Smart Client Architecture

By now you are probably aware that smart client applications differ significantly from thin client, browser-based wireless solutions. As a general rule, smart client applications enable the user to access data when disconnected from the network. There are many ways that this can be accomplished, but for the purposes of this section, we are going to focus on building applications that incorporate a persistent data store, rather than just using simple caching mechanisms.

The smart client architecture is illustrated in Figure 7.1. On the client, you have the user interface, the business logic, as well as a persistent data store. This application communicates with a back-end data source, often through an intermediate synchronization server. The communication stream itself can run either wirelessly or over a wireline connection. Depending on the technology being used, the connection may require an IP-based network or an additional communication layer for the synchronization process.

When this type of solution is implemented, it quickly becomes clear why it is called a smart client solution. By deploying an application to the device itself, you have the ability to give the client application some "smarts," or logic. This logic dictates many aspects of the application. It determines where the application gets it data from (either locally or from a round-trip to the server), how the data is presented and stored, as well as the set of data that needs to be communicated back to the enterprise systems via a synchronization process. In the wireless space, the impact of having business logic on the device is often overlooked as many vendors and developers focus on low-level technical features of a solution rather than satisfying the requirements of the mobile user.

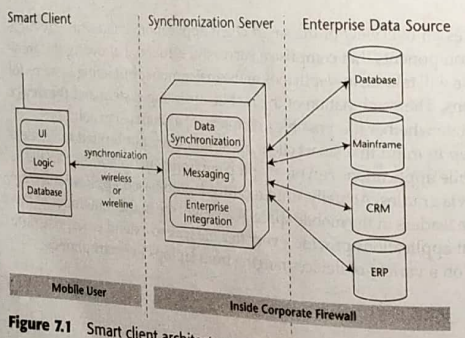

Figure 7.1 Smart client architecture.

The Client

Smart client applications provide many attractive features for end users. Many of these features reside in the client application itself. By providing a rich user interface with persistent data storage, smart client applications are suitable for a large variety of corporate applications. We are going to take a closer look at each of the components that comprise the client of smart client applications.

User Interface

Initially, the user interface may not seem important when developing mobile applications, as most people feel that there is always going to be a trade-off between size and usability. But this trade-off does not always have to take place. Smart client applications can have very sophisticated user interfaces. In most cases, they are programmed using development tools that have extensive component frameworks. Incorporating tables, drop-down lists, radio buttons, and graphs are straightforward operations. Contrast this with the capabilities available for most wireless Internet applications, where the user interface is often text-based with limited support for graphical components.

Whether you are developing for the Palm OS, Windows CE, Symbian OS, or a Java-based system, you will find that user interface development is a very important part of a successful application. When the screen size is limited, as it is for handheld devices, the developer must take full advantage of the space that is available. In addition, putting special focus on the navigation through an application is essential. Mobile application users are often the same people who have been using advanced Windows-based application for years; hence, they have certain usability expectations that need to be met. One of the main reasons given by consumers for their slow acceptance of wireless applications is the complexity of the available solutions. This complexity is usually due to poor application design, rather than technical limitations. Obtaining end-user feedback during the development phase via a prototype can help you address usability concerns at an early stage in the application development process. In the *Mobile Operating Systems* section later in this chapter, we will take a look at the most popular mobile operating systems and review the types of applications that each operating system can support.

Data Storage

When you look at the requirements of typical mobile enterprise customers, you see some trends emerge. First and foremost, data access is a requirement. Users do not really care how the data is retrieved; they care only about its availability. Theoretically, they understand that wireless access is not pervasive; nevertheless, they quickly become frustrated if they can access their data only periodically, depending on the network coverage and/or penetration. By providing data storage capabilities on the device, you can ensure that users will be able to access important data when they need it. (Note: Data storage mechanisms are not covered here, as they are covered in depth in Chapter 9, "Persistent Data on the Client.")

Performance

In addition to providing timely data access, smart client applications also provide performance benefits. Consider how computer technicians work: They travel from location to location repairing a variety of computer systems. As often as possible they carry some inventory with them for common repairs; but just as often, they need to order a special part from the warehouse and schedule a follow-up visit. Before they can schedule the next visit, they have to make sure that the part in question will be available at a time that fits into their schedule. By utilizing a smart client application that contains a persistent data store, technicians can have immediate access to inventory information, as well as their schedule. Imagine if they had to search a database for a specific part number. This task could take a considerable amount of time if a wireless browser-based application were being used. The task could even be impossible if a technician happened to be at a location that was outside of the wireless network's coverage. By implementing a mobile data store, technicians can perform a quick search of the inventory right on the device to provide them with the information that they require.

The one concern with this approach is data freshness. If the data on the device is not updated regularly, then the technician may end up, for example, believing that a part is available, when in fact it is out of stock, leading him or her to make decisions based on inaccurate data. This leads to the data synchronization aspect of smart client applications.

Data Synchronization

In most smart client architectures, the majority of the data synchronization work is executed, not on the client, but on a synchronization server. Nevertheless, the client application is still required to have a certain amount of synchronization knowledge. Minimally, the client has to know the location of the synchronization server, details about the communication stream to the server, what data has to be synchronized, and how to handle incoming data from the server.

If you are developing the synchronization layer yourself, you will have to keep these issues in mind when designing the application. If you are going to use commercially available software, most vendors provide a synchronization client module that incorporates the required functionality with various levels of sophistication. (Chapter 10, "Enterprise Integration through Synchronization" provides a closer look at the options available for data synchronization for both the client and the server.)

Messaging

Smart client applications can also take advantage of intelligent application-to-application messaging systems to communicate data. To do so, a client component has to be able to both send and receive these messages. When receiving a message, it has to be smart enough to relay the information appropriately. This communication may involve notifying the user directly or, possibly, updating a set of the data in the client data store. Application-to-application messaging can be important for applications that require frequent communication with enterprise systems. (For more information on all of the leading forms of mobile and wireless messaging, refer back to Chapter 5, "Mobile and Wireless Messaging.")

The Server

Though the server component of smart client applications is invisible to the end user, it is still very important. The server component is responsible for data synchronization, data storage, and messaging.

Data Synchronization

When we refer to data synchronization, we are talking about how enterprise data is moved from the back-end enterprise system to the mobile device, and vice versa. This data movement can be accomplished in a number of ways, depending on the solution that is chosen. For a general overview, we will look only at the major components that are required in a synchronization server, as shown in Figure 7.2. These components include the interface to the client application, the synchronization logic, and the integration layer to the back-end data source.

Data can be sent between the client application and the synchronization server in a variety of formats. Transferring data over a wired connection is a fairly straightforward process. Transferring it over a wireless network can be a different story. Depending on the requirements of the synchronization software, some wireless networks may work better than others, and some may not work at all. Discussions about wireless data synchronization are saved for Chapter 10; for the purpose of this overview, we will assume a valid connection exists between the client and the server.

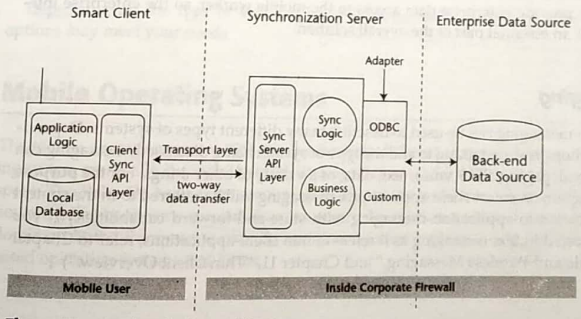

Figure 7.2 Synchronization architecture.

If you are developing an in-house solution, you may choose to create a proprietary data stream format, possibly basing it on XML, or you may decide to use an industry-specified format such as SyncML. The same options are available in commercial synchronization software. In both cases, a stream of data is being transferred that both the client and the server know how to decipher. When the server receives data, it needs to know how to process it. This capability involves understanding the data layout of the enterprise data source. In the most basic form, the synchronization logic will simply take the updates and apply them to the data store, without considering whether they should be applied or not. In more complex cases, the synchronization logic will perform a variety of duties, including conflict detection and resolution.

The integration with the back-end data source is often a simple process, using Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC) drivers for direct database access. At the same time, integration can be difficult, requiring complex logic to interface with more complicated systems such as Enterprise Resource Planning (ERP) systems. In either case, providing the ability to access enterprise data is critical for an enterprise synchronization solution. Rarely does a company use only one enterprise data storage mechanism, so, ideally, the synchronization server will be flexible enough to integrate with a variety of back-end sources.

Enterprise Data Source

The final part of the smart client solution is the enterprise data itself. This data will vary in formats ranging from enterprise databases from vendors, such as Oracle, Sybase, Microsoft, or IBM, to flat-file systems and everything in between. For the more common storage systems, you should be able to find a driver or adapter that will provide an integration layer for your synchronization server. If you are using something that is uncommon or something developed in-house, you will most likely have to develop this integration layer yourself.

Keep in mind that one of the most common reasons for implementing a mobile solution is to provide enterprise data access to the mobile worker, so the enterprise integration is an essential part of the overall solution.

Messaging

The term messaging can be used to describe many different types of systems. Depending on whom you are talking to and the type of system being discussed, messaging can mean email, paging, SMS, voice, text, data, or a variety of other things. For the purpose of this section on smart client applications, messaging will be referred to in the context of application-to-application messaging with store-and-forward capabilities. (If you are interested in text messaging as it refers to thin client applications, refer to Chapter 5, "Mobile and Wireless Messaging," and Chapter 11, "Thin Client Overview.")

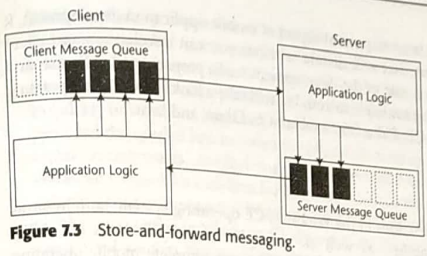

Figure 7.3 Store-and-forward messaging.

As seen in Figure 7.3, the basic form of smart client messaging is composed of a message queue on the client and another queue on the server. These message queues allow you to communicate with messaging system without a wireless network connection. When a message is sent from the server, it is simply placed in the queue that corresponds to a particular client alias. When the client matching that alias connects to the server, the messages in the server-side queue are sent to the client. In the same way, when a client sends a message to the server, it goes into a queue on the client device. When the client connects to the server, the messages from the client queue are sent to the server. This is commonly called store-and-forward messaging, since messages are stored when a connection is not available and forwarded when the connection becomes available.

When building smart client applications you have three options:

- Create a client that uses a persistent data store and synchronization.
- Create a client that uses store-and-forward messaging.
- Create a client that combines both a persistent data store and store-and-forward messaging.

Depending on the type of solution being implemented, any one of these three options may meet your needs.

Mobile Operating Systems

The mobile operating system (OS) market is vastly different from the desktop operating system market. In the mobile world, there is no clear leader in the operating system space. The Palm OS has a large amount of market share in the consumer space; Windows CE has been very successful in the enterprise market; and the Symbian OS is dominant in the European market. In addition, a following is growing for both Java-based operating systems and Linux.

The operating system is an important aspect of mobile application development. It is one of the main factors that will dictate whether you can develop a smart client application. It also plays a role in the development tools, prepackaged software, and device support that will be available to you. We will take a look at five main operating environments: Windows CE, Palm OS, Symbian OS, Linux, and Java.

Windows CE

Microsoft has made great strides in the Windows CE operating system, both in terms of functionality and usability, as well as in market share. Windows CE .NET, Microsoft's current operating system, delivers the most complete mobile operating system available. Windows CE enables multitasking, allowing the user to work on one application while another is executing in the background. For example, you could be working on PocketWord while synchronizing data in the background. Pocket PC 2002 devices use customized versions of Windows CE and look as though they will be the market leader for the enterprise market, and possibly the consumer market as well. But the situation was not always so positive for Microsoft.

Back in 1995, when Microsoft released the first version of Windows CE, it had very limited success. Microsoft had just released Windows 95 and had a near monopoly in the desktop operating system market. The Windows CE operating system was designed for Handheld PC devices, which, size-wise, are in between PDAs and laptops. Even though many device manufacturers supported Windows CE, it was given a very cool reception by both the consumer and enterprise markets. It did not look as though Microsoft was going to reproduce its desktop OS success with Windows CE.

Unwilling to concede, in 1998, Microsoft released Windows CE 2.x for the PDA market. At this time, Palm dominated the palm-sized device market. Nevertheless, Microsoft was confident that Windows CE was going to be the "PalmPilot killer." This proved not to be the case. Once again, Windows CE was given a cool reception. Many consumers found the user interface too cluttered and complex. Microsoft found itself in an unfamiliar position, trailing both Palm OS and Symbian OS (at the time called EPOC) in popularity. It had to come up with something better.

Finally, in 2000, Microsoft released a version of Windows CE that garnered some attention. Windows CE 3.0 was released when the PDA market was becoming more attractive to enterprises considering building line-of-business applications. For these applications, corporations were looking for an operating system that would allow them to run existing Windows-based software and to create sophisticated data-driven applications. Windows CE fit that description. Even though the popularity of the Palm OS was at an all-time high, Microsoft made some early penetration into the enterprise market during the year. By 2001, Microsoft found itself in a more familiar position, often being the OS of choice for new business application development.

Why did Windows CE 3.0 succeed? For many reasons. It comes with many familiar applications, including PocketWord, PocketExcel, PocketOutlook, and Pocket Internet Explorer. In addition to its office products, Windows CE also features built-in multimedia capabilities for both audio and video. These capabilities make it easy for existing

Windows desktop users to move to the Windows CE operating system. They can maintain the rich level of functionality they are used to, only on a smaller device.

On the development side, Windows CE 3.0 utilizes a scaled-down version of the commonly used Win32 application programming interface (API), allowing developers to quickly refit existing Windows applications for Windows CE 3.0-based devices. This provided the needed link between the Microsoft desktop operating systems and their mobile counterparts. Another major improvement from Windows CE 2.x to 3.0 is enhanced support for real-time functionality made to the OS kernel. This dramatically enhances the performance of Windows CE 3.0 over Windows CE 2.x.

The progression of Windows CE does not end there. At the end of 2001, Microsoft announced another version of the Windows CE family: Windows CE .NET 4.0. This version fits into Microsoft's overall .NET strategy by allowing developers to use the full suite of Microsoft tools, including Visual Basic and Visual C++, for development, as well as updated versions of the Microsoft suite of office products and Internet Explorer. Windows CE .NET also added built-in wireless capabilities, including broad support for WANs, LANs (including 802.11b), and PANs (including Bluetooth).

Then, in the fall of 2002, Windows CE .NET version 4.1 was released. This version added support for IPv6 (the latest version of Internet Protocol), as well as integrated speech recognition. Additionally, devices using Windows CE .NET 4.1 support popular PC file formats including Microsoft Word, Excel, PowerPoint, and Adobe Acrobat. Internet Explorer performance has also been increased by as much as 15 percent from Windows CE .NET 4.0 and 60 percent over Windows CE 3.0. It is anticipated that in 2003, most of the new Pocket PC and Handheld PC devices will take advantage of the Windows CE .NET operating system. Table 7.1 compares the various Windows CE version capabilities.

Table 7.1 Comparison of Windows CE Versions

Version/ Features	Year of Release	Processors Supported	Internet Browser	Wireless Capabilities	Multimedia Support
CE 2.12	1998	ARM, MIPS, PowerPC, SHx, x86	Based on Internet Explorer 4.0	IrDA	No (audio only)
CE 3.0	2000	ARM, MIPS, PowerPC, SHx, x86	Based on Internet Explorer 4.0	IrDA	Yes
CE .NET	2002	X-Scale, ARM,	Based on Internet Explorer 5.5	Bluetooth, 802.11x, Media Sense, OBEX	Yes

Confusion often exists between Windows CE and Pocket PC. To help distinguish between the two, the following extract is reprinted from Microsoft's Web site (www.microsoft.com/windows/ Embedded/ce/evaluation/faq/default.asp):

Pocket PC does not replace Windows CE at all: The former is the hardware device, the latter is the operating system. However, Pocket PC does use Windows CE 3.0 as its underlying operating system. Pocket PC uses a customized version of the Windows CE 3.0 operating system, built by Microsoft and used specifically in Personal Digital Assistants (PDAs) like the Compaq iPaq and the Hewlett-Packard Jornada. While this customized version is only used in PDA-type devices, Windows CE 3.0 can be used in a wide variety of devices including industrial automation, Internet access devices, Web terminals, kiosks, consumer electronics, or retail and point-of-sale devices.

This extract is somewhat dated as the latest versions of PocketPC are now based on Windows CE .NET 4.1, and will soon use Windows CE .NET 4.2.

Even though Windows CE is not the leading mobile operating system, it is definitely a strong platform on which you can build very robust, sophisticated smart client applications. Microsoft has spent the past several years enhancing the early versions of Windows CE to bring consumers the latest Windows CE .NET operating system, which should remain a strong player, if not become the leading mobile operating system.

Palm OS

Palm OS has experienced tremendous success in the PDA market. In the early days of PDAs, Palm established itself as the market leader, capturing nearly 75 percent of the worldwide mobile operating system market by early 2000 (IDC, June 2000). Since then, it has continued to be a dominant player in the market. Lately, however, it has faced some difficult times.

Palm has not had the same level of success in the enterprise market for new application development as it did with consumers. In the early days of PDA application development, many organizations chose Palm OS as a deployment platform because the devices were readily available and many employees were already familiar with them. This situation resulted in a wide range of applications being developed for Palm OS. As the requirements for business applications became more complex, however, Palm OS was found to be unsuitable for many of these tasks. For example, prior to the release of version 5.0, Palm OS had single-tasking capability only; meaning users could perform only one task at a time on the device. Thus, if they wanted to download a file, they had to wait for the download to complete before they could move on to other tasks. Many corporations found this to be a significant limitation. When, with version 5.0, Palm introduced some multi-tasking capabilities to the platform, it improved the usefulness of Palm OS for more complex applications. More enhancements for enterprise applications are also expected in Palm OS 6.0.

It's important to point out that Palm hardware devices and Palm OS used to be considered one and the same. This changed when Palm started licensing Palm OS to third-party device manufacturers. The first licensee was Handspring; many others followed suit, including IBM, Symbol, Kyocera, and Sony. Palm's goal by separating the hardware and software divisions was to maintain the widespread usage of Palm OS while avoiding the same problems Apple confronted as the sole manufacturer of both the

hardware and the operating system software for the Macintosh. To date, it looks as though Palm is on the road to success in this regard. Handspring, for example, in 2001, released a device on the market called Treo, which incorporates a phone with wireless modem, Palm OS, email, wireless Internet access, SMS text messaging, and a full thumb keyboard, all in one device. Samsung and Kyocera have similar offerings.

Palm OS 5.0 also includes new features that make it more suitable for enterprise customers. The most significant of these is the move to the ARM series of processors. The result of this move is a significant increase in the overall performance of Palm OS, allowing for the development of more advanced, feature-rich, enterprise-level applications. At the same time, version 5.0 also added the aforementioned multitasking capabilities, additional security for encrypting private data, new expansion slot capabilities, and new wireless capabilities for easier access to the Internet and email systems; it also enhanced the level of color support.

Moreover, the new features of the Palm i705, including wireless support for always-on wireless networks, Bluetooth, and clip-on modems, will make the Palm OS a formidable challenger to Windows CE in the enterprise space. The only concern is that these new capabilities may have come too late; Windows CE had two years to become established before Palm OS delivered these new enterprise-level capabilities. To further distinguish their enterprise offerings, in the fall of 2002, Palm introduced two new product families: Tungsten and Zire. The Tungsten family of products is targeted at the enterprise market, providing powerful solutions for mobile professionals and enterprise work forces; the Zire family is focused on the consumer market, providing affordable options for individuals to organize their schedules and contacts. These new offerings should help Palm regain the momentum it had in 2000/2001.

On the technical side, Palm OS has three main components, listed here (Figure 7.4 shows how these components relate to one another):

- Reference hardware design, consisting of device hardware, third-party hardware, and the hardware abstraction layer.
- Palm OS with embedded system and third-party libraries.
- Application software, featuring the applications included with Palm OS, such as the HotSync conduit and third-party applications.

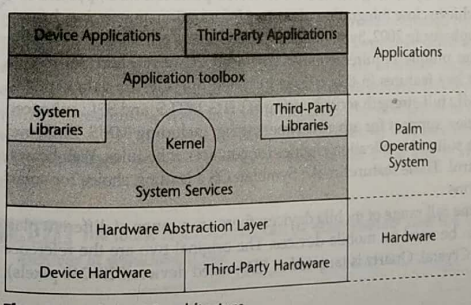

Figure 7.4 Palm OS architecture.

Today, Palm OS has a bright future in the PDA marketplace. Even though other mobile operating systems are closing in on its market lead, it will be some time before another operating system overtakes Palm OS in market share. With the release of version 5.0, as well as innovative new devices based on the Palm OS, the market will remain split between the Palm OS and Windows CE in North America, and among the Palm OS, Windows CE, and the Symbian OS in other parts of the world.

Symbian OS

The Symbian gained its popularity in the mobile phone sector; in fact, Symbian is an operating system developed exclusively for mobile devices. As phone capabilities increased, a more functional operating system was required. Proprietary operating systems from the device manufacturers were no longer practical, so a group of the largest cell phone manufacturers, including Nokia, Ericsson, Motorola, and Matsushita, together with Psion, formed a joint venture known as Symbian. The goal of this partnership was to create a standard operating system for smart phones and PDAs based on Psion's EPOC operating system.

NOTE **Symbian OS is the name assigned to the latest versions of the EPOC operating system. Versions prior to 6.0 are called EPOC, while later versions are called Symbian OS.**

In June 1999, EPOC version 5 started shipping. It contained support for devices based on a 640×240 screen resolution, with pen and keyboard capabilities. Since then, versions 6.x and 7.x of the Symbian OS have been released by Symbian. These new implementations, described in turn in the subsequent paragraphs, build upon the core EPOC operating system, while adding new capabilities required for next-generation mobile devices.

Symbian OS version 6.0, which made some radical improvements over the previous version, was released in 2000. The design goal was to bring together various forms of communication protocols, including TCP/IP, WAP, GSM, Bluetooth, IrDA, as well as serial connections, while allowing applications to be presented using C++, Java, WAP, and other Web protocols. By linking the communications protocols and applications, Symbian hoped to achieve close integration of contact information, messaging, browsing, and wireless telephony. In 2002, Symbian OS version 7.0 was released. This version was designed for the unique requirements of advanced 2G, 2.5G, and 3G mobile phones. Some of the key features in this release included over-the-air data synchronization using SyncML; full-strength security using HTTPS, WTLS, and SSL; enhanced multimedia capabilities; support for advanced messaging, including MMS, EMS, and SMS; and a complete suite of application engines for contacts, schedules, Web browsing, and system control. These features make Symbian OS a leading choice for smart phone implementations.

In order to target the full range of mobile devices, there are a range of different platform models that can be used on mobile devices. The original two are the reference designs: Quartz and Crystal. Quartz is targeted at PDA-sized devices(320×240 pixels),

while Crystal is targeted for communicator type devices (640×200 pixels). More recently with the release of Symbian OS 7.0, two other user interfaces called called UIQ and Series 60 were introduced. UIQ provides a customizable pen-based user interface for media-rich mobile phones. It was designed to take advantage of the large, touch-sensitive color displays found on 2.5G and 3G handsets. UIQ is targeted for screen sizes ranging from 208×320 to 240×320 pixels, the standard size of most PDAs. The first device that takes advantage of UIQ is the Sony Ericsson P800. The Series 60 platform is developed and licensed by Nokia to a variety of phone manufacturers including Siemens, Panasonic, and Samsung. It is designed for mobile phone users with easy-to-use, one-hand operated handsets that have high-quality color displays, rich communications, and enhanced applications.With these innovative new designs, Symbian OS is making a strong move into the smart phone and PDA markets. To make the job easier for device manufacturers, UIQ and Series 60 come with several built-in applications, including email, SMS and MMS messaging, integrated contact lists and calendars, and high-quality Web page delivery. In addition to providing the operating system platform, Symbian also provides a development kit for developers who are building applications for the Symbian OS. This kit offers a wide range of application development choices. For on-device applications, these include EPOC C++, an efficient and purely object-oriented language that provides full access to all of the platform APIs; and Java, which can run on Symbian's Java runtime, which implements the PersonalJava 3.0 JVM and the JavaPhone 1.0 profiles.

For server-based applications delivered to the device, Symbian OS implements WAP 1.2.1 and HTML 4.01 with full frames, HTTPS, and embedded Java applet support. The Symbian OS also supports GSM, GPRS, EDGE, and CDMA networks and full internationalization support with Unicode and international locales.

Historically, the Symbian OS has been dominant in the European markets, with little exposure in North America. This started to change in March 2002, with Nokia's release of the Communicator 9290 device in North America. It is expected that other vendors will also release Symbian OS devices outside of Europe, making Symbian OS a potential challenger to both Palm OS and Windows CE for enterprise application development.

Linux

Linux is a free UNIX operating system that was initially created by a developer named Linus Torvalds. It is an open source operating system developed under the GNU General Public License, making the source code freely available to anyone who is interested. Current versions of Linux are being implemented by developers around the world who submit updates to the original code. The development effort started in 1991 with version 0.02. In 1994, version 1.0 of the kernel was released. The current full-featured release, version 2.4, was released in late 2002, and is still constantly being updated.

NOTE **The latest Linux kernels can be downloaded from the Linux Kernel Archives at www.kernel.org.**

When it comes to mobile devices, many flavors of Linux are available. Most of the device manufacturers supporting Linux will have their own version of the OS. In addition, many commercially available versions of embedded Linux are available. The major implementations of the Linux OS seem to be based on the handhelds.org kernel. Three companies offer Linux operating systems that are compatible with the Compaq iPaq: TransVirtual PocketLinux, Century Software MicroWindows, and TrollTech. The iPaq is not the only device that has Linux support though; other versions of the Linux OS are available for devices from Casio, Hewlett-Packard, Sony, and Sharp.

Note, however, that many Linux implementations are provided by the developer community and are not officially supported by the device manufacturers. This situation is starting to change. In 2002, Sharp released the Sharp Zaurus device, which is based on the Lineo Embedix Linux platform. Several other smaller manufacturers, including Royal (www.royal.com) and Infomart (www.infomart.info), announced plans for Linux-based PDAs in 2002, although none were shipping commercially at the time this book was written.

NOTE **The fact that Linux is freely available does not mean that companies cannot charge for the enhancements they make. Companies such as RedHat have built an entire business around the Linux operating system.**

One of the drawbacks of using Linux is the range of applications available. With a small market share and no official support on many devices, it is a challenge to find the applications and support that is required in the enterprise space. This situation will surely change for the better with the release of more commercial devices.

While Linux is growing in popularity on the server market, it has yet to be shown whether it will have a lasting impact on the mobile operating system market. Judging by the number of delayed Linux PDA offerings, it looks as though Linux will fill a niche spot in the mobile operating system market for Linux enthusiasts.

Java and J2ME

When the Java programming language was initially developed by Sun Microsystems in the early 1990s, it was aimed at the embedded device market. The large number of devices with different chipsets made programming difficult with Assembler or C. The idea behind Java was to design a language that could be ported to different architectures and operating systems without reprogramming. It seemed like an ideal fit for the diverse device market, but device manufacturers were not interested. The result was a language that could run on multiple chipsets with multiple operating systems but that did not generate any interest in its target market.

Fortunately, Sun Microsystems did not give up. With the growing popularity of the World Wide Web, the need for such a language resurfaced. At the time, Internet content consisted of static HTML pages that did not generate much viewer excitement. Missing was dynamic content. But how could it be programmed? What was needed was a language that could be run on a variety of operating systems based on different hardware architectures—seemingly, a perfect fit for Java. Using the Java platform,

developers could create little Java applications called *applets* that could run inside of a Web browser and provide a rich, dynamic user interface for Web applications. The version of the Java platform that provides this interface is the Java 2 Platform Standard Edition (J2SE). Java applets failed to catch on as many expected. The general feeling was that client-side Java applications were too slow and too large to download.

Around that time, Sun Microsystems was also working on a server-side API for Java, now called the Java 2 Platform Enterprise Edition (J2EE). Rather than developing server-side Web applications using proprietary software, developers can use the J2EE specification to develop applications that can be ported to a variety of servers across a variety of operating systems. The Java platform finally found a large audience of companies and developers ready to take advantage of the capabilities offered by the language. As you are probably aware, most of today's application server vendors support the J2EE specification, and many wireless server-side applications are based on J2EE as well.

With the success of the J2EE specification, in 2000, Sun Microsystems decided to give Java another try on mobile devices. This time the company implemented another version of Java called the Java 2 Platform, Micro Edition (J2ME). J2ME targets the embedded and consumer device space, which ranges from Java Smart Cards to set-top boxes and smart appliances. Java smart phones and PDAs are included in that range. J2ME applications maintain the same core features of J2SE and J2EE: code portability, cross-platform support, along with the Java programming language. In addition, J2ME applications are upwardly scalable, to work with J2SE and J2EE platforms.

That said, it's important to point out that the J2ME architecture is somewhat different from J2SE and J2EE as it incorporates a few new concepts. J2ME is based around configurations and profiles. A configuration consists of a Java Virtual Machine (JVM), core libraries, standard classes, and APIs. Two configurations exist for J2ME today: Connected Limited Device Configuration (CLDC) and Connected Device Configuration (CDC). Each is designed for a set of devices with varying capabilities. The configuration provides the lowest common denominator set of classes and the building blocks on which profiles are created. A profile sits on top of a configuration and provides a complete set of APIs for a narrower set of devices. The combination of a configuration and a profile results in a complete J2ME platform.

The CLDC is based on the K Virtual Machine (KVM) and associated libraries. It is targeted at highly constrained devices with either 16- or 32-bit CPUs and less than 512 KB of total memory available for applications. The current profiles for CLDC include the Mobile Information Device Profile (MIDP) and the PDA Profile. Devices that fit into this category include smart phones, two-way pagers, and low-end PDAs.

For devices with more robust resources, the CDC is the configuration of choice. These devices typically run on 32-bit processors and have over 2 MB of memory available for applications. The base profile for the CDC is the Foundation Profile. On top of the Foundation Profile is the Personal Basis Profile and the Personal Profile. The relationship between these profiles is somewhat unique. The Personal Basis Profile actually includes the Foundation Profile and is a subset of the Personal Profile. The Personal Profile provides a similar level of client support to J2SE, but has a smaller footprint. The relationship between the J2ME configurations and profiles is shown in Figure 7.5.

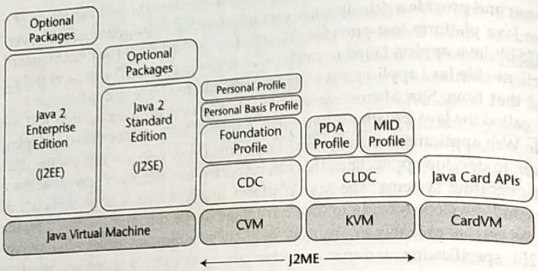

Figure 7.5 J2ME architecture.

Java is listed in the operating system section because you can create smart client applications using Java technology and deploy them to any operating system that has a JVM, which includes all the OSes listed previously. This means that rather than developing a native application that will work only on the targeted operating system, you can develop your application using Java and be able to target a variety of operating systems without having to rewrite the application. It sounds like the ideal solution. However, it is important to keep in mind that J2ME still suffers from some of the same problems experienced over five years ago with the Java on desktop clients; specifically, performance is not always adequate and the portability is not 100 percent. For most applications, extensive testing is required for each platform to which the application will be deployed.

As device manufacturers continue to add more capabilities to handheld devices, thus increasing their performance, it is expected that Java technology will become a viable option for creating sophisticated, data-driven smart client applications.

Proprietary Operating Systems

Before mobile operating systems were readily available, many manufacturers relied on proprietary operating systems for their devices. A proprietary operating system is defined as one that is only used, and available for use, by the device manufacturers themselves. Some examples include the RIM OS, used on the Research In Motion BlackBerry devices, and the Wysdom OS that Motorola used in its TimePort two-way pagers.

The use of proprietary operating systems is becoming quite rare, as it prevents large-scale development. For this reason, both RIM and Motorola have adopted J2ME as the development platform of choice for their most recent devices.

Summary

Smart client applications are an effective way to extend information to mobile devices in an always-available fashion. By incorporating persistent data storage and enterprise synchronization, the application users are ensured of having access to their data whenever and wherever it is required. The deciding factor on whether a smart client application can be developed for a particular device is the mobile operating system. Windows CE, Palm OS, Symbian OS, Linux, and Java/J2ME are the main operating systems that are well suited for smart client applications.

In the next chapter we will continue our exploration of the smart client architecture by investigating the smart client development process.

Helpful Links

The following are some useful links to find out more information on the topics discussed in this chapter.

Windows CE	www.microsoft.com/mobile/; www.microsoft.com/windows/embedded/ce.net/default.asp
Palm OS	www.palmsource.com/
Symbian OS	www.symbian.com/
J2ME information	http://java.sun.com/j2me/index.html; http://wireless.java.sun.com
Linux handheld information	www.handhelds.org

CHAPTER 8

Smart Client Development

Launching a mobile development project can be a challenging task. In addition to the user interface concerns that are always present on mobile devices, you have to worry about enterprise integration, connectivity issues, device constraints, and application deployment and management issues. All of these challenges have to be met using technology that is still in its infancy. It seems like a daunting task, but with proper research and project management these challenges can be overcome, resulting in a very rewarding solution.

The goal of this chapter is to prepare you for some of the technical challenges you will encounter while developing mobile applications, as well as to give you some pointers on how to get started. In this chapter, we will step through each part of the development process, investigating the technology that is available to help you build your mobile solutions. Then we will discuss the pros and cons of developing native versus Java applications.

NOTE **It is beyond the scope of this chapter to discuss general software design and development cycles. Many good books are available dedicated to this subject. Two that I recommend are *Code Complete* by Steve C. McConnel and *Design Patterns* by Erich Gamma, et al.**

The Development Process

Developing mobile and wireless applications will be a new experience for most business application developers, who usually have not had experience working with limited memory and screen sizes. Moreover, most developers are used to working over reliable networks, commonly TCP/IP or HTTP, and to designing, developing, and testing their applications on the same machine. They have not had to worry about the deployment issues that arise when deploying to several machines, all with different characteristics from the one being used for development. Welcome to the world of mobile application development.

Mobile devices vary in shape and size, thereby adding new complexities for the user interface design and resource management. These devices communicate over a variety of wireless networks, often not even IP-based, let alone TCP/IP or HTTP. Finally, the application development does not take place on the target platform, so emulators are commonly used to help make the development and testing cycle more manageable.

To begin the development process discussion, we are going to take a look at the phases involved in designing a successful mobile solution, as depicted in Figure 8.1.

Needs Analysis Phase

A wide variety of mobile solutions are on the market, including voice-based solutions, wireless Internet applications, smart client applications, and messaging-based applications. Which type is best suited for your current project? The answer to that question must be determined during the needs analysis phase of your mobile application development process.

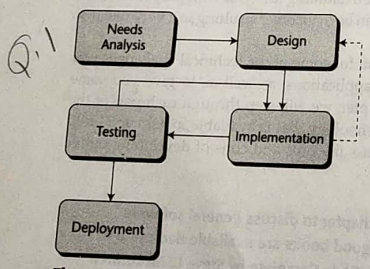

Figure 8.1 Smart client development cycle.

Questions to Ask When Researching User Requirements

While you are researching the user requirements, the following are some questions you may want to ask yourself:

- Who are the end users of this application?
- What is their technical skill level?
- What is the overall goal of this application?
- What data integration is required? Does the user require data access at all times?
- Does this application require wireless connectivity? If so, what type of wireless access does it require, and in which geographies?
- What are the primary usage scenarios for this application?

Try to answer most, if not all, of these questions before beginning to research the device, operating system, and development platform you are going to use. It is very easy to focus on the device and related operating system before determining the actual needs of the end user. If you fall into this trap, be careful not to let the device determine the features of the application. A complete application analysis will lead you to a solution that will be well received by the end user.

Things to Consider

As we consider the issues inherent to gathering the application requirements, we are going to focus on those related to smart client applications. (In Chapter 12, "Thin Client Development," we will look explicitly at the requirements relevant to wireless Internet applications. In Chapter 4, "Mobile Application Architectures," we gave an overview of smart client and thin client applications and the benefits—and drawbacks—related to each architecture type.) Once we gather information about the following items, we can make an educated decision on the solution that is most appropriate, including the operating system, device, wireless network provider, and development platform.

Application Goals

Across the globe, companies are implementing mobile and wireless applications for a variety of reasons. In some cases, the goal is to increase productivity; in others, it is to reduce costs; while in still others, the reason may be simply that it seemed like the next logical thing to do, regardless of the return on the investment. Whatever the reason, it is important that the final solution satisfy the original purpose of development.

During the needs analysis stage, it is important to determine why the project is being undertaken and to produce a result that is in line with expectations. If, say, the project is being implemented to automate order entry, thereby reducing paperwork, the final application should be well suited for rapid data entry, with integration back into the enterprise system. The technical features being implemented should correspond to a business problem that it will solve. By creating a user interface that allows for rapid data entry, we are solving the order entry problem. By having enterprise integration, we are ensuring that the data entered on the mobile device will flow into the back-end system without requiring additional paperwork. These are just a few examples of how technology can help a corporation meet its business goals as it implements a mobile solution.

End User

The end users of the application play a key role in the application design and rollout. Their level of technical ability determines some of the intricacies that may be required. If the user is not comfortable setting up wireless network connection parameters, such details will need to be automated within the application. If the user does not have experience with pen-based input devices, using a device with an alphanumeric keyboard may be more appropriate. Many mobile applications fail to reach their potential because they were not developed with the end user in mind.

Data Access

Extending corporate data to mobile users is essential to many mobile solutions. If this is the goal of the application being developed, then data integration and client storage options will have to be considered. Other factors to consider are how fresh the data must be on the client, how much data should reside on the client, and how often inputted data should be sent back to the server. All of these items will factor into the decision on the data storage and synchronization systems being used.

Wireless Access

Not all mobile applications require wireless access. In fact, many applications are better off without it since real time data access is rarely a requirement. Updates can be made over wireline connections, forgoing the need for wireless access. Other reasons for avoiding wireless access include: complexity of implementation, unreliability of the networks, cost of deployment, or lack of performance.

That said, many compelling reasons do exist for incorporating wireless access into a mobile solution. For field services applications, access to updated work orders, inventory analysis, or routing information can be critical to the success of the solution. Determining the level of wireless connectivity will affect both the implementation criteria and the cost of any given solution.

Usage Scenarios

Predicting how an application will be used is not easy. Nevertheless, it should be attempted, because having some idea of possible use can prove helpful for the application design phase. If, for example, an application is going to be used constantly and

is critical to the success of the user's job, then battery life of the chosen device must be considered. It would not make sense to choose a device that has a short battery life that requires constant recharging for such an application. On the other hand, if the application is used infrequently but must perform some complex logic, it might be sufficient to choose a device with better processing power but with perhaps a shorter battery life.

The same type of logic applies to the level of wireless connectivity required. If the application will be used in a fixed location, implementing a wireless LAN solution would be more manageable and cost-effective than relying on a wide area wireless network.

Design Phase

After performing the needs analysis, you will have a good idea of the characteristics the solution requires. You will have limited the number of operating systems, devices, and networks to just a handful that will meet your requirements. From these you will need to make a decision as to which solution best meets your needs and addresses your business constraints. When doing this, review your options with an open mind. Try not to gravitate toward a particular solution immediately, as it is difficult to recognize at this point what will be a successful implementation unless you have an unsuccessful solution for comparison.

One way to accomplish this is to review all the solutions that could possibly meet your requirements, without giving any weight to an individual category. So, for example, if an operating system will meet the requirements, but not perhaps as well as another OS, keep it on the list as a potential candidate. The same goes for devices, wireless networks, and development platforms. After you have gone through this process, ideally you will have two or three possible solutions on which you can concentrate.

Now you can start adding weight to the individual categories to help you choose the option that best meets your criteria. Remember to factor in some "soft" factors as well, such as costs, ease of use, and aesthetics, in addition to the core technical requirements. Once you have chosen what appears to be the ideal solution, keep the other options on file, as you may need to come back to them in the future. (Refer back to Chapter 2, "Mobile Devices," if you require a summary of the device market.)

At this point, you can start to focus on the application design, looking at such issues as what data is required on the device, how you are going to integrate to the enterprise systems, and how the user interface should be designed for maximum productivity.

Client Data Access

Enterprise data sources will contain much more data than you can hope to store on the mobile device. This is true for enterprise databases such as Oracle, Sybase, or IBM; for ERP systems such as SAP or PeopleSoft; and for CRM systems such as Siebel. The amount of data stored on the device will depend on the mobile data store solution being used, device performance characteristics, and physical limitations of the device. You may only be able to store between a few hundred kilobytes to a couple megabytes of data on the client.

When designing a smart client application, start by examining the subset of data that is required for the mobile user. This subset can be determined by looking at many factors, often depending on the type of application being developed. The subset of data that is required can have various levels of granularity. You can partition the data by its structure in the enterprise source—taking only specific tables from an enterprise database. Another approach is to base the partition on specific data, such as a userid, geography, or price range. The more layers of partitioning you add, the more specific the data on the device becomes.

Let's look at an example sales force automation (SFA) application. For such a system, the enterprise database may contain customer contact information, order history, product information, inventory levels, and supplier information. We do not need all of this data on the mobile device, so we can take a subset of it. In this case, only the contact information, order history, and inventory level tables provide a benefit on the device, so we can limit the data to those tables. To go even further, we can say that we only want data for a particular sales representative, perhaps based on a userid, to be sent down to the device. The result is that we will have a subset of the enterprise data that contains everything relevant to the mobile user, and nothing that is not.

The data access design goes beyond just evaluating the subset of data that is required. Before you can go much further, you will need to establish how you want to implement the data store on the client. If you are implementing your own solution, you will want to start laying out your data storage and management logic. If you are going to use a commercial solution, you will want to start examining how storage is implemented in it, and work on the related design issues, such as the database schema. (For more information on persistent data storage on the client device, see Chapter 9, "Persistent Data on the Client.")

Enterprise Integration

Enterprise integration is a term used to describe any communication to systems not on the device. It encapsulates integration with enterprise databases, business applications, XML data, Web content, and legacy data, among other things. For the purpose of designing your mobile solution, you will need to determine to which enterprise systems you require access. This should be based upon the data access requirements that you have already set forth in the client data access stage of the design process. There are two levels of integration that you may require: basic integration and complex integration.

Basic levels of enterprise integration include the ability to access enterprise databases using defined communication protocols. These capabilities may include the following:

- Device communication using the standard synchronization software such as HotSync for Palm devices or ActiveSync for Pocket PC devices
- Communication over IP-based networks
- Direct integration with a relational database or flat-file system
- Limited support for transactions

Usually, you can follow a well-documented API for these types of applications. This integration layer either can be developed in-house, or you can use a commercial synchronization solution. In either case, the enterprise integration is much more straightforward than it is for larger and more complex systems.

When you start incorporating other forms of enterprise data and other communication protocols to the solution, a more sophisticated enterprise integration layer is required. The capabilities for this type of system may include the following:

- Support for a variety of mobile clients including laptops, handheld PCs, and PDAs
- Communication over networks that may not be IP-based
- Support for synchronizing multiple users simultaneously to a central back-end data store
- Communicating with systems that do not have well-defined interfaces, often requiring custom adapters
- Synchronization of complex data models
- Support for very large amounts of data with many transactions
- Conflict detection and resolution
- Administration tools to manage the entire process

If your system requires these advanced features, you should look to a commercial solution so you do not consume too many internal resources on one aspect of the overall solution. Purchasing a solution outright will result in significant cost savings, compared to the costs involved with writing, testing, debugging, and maintaining the conduit that is required to support the features listed. Even with the use of commercial software, the design requirements are much more complex than for a basic system. You will still need to worry about security, data partitioning, enterprise system access, and scalability, in addition to the usual network coverage and bandwidth issues. (Data synchronization is covered in depth in Chapter 10, "Enterprise Integration through Synchronization.")

User Interface

The user interface can account for as much as 80 percent of the total code in a mobile solution. When you have one part of the application accounting for such a large portion of the development effort, it has to be designed correctly to avoid costly changes later in the development cycle.

Be consistent with the user interface (UI). Get a feel for how standard applications that come installed on the device work and stick very closely with their UI styles. Most of these applications provide a very basic entry screen that meets the needs of most users. Typically, these UIs make more advanced functionality accessible via user selection of a button or menu item.

A couple of key items to keep in mind when designing on the user interface include screen size and human interaction. Paying attention to these items will make the application effective for the application users and help you avoid changes later in the development cycle. They are explained in the following paragraphs.

Screen Size

One of the most dramatic differences between desktop applications and those developed for mobile devices is the screen "real estate." When targeting mobile applications, you will have a one-half VGA, one-quarter VGA, or even smaller screen to work with. Using the screen to its maximum benefit is crucial for successful applications. There are many different ways to accomplish this goal, depending on the operating system you are using. Windows CE, for example, supports a tab-based interface, allowing easy navigation to multiple forms with the click of a button. Taking advantage of the menu capabilities in Palm OS applications allows more information to be displayed on a small screen.

Human Interaction

Studying human interaction with your application will prove to be invaluable in determining its overall usability. History has shown that people always interact with the system in unexpected ways. There is no way that these ways can be predicted, so testing is critical.

Before you get to the stage where you can study human interaction, there are application-specific requirements that can lead the direction of the user interface. If the application focuses on data input, then the main input screens should be easy to navigate using the input properties of the device. For example, if the device offers keyboard support, make it possible for the user to quickly tab between entry fields, rather than having to use a scroll-wheel to get there. If the only means of input is a stylus with character recognition, then including radio buttons and drop-down lists are effective ways to improve the efficiency of data input.

By focusing on the screen design and the navigation practices of the general user, you will be able to design effective user interfaces on devices of all sizes. Do not forget that spending some extra time testing the application early in the development cycle can save you much more time later on.

Wireless Connectivity

If you determine that you require wireless data access within your application, keep in mind the state of today's wireless networks. If you are planning on using a public network, keep these key points in mind during the application design:

- *Wireless coverage is not guaranteed.* In fact, many areas do not have adequate coverage for data communication. This situation applies even in countries that have excellent overall wireless coverage.
- *Network penetration issues can arise even in areas that do have coverage.* Network penetration can be problematical not only in obvious places like subways and tunnels but also in many corporate buildings.

- *Wireless networks operate over a variety of protocols.* Some of these are not IP-based, so if your application requires IP for data communication, you may have to add an additional IP layer for connectivity. Many software vendors can provide this layer for you, if required.
- *Limit the frequency of wireless data transfer.* Because there are potential problems with the reliability of wireless networks, this consideration will make the application more effective. Having suitable persistent data storage within the application will make it possible to limit the frequency of network connectivity within the application. Infrequent wireless data transfer also has a positive effect on battery life, as wireless communications require more battery power than local access.
- *Limit the amount of data transferred.* This limitation is urged, once again because of the nature of the wireless networks.

Implementation and Testing Phase

The implementation and testing phase of the development cycle is what most people refer to as software development. It is during this phase that programming takes place and where developers start to see the rewards of their efforts. The preparation phases (needs analysis and design) are often overlooked because concrete results are not apparent. This does not mean that those parts of the development cycle should be minimized; it means the opposite. Having a complete design document is the best way to begin the actual application implementation. It will allow you to stay focused on the requirements; you will be spared having to figure out what is required on the fly.

The implementation of the final solution does not happen all at once. It is an iterative process that requires much testing. Figure 8.2 shows the process in flowchart format. The diagram introduces the concept of device emulators, which are programs that simulate the characteristics of a physical device using software. In order to get to a successful final product, there are some important steps you must keep in mind, the first of which is the development of a prototype.

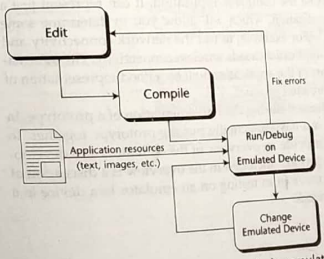

Figure 8.2 Development cycle using device emulators.

CHAPTER 11

Thin Client Overview

By now most likely you have realized that thin client refers to wireless Internet applications. We call this type of application "thin client" because no software is required on the wireless device except for an Internet browser. The browser accepts a markup language, parses it, and displays it to the application user. Any response from the user is sent back to the server, where it can be handled appropriately. For true thin client applications, all of the application logic resides and is executed on the server platform. Hence, the client does not require much processing power or memory to be able to run these types of applications, making them suitable for very small, resource-constrained devices.

Keep in mind that thin client applications are not limited to cell phones. Many devices on the market support thin client applications. Actually, any device that has a microbrowser and a wireless connection is suitable. This includes laptops, PDAs, smart phones, and cell phones, among other more specialized devices.

This chapter gives an overview of the thin client application architecture by highlighting the main components that comprise a successful solution. Following the architecture overview, we take an in-depth look at one of the leading wireless Internet protocols, WAP.

Architecture Overview

As mentioned previously, the thin client architecture is very similar to the architecture of Internet applications. It is based on a three-tier architecture, as depicted in Figure 11.1.

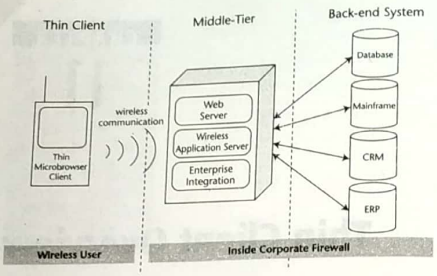

Figure 11.1 Wireless Internet architecture.

The following are the key components of the thin architecture:

Thin client. The client is a microbrowser focused on the presentation of the markup language. It is the part of the application that the end user sees, although it does not execute any of the application logic. The design of the client user interface is crucial to the overall success of the application.

Middle tier. The middle tier is where most of the work is performed. A couple of main components run in this tier: the Web server and the wireless application server. The Web server receives the inbound HTTP requests and routes them to the wireless application server. The application server then takes these requests and executes the appropriate logic. This execution may include session management, content management, as well as integration to the back-end system.

Back-end system. The back-end system is where the data and related services reside. This system may be a relational database, an email server, business applications, an XML content source, or any number of other enterprise systems. In any case, it is important that this data can be reached, so that it is available to the mobile worker.

NOTE ON TERMINOLOGY **The term *wireless application server* refers to the middle-tier server that contains both an application server and a framework designed for wireless content delivery. These two components may be supplied either together or independently.**

Figure 11.1 shows a very simple three-tier architecture. In reality, several other layers need to be addressed: the security layers, including firewalls, as well as the actual location of the wireless networks and related wireless gateways. In the rest of this section, we will cover each part of the architecture in more depth.

Client

The client interface for thin client applications is an Internet browser. In the desktop world, browsers have many advanced capabilities, with sophisticated user interfaces, often supporting client-side scripting languages such as JavaScript or VBScript. Most desktop browsers also have advanced graphics and multimedia support. Many of these features, however, require high bandwidth and powerful client-side processors, a luxury not available on most wireless devices. The client for most wireless applications is not nearly as sophisticated. The browsers have limited screen size, restricting the richness of the user interface. In addition, there is no de facto standard for wireless browsers, meaning that, to reach a broad audience, application providers have to support a variety of browsers and many markup languages.

With these challenges in mind, we will take a look at the various aspects related to the client layer of wireless Internet applications.

User Interface

The user interface may be the most important part of a wireless Internet application. People may disagree, but if users cannot quickly find what they are looking for, they will stop using the application, in which case, everything else becomes irrelevant. The problem is, even the best-designed user interfaces can cause user frustration. Trying to cram useful data and navigation controls onto a small screen such as a cell phone is a daunting task. Even when the screen size is larger, such as on a PDA, the task is not much easier. Much care has to be taken to ensure that the required information is easy to find, without having to navigate through a dozen screens to get it.

One of the difficulties in this endeavor is that the way we navigate desktop Web applications is fundamentally different from how we navigate wireless devices. A wireless device is not commonly used for general information gathering and Web surfing; rather, the wireless device is used for a specific purpose, and the user interface should make accomplishing that purpose as efficient as possible. Therefore, applications have to be targeted for specific tasks, such as placing an order or looking up a defined set of information. Examples in the corporate space include a salesperson looking up price lists or inventory levels, or a field service employee finding his or her next work order. In the consumer space, such a task may involve looking up a stock quote, purchasing a movie ticket, or viewing the latest sport scores. In all these examples, the application is being used for a specific reason; the faster users can accomplish their goal, the more satisfied they will be. The most effective user interfaces do one job and do it well. (In Chapter 12, "Thin Client Development," we will take a closer look at some design patterns that are useful for creating effective wireless Internet user interfaces.)

Browsers and Content Types

Another challenge in creating wireless thin client applications is the variety of markup languages and microbrowsers currently being used. Whereas in the wired world you can address an overwhelming majority of your target audience by creating HTML applications that run in Microsoft Internet Explorer and Netscape Navigator, the same is not so with wireless Internet applications.

Each device typically has its own browser or, in many cases, set of browsers. Each browser understands a particular markup language. The common markup languages include Wireless Markup Language (WML) as defined in WAP 1.x; XHTML, as defined in WAP 2.x; Handheld Device Markup Language (HDML), which is a predecessor to WML; and the subsets of HTML, such as Compact HTML (cHTML). Each of these markup languages has nuances that have to be addressed. Figure 11.2 shows the common microbrowsers on a variety of device classes.

Moreover, each browser handles markup languages differently, meaning you have to test each browser/markup language combination to be sure that the application is displayed correctly. For example, WML content may display correctly on a cell phone running the Openwave browser, but not function properly on a RIM device with a GoAmerica Go.Web WML browser. For such a situation, you have to create content with slight variations to accommodate different client browsers. The good news is that many wireless application servers incorporate this functionality, so it does not have to be handled manually. (In Chapter 13, "Wireless Languages and Content Generation Technologies," we take an in-depth look at the leading wireless markup languages and content-generation techniques for creating device-specific applications.)

Finally, there are voice browsers that use VoiceXML as the markup language. These applications do not display content visually, but rather speak the response over any telephone; the interaction with the system is through voice as well. (Note that because voice-based applications do not use the typical wireless gateways, they are covered separately, in Chapter 15, "Voice Applications with VoiceXML.")

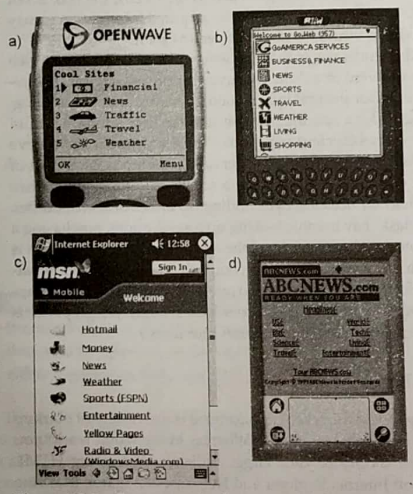

Figure 11.2 Common microbrowsers: (a) Openwave browser, (b) Go.Web browser on RIM 957, (c) Pocket Internet Explorer, (d) Palm Web Clipping.

Wireless Networks

In order for thin client applications to receive data from the server, a wireless network connection is required. This is different from smart client applications, which store data locally, allowing for offline access. With thin client applications, each request goes over a wireless network to a wireless gateway, at which point it is transferred to an HTTP request and forwarded to the Web server/wireless application server.

In most cases, the browser takes care of the communication with the wireless network. This saves developers from having to know the underlying network protocol, allowing them to focus on the application logic. This is a significant advantage, as a large number of wireless network protocols are being used. A complete step-by-step analysis of a wireless request procedure is included in the section *Processing a Wireless Request* later in this chapter.

Middleware

In wireless thin client applications, most of the work is performed using middleware specialized for wireless applications. In this section we take a look at the two main types of servers: wireless application servers and wireless gateways.

Depending on the corporate requirements and resources, these servers can either be hosted by a third-party vendor or be kept on premise. In the most common configuration, the wireless service provider hosts the wireless gateway, and the wireless application server is located within the enterprise.

Wireless Application Servers

Wireless application servers provide the core infrastructure upon which wireless Internet applications are built. They are the engine driving the wireless Internet architecture, providing many of the core feature requirements for this type of application. To meet this rather tall order, wireless applications servers must have a strong server foundation on which the wireless components can be built. For this reason, many vendors have chosen to use J2EE application servers as the underlying technology for their wireless offerings. Using J2EE enables wireless vendors to focus on the core requirements for wireless applications, while leaving the generic server capabilities to the application server vendors.

Initially, many vendors created their own proprietary servers, which were often targeted at a limited number of platforms, thus requiring the developer to use proprietary markup languages and APIs. Realizing that developers preferred to use standards-based technology, most vendors since have ported their frameworks and tools to standard J2EE servers produced by BEA Systems, IBM, Sybase, and Sun Microsystems. In doing so, wireless application server vendors have increased the number of platforms they can support and have reduced the complexity of their offerings. Later in this section, we will investigate the core features of applications servers and how they relate to the J2EE standard and the Microsoft .NET platform. But first we will examine the features provided by commercial wireless server products that are designed for the unique requirements of wireless applications.

The first implementations of many wireless server products were targeted at specific content/device/network combinations. An example would be WML content for the Openwave browser on Nokia devices over a GSM network. While this generated a positive result for the specific combination being targeted, it did not scale well for the multitude of browsers, devices, and networks that were being used as wireless technology became more mainstream. In order to succeed in this rapidly changing marketplace, wireless server technologies had to adapt. This involved creating frameworks that allowed for the development of a variety of wireless applications that could be deployed to a wide range of devices over an assortment of wireless networks.

These frameworks became the basis for the wireless server products on the market today. Most commonly built using J2EE technology, including Java servlets, JSPs, and EJBs, these frameworks provide the core features that are required for wireless application development. These features include device and browser identification, content transformation, voice technology integration, session management, and enterprise data integration. In addition, framework-specific development tools are often available to lower the learning curve associated with using this new technology. Each feature serves a very specific purpose that contributes to the overall success of the wireless application being developed.

A few features on this list generate the most interest. The device and browser identification capabilities are key to providing the appropriate content to the user. By identifying the client device and browser, we can determine the required markup language, the size of the user interface, the types of graphics supported, and the preferred language, among other things. Once we know these factors about the client, the content can be created accordingly, using a variety of techniques and technologies. (In Chapter 13, we take a closer look at each of the leading markup languages, as well as the most common content-generation techniques. In Chapter 14, "Wireless Internet Technology and Vendors," we take a closer look at the features of wireless applications servers and provide summaries of the commercial solutions available.)

Application Server Basics

Here we will take a quick look at the basic services that application servers provide as they apply to wireless Internet applications. (An in-depth look at the capabilities of an application server is beyond the scope of this book.)

The application server is the cornerstone of the distributed Internet architecture. It connects different components of complex applications, while allowing the application to remain modular. It provides the glue to pull together enterprise data, business logic, and the client presentation layer. All of these capabilities are provided in such a way that frees developers from being concerned with the implementation details, allowing them to focus on building the application logic and user interfaces. In general, application servers provide the following three distinct layers:

- Web integration
- Component transaction server
- Enterprise connectivity

Because many commercial wireless servers utilize other application servers as their base platform, we also have to examine the core technology on which they are built. That endeavor involves investigating the core features of the two leading specifications implemented by application server vendors: J2EE and .NET. They both provide the same core functionality, but in significantly different ways; hence, we will review each technology individually.

J2EE

The Java 2 Platform, Enterprise Edition (J2EE) is a specification delivered by Sun Microsystems using the Java Community Process (JCP), which is used to develop and revise Java technology specifications with input from the international community of Java technology experts. In other words, it allows the greater Java community, rather than a single entity, to govern the J2EE specification (and other Java specifications).

The J2EE specification defines a standard platform for building multitier enterprise applications. It builds upon capabilities provided in the Java 2 Platform, Standard Edition (J2SE), such as the core Java libraries, the promise of code portability, and Java Database Connectivity (JDBC). In addition, it defines an enterprise component model in Enterprise JavaBeans (EJBs), Internet capabilities through Java Servlets and JavaServer Pages (JSPs), and XML integration support.Vendors cooperate on developing the specification, then compete on the implementation. This allows a broad range of enterprise systems to use a single standard, allowing for portability between technology vendors and preventing corporations from being tied to a single vendor. Many corporations have desired this type of specification for some time.

NOTE Keep in mind is that J2EE is not a product; it is a standard to which products can be written.

Wireless Internet applications usually take advantage of four main J2EE technologies: EJB, Java servlets, JSPs, and JDBC. They may also use others such as Java Naming and Directory Interface (JNDI), JavaMail, and Java Transaction API (JTA), but usually to a lesser extent, so we will not discuss them here.

NOTE The Java Message Service (JMS) is also an important technology for mobile and wireless applications. It was discussed in more depth in Chapter 5, "Mobile and Wireless Messaging," and is mentioned again later in this chapter in the *Messaging Servers* section.

Let's take a look at each of these four technologies in relation to the functionality they provide to Internet applications. Figure 11.3 provides a logical diagram of how these core layers may be implemented. As you can see, servlets and JSPs act as the main interface between the client applications and the enterprise business logic contained in Enterprise JavaBeans. The EJBs can then communicate with the enterprise data source using JDBC or other adapters.

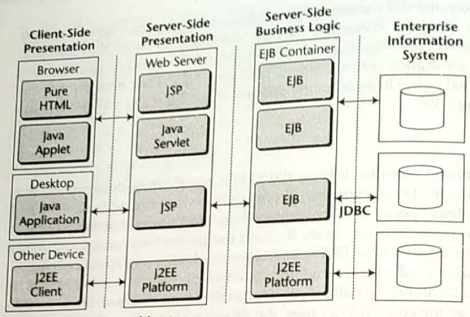

Figure 11.3 J2EE architecture.

Java servlets and JavaServer Pages provide the capability to create dynamic, data-driven content. Java servlets enable developers to easily implement Java server-side code that takes advantage of the other J2EE APIs. They act as an interface between the client application and the enterprise business logic. JSPs provide the same interface, but using a scripting language such as HTML or WML. The JSP specification supports static templates, dynamic markup language generation, and custom tags. Both Java servlets and JavaServer Pages play a prominent role in most wireless application server products.

Business logic is encapsulated in Enterprise JavaBeans components. EJBs make it possible for application developers to create reusable segments of logic, without having to be concerned with the underlying application complexity. The EJB container within the application server handles the various details such as threading, component pooling, and garbage collection. Developers can implement two types of EJB components: Session Beans and Entity Beans. Session Beans typically contain the business logic that maintains workflow and transactions. Entity Beans represent collections of data. They encapsulate data sets and enable operations that can be used to interact with that data. Both component types allow for rapid development and deployment of enterprise logic. Most wireless application servers support the use of EJBs for business logic, but they often do not utilize them for the core wireless capabilities.

The final layer is the enterprise integration layer. If the data resides in a relational database, then JDBC will most likely be the technology used to interact with the data. Most enterprise data vendors have JDBC drivers for their offerings, providing a standard way to manipulate their databases. JDBC offers support for user-defined types, rowset manipulation, connection pooling, and distributed transactions.

If the data does not reside in an enterprise database, then a custom adapter most likely will be required. Depending on the data source, a commercial adapter using EJB technology may already be available. In most cases, these adapters are built to the EJB specification, so they should be able to work in the application server you are using (assuming it is J2EE-compliant). If a commercial adapter is not available, then you may be required to implement an adapter in-house. This undertaking may not be as difficult as it seems. Many enterprise applications have defined APIs that you can use to access the systems, often available for the Java programming language. Additionally, the Java Connector Architecture (JCA) provides an industry standard way to integrate with legacy or nonstandard data sources from J2EE applications.

NOTE **All of the leading J2EE application server vendors are compliant with the J2EE specification. This means that J2EE components such as Java servlets, JSPs, and EJBs written to the appropriate specification should be portable across application server offerings. We say "should," because in reality, testing is usually required on each platform to adjust for vendor-specific nuances.**

Microsoft .NET

Microsoft .NET is the overarching technology around which Microsoft is building its entire platform. This includes its Visual Studio development tools, programming languages, mobile operating system, messaging architecture, and database servers. In the future, most Microsoft products will be part of the .NET initiative. A major component of .NET is XML Web services. Web services are geared to allow applications to communicate and share data over the Internet, independent of the operating systems or programming language being used.

To help highlight the .NET platform, we will compare the J2EE standard and the .NET products. Notice that J2EE is a standard to which products can be developed, whereas .NET is a set of products offered primarily by Microsoft. We will take a look at these products in relation to how they compare with the J2EE standards.

Let's start with the programming language of choice, C#, the base language of the .NET platform (although other languages can be used as well). In fact, Microsoft is promoting the Microsoft Intermediate Language (MSIL), a set of byte codes into which any supported language (such as VB.NET and C#) can be compiled. These byte codes are then interpreted and translated into native executables using the common language runtime. A strong correlation exists between MSIL and the common language runtime and Java byte codes and the Java Runtime Environment (JRE).

Beyond the programming language, the three main aspects of .NET that we will investigate are the Web integration layer, the component transaction layer, and the enterprise integration layer. Figure 11.4 illustrates the main technologies that are of interest to us in the .NET platform. These include (Active Server Pages) ASP.NET, .NET-managed components, and ADO.NET.

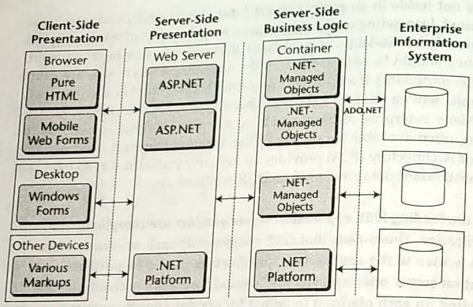

Figure 11.4 .NET platform architecture.

As the name suggests, Active Server Pages .NET are a scripting technology similar to JavaServer Pages. They provide dynamic content to a variety of clients, including desktop clients using Windows forms and browser clients using a variety of markup languages including HTML and WML. As depicted in Figure 11.4, ASPs provide the integration layer between HTTP-based clients and the server-side business logic.

Business logic is encapsulated in .NET-managed components. These are the successors to COM and COM+ components and provide interoperability with them. Essentially, .NET-managed components provide the business logic and data processing layer of the .NET platform, similar to what EJBs provide to J2EE platforms. The business logic for .NET-managed components can be written in a variety of programming languages, since the language will be compiled into the MSIL and executed using the Microsoft common language runtime.

The final layer that we will look at in .NET is the enterprise integration layer. ADO .NET is the technology that provides connectivity to enterprise data sources. It is the latest in a long line of database access technologies that started with Open Database Connectivity (ODBC) many years ago. Since then we have seen OLE DB and ActiveX Data Objects (ADO). One of the main differences of ADO.NET over its predecessors is that it is not as database-centric. ADO.NET is required to support other data sources, as it is the main integration layer between the .NET-managed components and enterprise data. For instances where ADO.NET does not provide adequate enterprise integration, custom adapters can be created using the .NET-managed component architecture. In addition, because .NET is focused on Web services, a variety of XML-based protocols are available for data integration.

Debate is ongoing as to which platform is best: J2EE or .NET. Resolving this debate is not easy, as arriving at an answer involves detailed technical and business analysis. The capabilities of both platforms are comparable, but the ways they are implemented are distinctly different. J2EE is a standard that is widely supported throughout the software industry. There are well over 50 vendors that offer J2EE support on a variety of software and hardware platforms. This results in a wide range of companies providing tools, servers, and other products related to J2EE technology, giving the application developer many options for developing and deploying J2EE applications.

.NET, on the other hand, is a product suite offered by Microsoft on Windows platforms. It provides strong tool and server offerings, but not much flexibility outside of Microsoft's product realm. Companies will either see this as a limitation, since they are locked into Microsoft technology, or as a benefit, since it does provide a tightly integrated solution. Having everything provided by a single vendor does have positive aspects: You will have some assurance that the various technologies will work well with each other and with the operating system used for deployment.

Choosing which platform to use comes down to your specific application and business requirements. Both J2EE platforms and Microsoft .NET are enterprise-class offerings. If you are looking for an open solution, with wide industry and vendor support, J2EE is most likely a better choice for you. If you are primarily using Microsoft technology, and are looking for a well-integrated solution from a single vendor, the .NET product suite should suit your needs well.

Wireless Gateways

Wireless gateways are the link between wireless and wired networks, providing integration between wireless communication protocols and the enterprise. This allows all wireless applications to access enterprise applications and data through the same entry point, independent of the wireless technology being used. Wireless gateways allow users with varying needs (devices, applications, coverage) to select the best network for their situation.

As discussed in Chapter 3, "Wireless Networks," wireless networks use many different protocols for communication, often not based on IP. This means that wireless gateways have to obtain a direct connection to the particular wireless carrier for communication, since it cannot travel over the Internet, which is IP based. With this connection in place, the gateway then transfers the information from the proprietary wireless protocol to an IP-based protocol, such as TCP/IP or HTTP(S) allowing for direct communication to the enterprise system. In this way, the wireless gateway is able to provide connectivity from a wide range of wireless networks to enterprise systems.

Due to the complexity and time required to implement and maintain the connections, wireless gateways are usually hosted by third-party vendors outside corporate firewalls. Here's a concrete example. Let's assume the client is a Go.America browser running on a RIM device in North America. The wireless protocol being used is Mobitex,

which is not IP-based. The wireless application server is listening using HTTP for any incoming requests. The browser on the RIM device is not using HTTP, so it cannot send a request directly to the enterprise wireless server. At some point between the client and the wireless server, the request has to be adapted from the Mobitex protocol to HTTP. This is one of the roles of the wireless gateway. Next assume that a request is made by the user to retrieve content from www.ianywhere.com. Once made, the request is transferred wirelessly to a base station in the user's vicinity. From there, it will be routed using a direct wired connection to the network operation center (NOC), where the wireless gateway resides. The wireless gateway then takes the request, coverts it to HTTP, and forwards it to the appropriate enterprise Web server. The Web server can then output the appropriate WML content and send it back to the wireless gateway, where it is changed back to the Mobitex protocol and sent back to the user. In this scenario, the wireless server and the gateway are located on separate servers in different locations. (Note: Content transformation capabilities are not always handled this way. In many cases, they are executed by the wireless gateway as well.)

Other features offered by the wireless gateway include optimized communication streams, support for IP over non-IP networks, WAP push messaging, and enhanced security. For some of these features, an application must be deployed to the client device, but not always. It depends on the specific feature, as well as the vendor supplying the technology.

One common wireless gateway is a WAP gateway. WAP gateways provide the link between clients using WAP and enterprise systems. Later in this chapter, in the section called *Wireless Application Protocol (WAP) Overview* we will take a closer look at the role of the WAP gateway within the WAP architecture.

Messaging Servers

Recall that in Chapter 7, "Smart Client Overview," we looked at messaging as it pertained to smart client applications. Here we focus on the aspects that are relevant to thin client applications. Included in this category are the Short Message Service (SMS), HDML notifications and alerts, and WAP Push. (Note: Wireless email will not be covered here, as it is discussed within the context of mobile information management in Chapter 16, "Mobile Information Management.")

Let's start by examining the types of messages that thin client applications can receive. As you may recall, the only application on the mobile device is the microbrowser. All of the input and output for wireless Internet applications goes through the browser interface. This is often the case for thin client messaging. There is no persistent data on the device, meaning the messages sent are intended to be viewed by the user, not to be consumed by an application, as in smart client messaging. The most common form of text messaging is SMS, particularly in Europe and Asia, and more often now in North America.

SMS is based on a packet-based architecture that sends data over the control channel. The messages are typically limited to 160 characters. Typical uses include weather reports, stock quotes, calendar reminders, and communication between two users. SMS messaging is controlled using SMS centers, or SMSCs. Each carrier typically has its own SMSC, with its own proprietary interface. In order for server applications to send messages using SMS, they must have interfaces to each of the SMSCs being used.

Implementing this in-house can be a huge burden. For this reason, companies have created a common interface to the various SMSCs to make server integration easier. Using this type of service prevents you from having to get your own connection to each carrier, and from having to interface directly with each carrier's APIs.

The size limitation of SMS messages restricts the content being sent. Text is the only practical messaging format that can be sent in 160 bytes. Because of this limitation, sending more feature-rich messages using this protocol is not possible. The Enhanced Message Service (EMS) changes this. It adds powerful new functionality on top of the SMS standard. With EMS, mobile phone users can add some "life" to SMS text messages, in the form of pictures, animation, and sound. This gives users new ways to exchange pictures and ringtones with other users. One of the great things about EMS is that it is based on SMS standards, meaning that wireless carriers can support EMS with little effort.

In addition to SMS and EMS, WAP 2.x provides a new messaging service called Multimedia Message Service (MMS) that allows for even richer messaging. MMS is aimed at providing a feature-rich messaging solution, in both instant delivery mode (similar to SMS) and with store-and-forward queuing (similar to email). In the future, MMS is expected to become the preferred messaging protocol for mobile users, since there are virtually no limits on what can be sent using it. An MMS message can contain graphics, data, animation, images, audio clips, voice transmissions, and video sequences. That said, it's important to point out that MMS messages are much larger than SMS messages, so MMS will not find widespread use until wireless bandwidth increases.

Outside of the SMS world, browser alerts and notifications are also used for messaging. They provide an alternative means of sending information to wireless Internet users via the microbrowser. In WAP 1.2, the concept of WAP Push was introduced. WAP Push enables a content server to push content to a WAP 1.2-enabled handset by sending a specially formatted XML document to the Push Proxy Gateway, which in turn forwards it to the handset.

Another form of quick-delivery text messaging is instant messaging (IM). Instant messaging from companies such as AOL, Microsoft, and Yahoo! is common on desktop clients, and is quickly moving into the wireless world. One difference between instant messaging and SMS is that IM uses the Internet to communicate. One of the challenges of using wireless IM is that no standard is in place, so IM vendors have their own proprietary interfaces in their IM servers. Though this situation sounds similar to that of SMSCs, with all the challenges that entails, actually it is quite different. With SMSCs you have to integrate with proprietary interfaces, over proprietary networks, whereas with IM, you are communicating over the Internet, meaning you do not have to have a fixed connection to the IM servers. The only level of enterprise integration that is required is communication with the various servers.

NOTE **For the purpose of this chapter, we focus on the text-messaging capabilities in thin client applications. Although more advanced messaging with multimedia is possible, it is not commonly used at this time due to the current state of wireless networks and mobile devices. For a more complete look at messaging as it relates to all forms of mobile and wireless computing, see Chapter 5, "Mobile and Wireless Messaging."**

Processing a Wireless Request

With an understanding of the various parts of the wireless Internet application architecture, we can now examine what is involved in each stage of a wireless Internet request. Figure 11.5 shows each step that a request goes through as it is processed.

The following is a detailed explanation of the steps of the process illustrated in Figure 11.5 (the numbers here correspond with the numbers shown in the illustration):

1. *Establish a wireless session.* If the device is not already connected to a wireless network, a connection has to be made. Most packet-data networks such as Mobitex and GPRS allow devices to be always connected, so this step may not be required. On networks that are not packet-based, such as GSM, the user will have to be authenticated to establish a connection.
2. *Submit a request.* The process starts with the user submitting a request through the client browser. The server uses this request object to obtain information about the user. It contains the URL of the page being requested, as well as other information about the device and browser being used. In addition, it specifies whether it is a GET or a POST request. This request is often encoded in a binary format to reduce the data sent over the bandwidth-limited wireless network. The data is transmitted wirelessly to the network tower, or base station, at which time it is transmitted over a wireline connection to the wireless gateway.
3. *Translate a request.* The wireless gateway decodes the request object from the binary format to a text format and forwards it to the enterprise wireless application server. This request is converted from the wireless protocol (for example, WAP) to HTTP and transported over a wireline IP-based network connection. In order to convert from the wireless protocol to HTTP, the data has to be decrypted. After the conversion, it can once again be encrypted, most commonly using Secure Sockets Layer (SSL). This decryption and reencryption process is often called the WAP gap. Because of this moment of weakened security, many vendors allow the enterprise to host the wireless gateway on-premises, so that data conversion can occur within the corporate firewall, adding additional security.

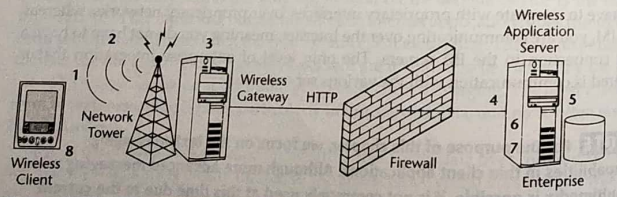

Figure 11.5 Stages of a wireless Internet request.

NOTE **The wireless gateway is not always required. For wireless applications using an HTTP-based client, it is possible to go directly from the client browser over the Internet to the wireless application server. WAP 2.x allows for this type of architecture, removing the requirement for a wireless gateway.**

4. *Receive a request.* The request is received by the wireless application server. In most cases, the wireless interface to the enterprise application is by HTTP (or HTTPS) using Java servlets/JSPs, ASPs, or other Web interfaces. The wireless application server/Web server can receive the request and perform the appropriate transformation based on the client device/browser combination. The server also performs many other tasks as described earlier in this chapter. If the corporation does not want to maintain this server in-house, many vendors offer hosting services: They host the wireless server and connect to the enterprise only for data access.
5. *Identify the wireless client.* Using the HTTP request object, the wireless server can determine which device and microbrowser is making the request. The request object may also be used to determine other information, such as the image types supported, and preferred language. Using this information, the server can ensure that the appropriate content is sent back to the client. At this point, users may be authenticated to the enterprise system to make sure they have access to the data being requested.
6. *Process the request.* Once the user is authenticated, the server can process the request. The URL will specify the information that is required. If it is a static file, the server will simply return the file to the wireless gateway. If the user requires dynamic content, the data can be personalized based on user information. This will involve accessing one or more enterprise data sources to obtain the dynamic data.
7. *Transform content for device.* At this point, the enterprise information has to be transformed to the appropriate format for the client that made the request. This job may involve changing the markup language, the image types, as well as the richness of the user interface. In most cases this is accomplished on the wireless application server located in the enterprise, although in some situations the wireless gateway performs this translation. Many servers on the market use XML as the base data format and transform it using XSL stylesheets for specific devices and browsers.
8. *Return the content.* Once the content is formatted appropriately, it is sent back to the client. It will once again pass through the wireless gateway where it will be encoded to the format that the browser understands. Any information that the server wants to communicate back to the client browser is contained in the response object.

Wireless Application Protocol (WAP) Overview

The Wireless Application Protocol (WAP) is a worldwide standard for the delivery and presentation of wireless information to mobile phones and other wireless devices. The idea behind WAP is simple: simplify the delivery of Internet content to wireless devices by delivering a comprehensive, Internet-based, wireless specification. The WAP Forum released the first version of WAP in 1998. Since then, it has been widely adopted by wireless phone manufacturers, wireless carriers, and application developers worldwide. Many industry analysts estimate that 90 percent of mobile phones sold over the next few years will be WAP-enabled.

The driving force behind WAP is the WAP Forum component of the Open Mobile Alliance. The WAP Forum was founded in 1997 by Ericsson, Motorola, Nokia, and Openwave Systems (the latter known as Unwired Planet at the time) with the goal of making wireless Internet applications more mainstream by delivering a development specification and framework to accelerate the delivery of wireless applications. Since then, more than 300 corporations have joined the forum, making WAP the de facto standard for wireless Internet applications. In June 2002, the WAP Forum, the Location Interoperability Forum, SyncML Initiative, MMS Interoperability Group, and Wireless Village consolidated under the name Open Mobile Alliance to create a governing body that will be at the center of all mobile application standardization work.

The WAP architecture is composed of various protocols and an XML-based markup language called the Wireless Markup Language (WML), which is the successor to the Handheld Device Markup Language (HDML) as defined by Openwave Systems. WAP 2.x contains a new version of WML, commonly referred to as WML2; it is based on the eXtensible HyperText Markup Language (XHTML), signaling part of WAP's move toward using common Internet specifications such as HTTP and TCP/IP.

In the remainder of this section we will take a look at the WAP programming model and the various components that comprise the WAP architecture. Where it is applicable, we will supply information on both the WAP 1.x and 2.x specifications. (More information on the leading markup languages used in wireless Internet applications is provided in Chapter 13.)

WAP Programming Model

The WAP programming model is very similar to the Internet programming model. It typically uses the *pull* approach for requesting content, meaning the client makes the request for content from the server. However, WAP also supports the ability to *push* content from the server to the client using the Wireless Telephony Application Specification (WTA), which provides the ability to access telephony functions on the client device.

Content can be delivered to a wireless device using WAP in two ways: with or without a WAP gateway. Whether a gateway is used depends on the features required and the version of WAP being implemented. WAP 1.x requires the use of a WAP gateway as an intermediary between the client and the wireless application server, as depicted in Figure 11.6. This gateway is responsible for the following:

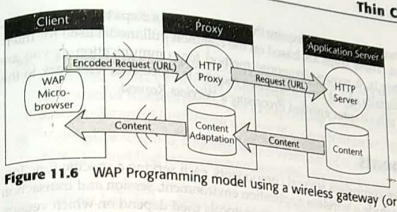

Figure 11.6 WAP Programming model using a wireless gateway (or proxy).

- Translating requests from the WAP protocol to the protocols used over the World Wide Web, such as HTTP and TCP/IP.
- Encoding and decoding regular Web content into compact formats that are more appropriate for wireless communication.
- Allowing use of standard HTTP-based Web servers for the generation and delivery of wireless content. This may involve transforming the content to make it appropriate for wireless consumption.
- Implementing push functionality using WTA.

NOTE **The WAP gateway is often called the *WAP proxy* in the WAP 2.x documents available from the OMA. In this chapter we continute to refer to it as the WAP gateway; just be aware that both terms are used to refer to the same technology.**

When developing WAP 2.x applications, you no longer are required to use a WAP gateway. WAP 2.x allows HTTP communication between the client and the origin server, so there is no need for conversion. This is not to say, however, that a WAP gateway is not beneficial. Using a WAP gateway will allow you to optimize the communication process and facilitate other wireless service features such as location, privacy, and WAP Push. Figure 11.7 shows the WAP programming model without a WAP gateway: Note that removing it makes the wireless Internet application architecture nearly identical to that used for standard Web applications.

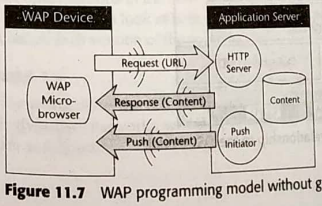

Figure 11.7 WAP programming model without gateway.

Both WAP programming models require the same core set of steps to process a wireless Internet request. These steps are based on the common pull model used for Internet applications; that is, a request/response method for communication. If you are interested in more details about how a wireless request is processed, refer back to the previous section of this chapter entitled *Processing a Wireless Request*.

WAP Components

The WAP architecture comprises several components, each serving a specific function. These components include a wireless application environment, session and transaction support, security, and data transfer. The exact protocols used depend on which version of WAP you are implementing. WAP 2.x is based mainly on common Internet protocols such as HTTP and TCP/IP, while WAP 1.x uses proprietary protocols developed as part of the WAP specification. We will investigate each component and its related function.

To begin, we will look at how WAP conforms to the Open Systems Interconnection (OSI) model as defined by the International Standards Organization (ISO). The OSI model consists of seven distinct layers, six of which are depicted in Figure 11.8 as they relate to the WAP architecture. The physical layer is not shown; it sits below the network layer and defines the physical aspects such as the hardware and the raw bitstream. For each of the other six layers, WAP has a corresponding layer, which will now be described in more depth.

Wireless Application Environment (WAE)

The Wireless Application Environment (WAE) is the application layer of the OSI model. It provides the required elements for interaction between Web applications and wireless clients using a WAP microbrowser. These elements are as follows:

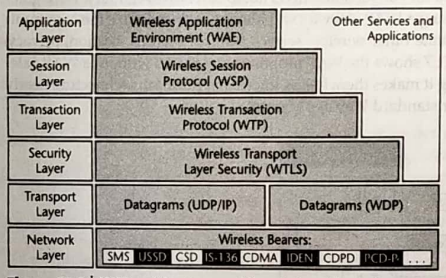

Figure 11.8 WAP architecture and its relationship to the OSI model.

- A specification for a microbrowser that controls the user interface and interprets WML and WMLScript.
- The foundation for the microbrowser in the form of the Wireless Markup Language (WML). WML has been designed to accommodate the unique characteristics of wireless devices, by incorporating a user interface model that is suitable for small form-factor devices that do not have a QWERTY keyboard.
- A complete scripting language called WMLScript that extends the functionality of WML, enabling more capabilities on the client for business and presentation logic.
- Support for other content types such as wireless bitmap images (WBMP), vCard, and vCalendar.

WAP 2.x extends WAE by adding the following elements:

- A new markup language specification called WML2 that is based on XHTML-Basic. Backward compatibility with WML1 has been maintained.
- Support for stylesheets to enhance presentation capabilities. Stylesheet support is based on the Mobile Profile of Cascading Style Sheets (CSS) from the W3C, and supports both inline and external style sheets.

NOTE **WAP 2.x WAE has backward compatibility to WML1. This is accomplished either via built-in support for both languages or by translating WML1 into WML2 using eXtensible Stylesheet Language Transformation (XSLT). The method used depends on the implementation by the device manufacturer.**

WAP Protocol Stack

The WAP protocol stack has undergone significant change from WAP 1.x to WAP 2.x. The basis for the change is the support for Internet Protocols (IPs) when IP connectivity is supported by the mobile device and network. As with other parts of WAP, the WAP 2.x protocol stack is backward-compatible. Support for the legacy WAP 1.x stack has been maintained for non-IP and low-bandwidth IP networks that can benefit from the optimizations in the WAP 1.x protocol stack.

We will take a look at both WAP 1.x and WAP 2.x, with a focus on the technologies used in each version of the specification.

WAP 1.x

The protocols in the WAP 1.x protocol stack have been optimized for low-bandwidth, high-latency networks, which are prevalent in pre-3G wireless networks. The protocols are as follows:

Wireless Session Protocol (WSP). WSP provides capabilities similar to HTTP/1.1 while incorporating features designed for low-bandwidth, high-latency wireless networks such as long-lived sessions and session suspend/resume. This is particularly important, as it makes it possible to suspend a session while not in use, to free up network resources or preserve battery power. The communication from a WAP gateway to the microbrowser client is over WSP.

Wireless Transaction Protocol (WTP). WTP provides a reliable transport mechanism for the WAP datagram service. It offers similar reliability as Transmission Control Protocol/Internet Protocol (TCP/IP), but it removes characteristics that make TCP/IP unsuitable for wireless communication, such as the extra handshakes and additional information for handling out-of-order packets. Since the communication is directly from a handset to a server, this information is not required. The result is that WTP requires less than half of the number of packets of a standard HTTP-TCP/IP request. In addition, using WTP means that a TCP stack is not required on the wireless device, reducing the processing power and memory required.

Wireless Transport Layer Security (WTLS). WTLS is the wireless version of the Transport Security Layer (TLS), which was formerly known as Secure Sockets Layer (SSL). It provides privacy, data integrity, and authentication between the client and the wireless server. Using WTLS, WAP gateways can automatically provide wireless security for Web applications that use TLS. In addition, like the other wireless protocols, WTLS incorporates features designed for wireless networks, such as datagram support, optimized handshakes, and dynamic key refreshing.

Wireless Datagram Protocol (WDP). WDP is a datagram service that brings a common interface to wireless transportation bearers. It can provide this consistent layer by using a set of adapters designed for specific features of these bearers. It supports CDPD, GSM, CDMA, TDMA, SMS, FLEX (a wireless technology developed by Motorola), and Integrated Digital Enhanced Network (iDEN) protocols.

WAP 2.x

One of the main new features in WAP 2.x is the use of Internet protocols in the WAP protocol stack. This change was precipitated by the rollout of 2.5G and 3G networks that provide IP support directly to wireless devices. To accommodate this change, WAP 2.x has the following new protocol layers:

Wireless Profiled HTTP (WP-HTTP). WP-HTTP is a profile of HTTP designed for the wireless environment. It is fully interoperable with HTTP/1.1 and allows the usage of the HTTP request/response model for interaction between the wireless device and the wireless server.

Transport Layer Security (TLS). WAP 2.0 includes a wireless profile of TLS, which allows secure transactions. The TLS profile includes cipher suites, certificate formats, signing algorithms, and the use of session resume, providing robust wireless security. There is also support for TLS tunneling, providing end-to-end security at the transport level. The support for TLS removes the WAP security gap that was present in WAP 1.x.

Wireless Profiled TCP (WP-TCP). WP-TCP is fully interoperable with Internet-based TCP implementations, while being optimized for wi ronments. These optimizations result in lower overhead for the com stream.

NOTE **Wireless devices can support both the WAP 1.x and WAP 2.x pr stacks. In this scenario, they would need to operate independently of e other, since WAP 2.x provides support for both stacks.**

Other WAP 2.x Services

In addition to a new protocol stack, WAP 2.x introduced many other new features and services. These new features expand the capabilities of wireless devices and allow developers to create more useful applications and services. The following is a summary of the features of interest:

WAP Push. WAP Push enables enterprises to initiate the sending of information on the server using a push proxy. This capability was introduced in WAP 1.2, but has been enhanced in WAP 2.x. Applications that require updates based on external information are particularly suited for using WAP Push. Examples include various forms of messaging applications, stock updates, airline departure and arrival updates, and traffic information. Before WAP Push was introduced, the wireless user was required to poll the server for updated information, wasting both time and bandwidth.

User Agent Profile (UAProf). The UAProf enables a server to obtain information about the client making the request. In WAP 2.x, it is based on the Composite Capabilities/Preference Profiles (CC/PP) specification as defined by the W3C. It works by sending information in the request object, allowing wireless servers to adapt the information being sent according to the client device making the request.

External Functionality Interface (EFI). This allows the WAP applications within the WAE to communicate with external applications, enabling other applications to extend the capabilities of WAP applications, similar to plug-ins for desktop browsers.

Wireless Telephony Application (WTA). The WTA allows WAP applications to control various telephony applications, such as making calls, answering calls, putting calls on hold, or forwarding them. It allows WAP WTA-enabled cell phones to have integrated voice and data services.

Persistent storage interface. WAP 2.x introduces a new storage service with a well-defined interface to store data locally on the device. The interface defines ways to organize, access, store, and retrieve data.

Data synchronization. For data synchronization, WAP 2.x has adopted the SyncML solution. As outlined in Chapter 10, "Enterprise Integration through Synchronization," SyncML provides an XML-based protocol for synchronizing data over both WSP and HTTP.

Multimedia Messaging Service (MMS). MMS is the framework for rich-content messaging. Going beyond what is possible for SMS, MMS can be used to transmit multimedia content such as pictures and videos. In addition, it can work with WAP Push and UAProf to send messages adapted specifically for the target client device.

WAP Benefits

The WAP specification is continually changing to meet the growing demands of wireless applications. The majority of wireless carriers and handset manufacturers support WAP and continue to invest in the new capabilities it offers. Over the years WAP has evolved from using proprietary protocols in WAP 1.x to using standard Internet protocols in WAP 2.x, making it more approachable for Web developers. The following are some of the key benefits that WAP provides:

- WAP supports legacy WAP 1.x protocols that encode and optimize content for low-bandwidth, high-latency networks while communicating with the enterprise servers using HTTP.
- WAP supports wireless profiles of Internet protocols for interoperability with Internet applications. This allows WAP clients to communicate with enterprise servers, without requiring a WAP gateway.
- WAP allows end users to access a broad range of content over multiple wireless networks using a common user interface, the WAP browser. Because the WAP specification defines the markup language and microbrowser, users can be assured that wireless content will be suitable for their WAP-enabled device.
- WAP uses XML as the base language for both WML and WML2 (which uses XHTML), making it easy for application developers to learn and build wireless Internet applications. It also makes content transformation easier by incorporating support for XSL stylesheets to transform XML content. Once an application is developed using WML or WML2, any device that is WAP-compliant can access it.
- WAP has support for WTA. This allows applications to communicate with the device and network telephony functions. This permits the development of truly integrated voice and data applications.
- Using UAProf, the information delivered to each device can be highly customized. (Chapter 13 provides more details on how this information can be used to deliver user-specific content.)
- WAP works with all of the main wireless bearers, including CDPD, GSM, CDMA, TDMA, FLEX, and iDEN protocols. This interoperability allows developers to focus on creating their applications, without having to worry about the underlying network that will be used.

At present, all major wireless carriers support the WAP specification. This universal support is expected to continue as WAP evolves, providing a robust, intuitive way to extend Web content to wireless devices.

Summary

Thin client applications do not require any software on the device other than a microbrowser. The architecture is very similar to desktop Web-based applications, with the content generation, business logic, and enterprise integration all located on the server platform. The capabilities of wireless Internet applications are often dictated by the microbrowser that is being used. The features available for an application depend on many factors, including the markup language, graphics support, and display size on the device.

WAP is the de facto standard for building wireless Internet applications. It provides a specification for each layer of the wireless architecture, allowing carriers, device manufacturers, and developers to create applications that will work across many target platforms. In addition to WAP, other technologies can be used to display wireless Internet content, including HTML, cHTML, and HDML.

In the next chapter we will continue to learn how to build thin client applications; there we investigate the application development process for thin client applications.

Helpful Links

Use these links to find out more information on the topics discussed in this chapter:

J2EE information	http://java.sun.com/j2ee
Java Community Process	www.jcp.org
Microsoft .NET	www.microsoft.com/net
Open Mobile Alliance	www.openmobilealliance.org
International Standards Organization	www.iso.ch