from Nokia.

Business Generation

Nokia helps developers to get applications to the market. For large developers, Nokia provides opportunities to include custom applications directly on shipped devices. For example, Developer Platform devices can be shipped with add-on MMC flash cards that have third-party applications preinstalled. For smaller developers, the Nokia Tradepoint program is a worldwide online application and service catalog. In March 2004, there were more than 2,500 applications and 200 buyers in the Tradepoint channels. In addition, Nokia sponsors co-marketing events with local developers and buyers around the globe.

Together with Sun Microsystems and other mobile handset vendors, Nokia provides certification services for Java applications. A certificate guarantees that the application works correctly with Nokia Developer Platform devices and hence makes it eligible for software publisher catalogs.

Nokia's catalog, co-marketing, and certification services help developers connect with customers.

Summary

The Nokia Developer Platforms enable us to develop portable and scalable mobile applications for hundreds of millions of Nokia devices. In this chapter, we covered the Series 40 and 60 Developer Platforms and introduced the four enabler technology pillars. They are WAP, MMS, J2ME, and Symbian C++. We reviewed the application paradigms each technology enables and discussed their strengths and shortcomings. It is crucial for us to understand those technologies and know how to apply them correctly to suit specific application needs. Near the end of this chapter, we also covered Nokia's developer support programs that connect developers to volumes, answers, and customers.