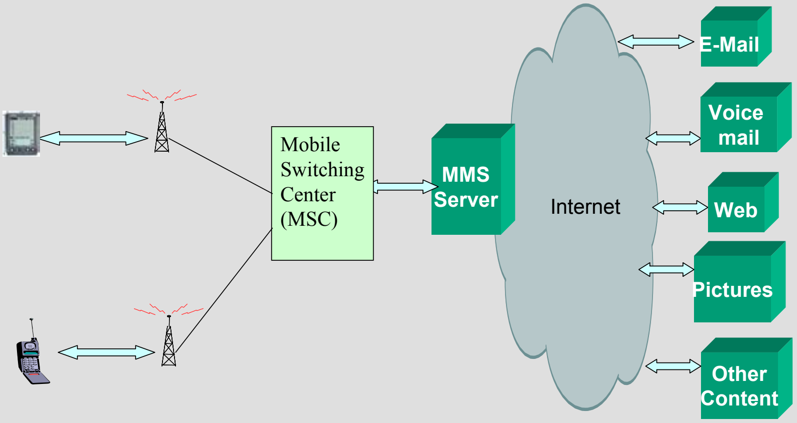
4 Mobile Computing Platforms, Middleware, and Servers

Example: What is Needed to Support Multimedia Messaging Service (MMS)

A US-based company wanted to support MMS, mainly picture messaging, on its cellular phones. The first step was, of course, to identify the technologies needed to support this initiative. The following figure shows an architecture of MMS (we discussed this in Chapter 2). This company subdivided the technologies into two parts: a front end that deals with the mobile devices to the MMS server, and a back end that deals with connecting the MMS server to the various content providers. The following "front-end" technologies, under the general umbrella of "mobile computing platforms" needed for MMS, were the key candidates:

- **Local platform services** that resided on the handsets. These mainly consisted of a Symbian operating system with support for MPEG, JPEG, and digital cameras. In addition, several utilities to backup and synchronize the data on the handset are needed.
- **Middleware services** that interconnect the handset to the MMS server and then deliver the MMS services to the users. These services are needed by MMS phones to exchange messages including still pictures, animations and sounds. In addition, email and email protocol (e.g., POP) support is needed. For most of the MMS services, this company chose WAP (Wireless Application Protocol). MPEG-4, the standard for delivery of video and audio over the Internet, was also chosen.
- **Network transport services** that shuffle the messages over the cellular network. TCP/IP was chosen for this technology. Although MMS is supposed to be supported on 3G networks, this company chose MMS over 2.5G networks.

The back-end technologies, not discussed here in detail, consist of typical Web and Internet-based middleware and network services (see [Umar 2004]).



4.1 Introduction

The mobile information space is packed with thousands of mobile devices that cooperate with other mobile or fixed devices to support mobile computing applications. Platforms to support these applications, called ***mobile computing platforms***, provide interconnectivity services between partners and transfer information in a secure and efficient manner. These platforms, the focus of this chapter, enable the operation and, in many cases, development and deployment, of mobile computing applications such as mobile messaging, mobile commerce, mobile business, and many others discussed in Chapter 2. Figure 4-1 illustrates the position of mobile computing platforms relative to the mobile applications and physical wireless networks, and serves as a general framework for our discussion. As depicted in Figure 4-1, these platforms provide three types of services:

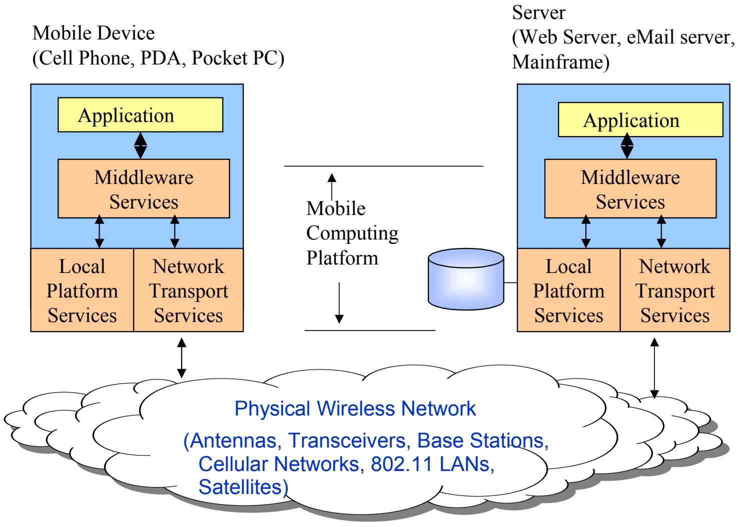

Mobile Device
(Cell Phone, PDA, Pocket PC)
Server
(Web Server, eMail server, Mainframe)
Application
Middleware Services
Local Platform Services
Network Transport Services
Mobile Computing Platform
Application
Middleware Services
Local Platform Services
Network Transport Services
Physical Wireless Network
(Antennas, Transceivers, Base Stations, Cellular Networks, 802.11 LANs, Satellites)

Figure 4-1: Platforms for Mobile Applications

- **Local platform services** that support the applications on the individual mobile devices. These services consist of operating systems (e.g., Symbian OS) needed to run the mobile devices, and also include local system software services such as database managers, transaction managers, and utilities for mobile devices (see Section 4.2).
- **Middleware services** that interconnect mobile users, databases and applications with each other. Simply stated, middleware is *business/industry unaware* software that provides interconnectivity between remotely located applications, databases and users. For example, wireless middleware provides remote access to a corporate database from a mobile phone and may also encrypt and compress the messages for security and performance. We will review the general features of middleware needed to support mobile applications in Section 4.3. An interesting trend at present is to package a variety of middleware services into "wireless gateways" and "mobile application servers" that can support the current and future breed of mobile applications (see Section 4.4). Examples of such packages are WAP (Wireless Application Protocol), i-mode and J2ME (Java2 Micro Edition), and Microsoft Mobile Internet Toolkit (see sections 4.5 and 4.6). The platforms to support voice applications (Voice XML) are discussed in Section 4.7.
- **Network transport services** that are responsible for shuffling the messages over, in this case wireless, networks. These services, mostly handled by TCP/IP, were discussed in the

previous chapter. The underlying physical communication networks are discussed at great length in the next part of this book (Chapters 5 to 10).

Due to the central role played by wireless middleware, the bulk of this chapter is devoted to wireless middleware and its various packages that are commercially available at present. The chapter concludes by discussing a few short examples and case studies (Section 4.8).

Chapter Highlights

- Mobile computing platforms support the development as well as operation of mobile applications such as mobile messaging, mobile commerce, mobile e-business, and others.
- Mobile computing platforms consist of the following:
  - Local platform services that run on the mobile device (clients) and also on the servers
  - Interconnectivity software (the wireless middleware) that interconnects the two
  - Network transport services that transfer message over the wireless networks
- Wireless middleware itself can be subdivided into:
  - Individual services such as compression and content conversion
  - Wireless gateways that combine many individual services together to provide run-time support
  - Mobile application servers that go beyond the gateways and include mobile application development, monitoring, and control features.
- Local services support the individual mobile devices. These services consist of operating systems, database managers, transaction managers, and utilities for mobile devices.
- Middleware services interconnect mobile users, databases and applications with each other. Mobile computing applications need this middleware to smooth over the mobile computing issues as much as possible, so that the same applications can run on wired as well as wireless networks.
- WAP (Wireless Application Protocol) is an open specification for mobile computing platforms. It supports a WAE (Wireless Application Environment) and uses WML (Wireless Markup Language).
- i-mode, developed by NTT DoCoMo, is a popular mobile computing platform in Japan and is currently gaining momentum in the US. It uses special phones capable of voice and packet transmission along with a browser installed. I-mode phones are specialized phones, with larger windows for multimedia, that display content in cHTML (compressed HTML) over packet-switching networks.
- Wireless Java is the general umbrella name under which Sun is supporting its Java platform for developing wireless applications. Java 2 Micro Edition (J2ME) is the core technology in wireless Java for writing applications that run on small hand-held devices.
- The Microsoft Mobile Internet Toolkit (MMIT) is an extension of the Microsoft .NET framework for building Wireless applications. MMIT has two interesting features: it automatically generates code (WML, cHTML, and HTML) for different types of mobile devices, and it is built around .NET and Visual Studio – popular platforms for application development at present.
- BREW, from QualComm, provides application developers with a platform in which to develop their products, and to incorporate the same features as J2ME, including cross-device platform capabilities. BREW currently uses Microsoft .NET Visual Studio.
- VoiceXML is a markup language for voice browsers. It is designed for creating audio dialogs that feature synthesized speech, digitized audio, and recognition of spoken and digitized voice mail.
- Specialized platforms for systems such as wireless sensor networks are evolving.

The Agenda
- Local Services and Wireless Middleware
- WAP, i-mode, J2ME, MMIT, BREW
- Voice XML, Examples and Case Studies

4.2 Local Platform Services for Mobile Devices

Local platform services reside on mobile devices (laptops, pocket PCs, PDAs, cellular phones, and other handsets). These services consist of operating systems, database managers, transaction managers, and utility programs. These platform services are briefly discussed in this section. Although laptops are among the commonly used mobile devices, we will concentrate mainly on the handheld devices because they present unique challenges.

4.2.1 Mobile Devices at a Glance

The current generation of mobile devices are expected to run more and more applications. For example, the cellular phones were initially targeted for voice applications only, but now are expected to support many more applications. In general, many applications will be supported by the mobile devices of today and tomorrow. These include, but are not limited to, telephone calls, email services, short message services, web browsing, multimedia message services, location services, and m-commerce. The main challenge for mobile devices is that each of these applications requires a collection of enabling technologies, shown in the following table.

Table 4-1: Typical Applications and Enabling Technologies

Mobile Device Applications	**Enabling Technologies**
Voice processing	Speech recognition, voice recording, call forwarding
Web browsing	Microbrowsers, content display capabilities
email and short messaging services	email client, email protocol (e.g., POP) support
Multimedia communications	MPEG, JPEG, digital cameras
Location-based services	Speech recognition, GPS, JPEG
M-Commerce	MP3, MPEG, Encryption, transaction management, data management

To support these applications and the underlying technologies, the mobile devices need numerous capabilities. Figure 4-2 shows typical hardware capabilities needed by the modern mobile devices. Of course there is a LCD/Touch screen for the data to be displayed and for the user to input data. At the input, there is usually an antenna or a microphone as a voice recognition device. There is also an output device such as an Infrared (I/R) port, Bluetooth connection or a speaker. The memory and the processor must be small yet powerful enough to run real-time applications. To support multimedia applications, for example, the processor must be able to support digital signal processing algorithms that require high-speed mathematical processing, a large data array, and a fast interrupt response from multiple sources.

The battery must be small but powerful to support sophisticated operations. Due to size limitations, many system components are typically reused and shared among different applications. For example, the larger LCD for the PDA can also be used as a digital camera viewer, display for web browsers, and cell phone screen display. The speaker or headphone can be used as a sound source during the telephone mode or used as the output for MP3 music in the consumer electronics mode. The built-in camera lens can be used during a digital camera mode or used for video mode.

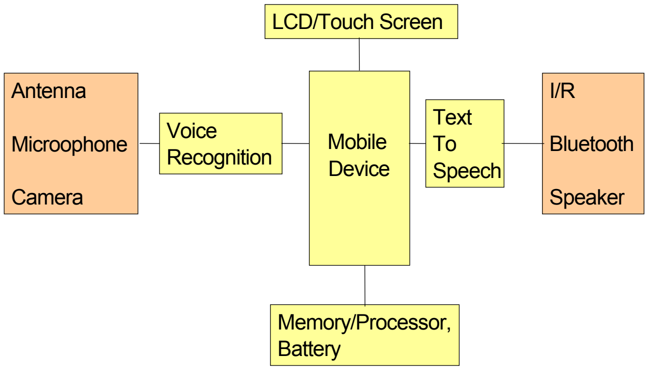


Figure 4-2: Conceptual View of Mobile Device Hardware

Sophisticated operating systems are needed to handle all the devices and to allow the simultaneous use of various applications. For example, a user may want to listen to an MP3 music file on the device while using it to take a picture. Or a user may use the phone feature to talk while using the PDA feature to send the picture she just took with her friends. The combination of usages varies with different users and can be very complex with multiple applications running at the same time. Thus there is a need for a multitasking operating system (OS) with memory protection capability running on the mobile device. To handle multiple applications simultaneously, OSs like Symbian, WinCE, or Linux are typically used. We will review the OSs in Section 4.2.2. The users may need data management capabilities to access and manipulate information on the handset (see Section 4.2.3). In addition, transactional capabilities may be needed for m-commerce applications (see Section 4.2.4) and utilities may be needed to backup and synch information on the handset (see Section 4.2.5). A strong requirement in mobile devices is that a misbehaved application must not destroy data in other applications. The processor and the OS must support memory protection, and be able to support code written in high-level languages (HLL) such as C, C++, and Java.

As the mobile device market grows, more features are being added into a single device such as a handset, and these features need to be invoked simultaneously. To address these challenges of running multiple applications on a handset, many companies are developing "integrated" solutions for the mobile devices. Examples of these companies are Texas Instruments (TI), Intel, Ericsson, Nokia, and Qualcomm. Each company has a slightly different approach to address the same set of challenges. At the hardware level, some use single processors while others, such as Texas Instruments, use dual processors. In the single processor approach, a single processor is expanded to increase its capabilities to support more features. But more features put demands on the battery; thus single processors have been successful in the PDA market where a larger battery can be installed in the handset. Depending on the type of mobile device, the challenges also vary somewhat. For example, a cellular phone handset must meet a strict requirement for low power consumption so that it can use a smaller battery to fit in a smaller form factor. But a PDA can be larger size, with a larger touch-screen LCD, and can hold a larger battery.

This section gives a broad overview of the key developments in mobile computing platform services. For additional detail, the websites of the key players (Texas Instruments, Intel, Ericsson, Nokia, and Qualcomm) should be consulted. As an illustrative example, Section 4.8.1 briefly discusses the Texas Instrument OMAP mobile computing platform.

4.2.2 Operating Systems (OSs) for Mobile Devices

Many operating systems support the PDAs and smart phones. Main examples include Palm Inc.'s Palm OS, Microsoft's Windows CE, and Symbian's Symbian Platform. The operating system's capabilities depend on the processing power of the handheld devices and consequently the type of applications that need to be supported on these devices. Processors used range from 16 MHz to 200 MHz or higher. Low processing power is adequate for the standard calendar and contact functionality, but greatly limits the more demanding applications such as word processors, spreadsheets, and games. Due to the device differences, no operating system dominates the market for handheld devices at this stage. The general principles of OSs are briefly reviewed before discussing the main players in this area.

4.2.2.1 What are Operating Systems? – A Quick Overview

Simply stated, an operating system is the computer system's chief manager. As shown in Figure 4-3, an operating system surrounds the computing hardware and controls all other software – applications and local system software such as database managers, middleware, application software, and the user interfaces. It decides which hardware resources will be used, which programs will be run, and the order in which activities will take place. Technically, an operating system is a program, or a collection of programs, which allocates computer resources (memory, CPU, I/O devices, files, etc.) to processes (user commands, application software, middleware services, database managers, other operating systems). An operating system also gives the users a command language with which to invoke the operating system facilities and to access the computing resources. This command language, also called control language in some systems, is used to access editors, compilers, utilities and other operating system resources.

Many operating systems have been developed over the last thirty years. Examples of state-of-the-market operating systems that are used in the laptops are briefly reviewed in the sidebar, "Examples of Laptop Operating Systems." Typical functions performed by an operating system are:

- **Handle User Commands**. The operating system receives, parses and interprets the user commands. It also displays results of the user session with information about resource utilization, etc.
- **Allocation and Assignment.** The operating system allocates resources needed to execute the user commands and jobs, if the user is allowed to access the resources. It provides locations in primary memory for data and programs and controls the input and output devices such as printers, terminals, and telecommunication links.
- **Scheduling.** The operating system decides when to schedule the jobs and user requests that have been submitted. It also coordinates the scheduling in various areas of the computer so that as many things can be done in parallel as possible. The operating system

must schedule jobs according to organizational priorities. Online order processing may have priority over a job to generate mailing labels.

- **Monitoring.** The operating system monitors the activities of the computer system and the processes in the system. It keeps track of each computer job and may also keep track of who is using the system and what programs have been run, as well as generating any error messages.
- **Security.** The operating system authenticates the user request for proper authority and keeps track of any unauthorized attempts to access the system.

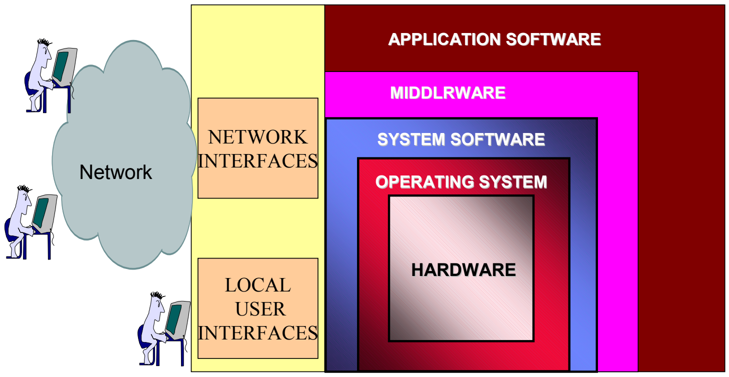


Figure 4-3: Interrelationships Between Various IT Components

Figure 4-4 shows a functional model of operating systems. The user command manager parses and executes the user commands. The memory manager allocates the main memory and the hardware registers to various processes. Most memory managers include the capabilities to manage virtual memory and translate between virtual to real memory. The CPU (central processing unit) manager allocates the CPU to the processes. A variety of CPU scheduling schemes (time-slicing, interrupt-driven) have been employed in the operating systems. The file managers manage the data resources in the system. This normally includes catalogs, file sharing, etc. The I/O (input/output) managers steward all the local and remote I/O activities. The network communication facilities may be included in some operating systems as I/O facilities.

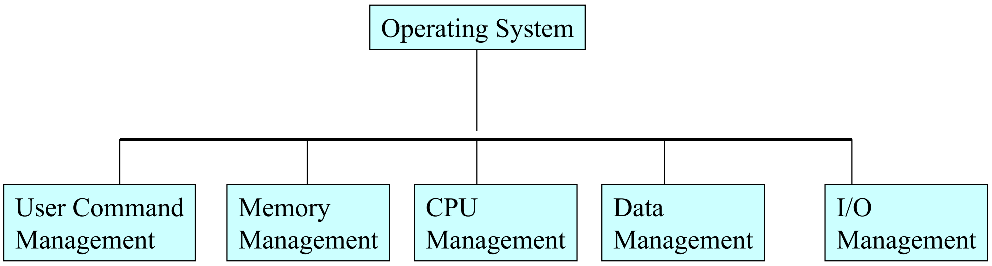


Figure 4-4: Centralized Operating System – Functional View

Examples of Laptop Operating Systems

Microsoft's Windows 98 is a 32-bit operating system that can address data in 32-bit chunks and run programs that take up more than 640 kilobytes of memory. It features multitasking, multithreading, and powerful networking capabilities, including the capability to integrate fax, email, and scheduling programs. Windows 95 was an earlier version of this operating system. Windows 98 also has support for additional hardware technologies such as MMX, digital video disk (DVD), videoconferencing cameras, scanners, TV tuner-adapter cards, and joysticks. It also includes a group collaboration tool called NetMeeting and FrontPage Express, a tool for creating and storing Web pages.

Windows Millennium Edition (Windows Me) is an improved version of Windows 98 and features tools to let users edit video recordings and put them up on the Web, and tools to simplify home networking of two or more PCs. A media player bundled with Windows Me can record, store, and play CDs, digital music downloaded from the Internet, and videos. Windows Me users can also import, store, and share photos. It also has improved capabilities for safeguarding critical files.

Windows 2000 is another 32-bit operating system with features especially for applications in large networked organizations. Windows 2000 is used as an operating system for high-performance desktop and laptop computers and network servers. There are two basic versions of Windows 2000 – a Professional version for users of standalone or client desktop and laptop computers, and several server versions designed to run on network servers and provide network management functions, including tools for creating and operating websites and other Internet services. Windows 2000 shares the same graphical user interface as the other Windows operating systems, but it has more powerful networking, multitasking, and memory-management capabilities. Windows 2000 can support software written for Windows and it can provide mainframe-like computing power for new applications with massive memory and file requirements. It can even support multiprocessing with multiple CPUs.

Windows XP (for eXPerience) combines strong features of Windows 2000 and Windows 98/Me. The Windows XP Home Edition is for home users and the Windows XP Professional Edition targets mobile and business users.

Mac OS, the operating system for the Apple Macintosh computer, provides multitasking, powerful multimedia and networking capabilities, and a mouse-driven graphical user interface. New features of this operating system allow users to connect to, explore, and publish on the Internet and World Wide Web. It also allows users to use Java software and load different language fonts in Chinese, Japanese, Korean, Indian, Hebrew, and Arabic for use in Web browser software. Mac OS X, a newer Apple operating system, has a Unix-based foundation for additional features of reliability, graphics, and open-source software.

Linux is a Unix-like operating system that runs on Intel, Motorola, Mips, and other processors. Linux is a competitor to Windows operating systems and can be downloaded from the Internet free of charge, or purchased for a small fee from companies that provide additional tools for the software. The source code for Linux is available along with the operating system software so that it can be modified by software developers to fit their particular needs. Linux is an example of open-source software, which provides all computer users with free access to its source code so that they can modify the code to fix errors or to make improvements. Open-source software such as Linux is not owned by any company or individual. Because it is free and reliable, it has become popular during the past few years among sophisticated computer users and businesses as an alternative to Unix and Windows 2000. Major application software vendors are starting to provide versions that can run on Linux.

4.2.2.2 Palm OS

Palm OS supports a very specific hardware platform designed exclusively by Palm Computing. Because of the popularity of Palm Pilot devices, Palm OS has a very large community of developers and many freeware applications for end users. The Palm Pilot has a 16-bit memory allocation scheme, thus it can only address 64K at a time.[1] This requires software developers to address data in 64K chunks. The basic versions of Palm OS (e.g., Version 4.0), include password-based security and Short Message Service (SMS) support. In addition, native Universal Serial Bus (USB) support and expanded alarm and notification capabilities are included. Support for Address Book Dial Command and 65,000 colors is also provided. Newer versions, e.g., Palm OS release 5, support more powerful processors that allow new OS functionalities for new devices and new telephony features.

4.2.2.3 Microsoft Windows CE

Windows CE is Microsoft's embedded operating system for small-footprint and portable devices such as PDAs. Windows CE is a scaled-down version of Windows to run on small handheld computers, personal digital assistants, or wireless communication devices such as pagers and cellular phones. It is a portable and compact operating system requiring very little memory. Information appliances and consumer devices can use this operating system to share information with Windows-based PCs and to connect to the Internet.

Stinger, Microsoft's platform for smart phones, is based on a scaled-down version of Windows CE 3.0. Windows CE uses the OEM Adaptation Layer (OAL) between the kernel and microprocessor to facilitate adaptation to a specific platform. This architecture has made Windows CE portable to a large number of processors. In addition, CE support for the well known Win32 application programming interface (API) has encouraged independent software vendor (ISV) development for CE applications.

Various releases of Windows CE over the years have appeared with enhancements to provide better embedded real-time capabilities ranging from car navigation systems to industrial controls. Real-time operating system (RTOS) is used to support these applications with multiple interrupt levels, efficient thread responses, and several task priorities. Enhancements have also included multimedia capabilities, integration with HTTP server and Internet Explorer for Web-based operations, support for multiple languages, and increased component support for providing modularity for developer use. Security features have been added by including Secure Socket Layer (SSL) and Kerberos in CE for client-side authentication of connections from devices to enterprise networks. Support for Bluetooth has also been added.

4.2.2.4 Symbian Platform

Symbian Platform, earlier known as EPOC, is developed and licensed by the Symbian consortium, a joint venture between Ericsson, Nokia, and Motorola. The Symbian Platform is a 32-bit multitasking operating system, with communications support for remote dial-up and infrared. This includes a telephony API that enables clients to initiate, control, and terminate data, fax, and voice calls using the same methods for any hardware. The Platform is designed for real-time capabilities for voice and data applications over wireless networks. Like Windows CE and Palm OS, it also supports voice recording and has its own synchronization code that lets smart phones exchange data with PC applications.

[1] Memory address for 16 bit address is 2^{16} = 64K.

Over the years, Symbian has introduced several product categories. Some products are data-centric for data applications, while others are more phone-centric (voice-first) design. These products have the capability to execute applications locally and thus perform tasks offline. The Symbian Platform also supports mobile phone technology with email, SMS, WAP, and HTML. It also supports general packet radio service (GPRS) networks, 3G wireless, and Bluetooth.

4.2.3 Mobile Database Management

Traditional database managers, also known as database management systems (DBMSs), provide access to databases for online and batch users. In a traditional database environment, different users can view, access and manipulate the data in a centrally located database. A traditional DBMS is designed to a) manage logical views of data so that different users can access and manipulate the data without having to know the physical representation of data, b) manage concurrent access to data by multiple users, enforcing logical isolation of transactions, c) enforce security to allow access to authorized users only, and provide integrity controls and backup/recovery of a database. Relational database managers such as DB2, Oracle, and Microsoft SQL Server are typically used in many contemporary applications. Older systems use hierarchical database managers such as IMS. Object-oriented databases are still in their infancy.

Databases in a wireless world is an interesting area of work because as the computing migrates away from the desktops and mainframes towards mobile devices, it becomes necessary to manage data on numerous PDAs, cellular phones and other such devices. The problem is that data management in these environments requires storage, retrieval, update, and synchronization of information that is dispersed across multitudes of mobile as well as immobile computing devices – large and small. A brief overview of developments in this area is summarized here. Extensive discussion of these topics is beyond the scope of this book. See *Data Management for Mobile Computing* by E. Pitoura et al. (Kluwer Academic Publishers) for details.

Many databases for mobile devices provide limited functionality that is suited for mobile devices. For example, Syware (www.syware.com) provides a database manager for Pocket PC, Windows CE, or Windows Mobile devices. These allow creation of mobile databases, printing of reports, and simple queries. This software is compatible with all Windows CE devices but is not a full-fledged DBMS.

The main trend at present is to provide almost full-functional relational database management systems, as much as possible, on mobile devices. This is becoming possible because of the increase in the processing capabilities of mobile devices and the ability of software developers to build component-based software that can fit on small devices. For example, Upright Database Technology delivers the Mimer SQL Mobile database management system on the Symbian OS for mobile phones. Mimer SQL Mobile is a full-fledged DBMS server with support for data compression, multimedia data (e.g. MMS – Multimedia Message Services) and standard interface connectivity. The product supports full industry standard SQL with full transactional support, stored procedures and triggers. The product also includes features like multi-user database access. In short, it is a full-fledged DBMS that runs on cellular phones under the Symbian OS.

Yet another trend is "downsizing" of large DBMSs to fit on small handheld devices. For example, IBM's DB2 Everyplace software provides a suite of database services that can run on a wide range of computing devices – from mainframes to PDAs. The main advantage of such an approach is that the user gets the same look and feel across all devices and also gets integration with other databases and applications. For example, DB2 Everyplace gives mobile workers the ability to query, retrieve, and modify information in real time from a wide variety of applications and data sources, including information stored in Oracle and Microsoft databases. It also provides support for the IBM WebSphere platform. DB2 Everyplace provides a Mobile Application Builder (MAB) that enables rapid application development for handheld devices based on the PalmOS and any device supporting Java. Oracle Corp has provided similar capabilities through Oracle9i Mobile.[2]

4.2.4 Mobile Transaction Management

Traditional transaction managers (TMs), also known as transaction processing monitors (TP monitors), manage the execution of transactions (sequence of statements that must be executed as a unit). TMs specialize in managing transactions from their point of origin to their termination (planned or unplanned). Some TM facilities are integrated with the DBMS facilities to allow database queries from different transactions to access/update one or several data items (IBM's IMS DB/DC is an example). However, some TMs only specialize in handling transactions only (CICS, Tuxedo and Encina are examples).

The key issue in mobile distributed environments is how to extend the scope of local transaction management to managing the execution of transactions across multiple computing devices, mobile as well as immobile. In particular, mobile applications have to operate in disconnected mode due to economical factors or unavailable connectivity. Thus mobile users collect data in standalone mode and synchronize it periodically. The common synchronization approach is based on optimistic replication techniques in which shared data is replicated on mobile computers and users are allowed to continue their work while disconnected. Updates performed by disconnected users are logged and later propagated to servers. This approach is the foundation of "Replication Servers" that are used in many large scale systems. Replication servers are commercially available from IBM, Oracle, Praxis, and others.

The optimistic synchronization works well for small populations of users who share data that is updated occasionally. In general mobile environments, several problems arise from such approaches because updates performed by different disconnected users may conflict with each other. In addition, it is not possible to determine the result of an update in the mobile device in case of a conflict. Mobile users may also wish to perform operations over data that is not locally replicated.

Several approaches to overcome these limitations have been proposed. An example is the work by Preguica [2000] on the Mobisnap system (http://asc.di.fct.unl.pt/mobisnap). In this approach, mobile clients cache subsets of the database state and allow disconnected users to perform transactions independently. Transactions are specified as mobile transactional programs that are propagated and executed in the server, thus allowing the validation of transactions based on application-specific semantics. In this model, the final result of a **transaction** is only determined when the **transaction** is processed in the central server. The system implements a reservation mechanism in order to guarantee the results of transactions performed in disconnected computers. Other approaches, especially for transactions in m-commerce, have been presented by researchers such as Veijalainen1 [2001].

[2] Alan Yeung et al., *Oracle9i Mobile* (McGraw Hill, 2002).

4.2.5 Utilities for Mobile Devices

A wide range of utilities are available on mobile devices. These include file management utilities for access and manipulation of text documents, directories, diagrams, charts, and images. Other utility programs offer calendaring and sorting functions or serve as calculators and clocks. Examples of utility programs are Microsoft Accessories, consisting of Notepad, Clock, and a Calculator, among other things. Due to the popularity of handheld devices, there are hundreds of add-on utilities for Palm OS, Windows CE, and Symbian OS. These include alarm clocks to wake you up, travel trackers that keep track of travel arrangements and add flight times and other information automatically to calendars, and a bevy of organizers, note takers, and synchronizers.

4.3 Wireless Middleware

4.3.1 What is Wireless Middleware?

Simply stated, wireless middleware interconnects mobile users, databases and applications across wireless networks. Wireless middleware, also known as mobile computing middleware, is a special class of general purpose middleware (see the sidebar, "What is Middleware?") that smoothes over the mobile computing issues, as much as possible, so that the same applications can run on wired as well as wireless networks. The following common features of wireless middleware products are needed to support mobile computing applications [Umar 2004, Vichr 2001]:

- **Connection and message delivery**: Middleware helps establish connections between mobile clients and servers over wireless networks and delivers messages over the connection. It also stores and forwards messages if the user is disconnected from the network.
- **Transformation**: The middleware transforms data from one format to another (e.g., HTML to WML). The transformation may be intelligent enough to transform different types of data to different types of devices. For example, it can produce VXML or WML depending on the type of device.
- **Detection and storage**: Wireless middleware products can detect and store mobile device characteristics in a database. Upon detecting the type of mobile device or channel being used (e.g., GSM or 802.11 frequency range), the middleware can optimize the wireless data output according to device attributes.
- **Optimization**: Middleware products can compress data to minimize the amount of data being sent over a slow cellular wireless link.
- **Security**: Security features can be imbedded in wireless middleware to ensure end-to-end security. For example, digital certificates for handheld devices can be managed by a middleware service.
- **Operation support**: Middleware can offer network and systems management utilities and tools to allow monitoring and troubleshooting wireless devices and networks. We will discuss this topic in the "Management Platforms" section of Chapter 13.

A variety of general-purpose middleware services have emerged over the years (see Umar [2004]). Example are CORBA, DCOM, MOMs (message-oriented middleware), and others. Many of these have been extended and specialized for wireless. For example, Wireless CORBA was specified by OMG as a specialization of CORBA. In the same vein, Sun's

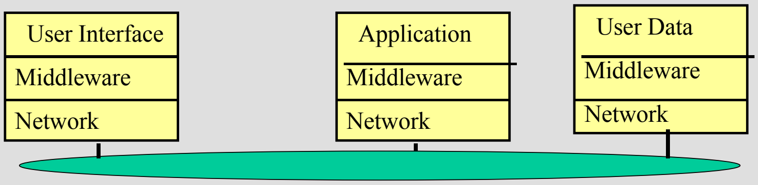
J2ME (Java 2 Micro Edition) has been specified as a family member of the Sun J2 EE (Java 2 Micro Edition) product line. In addition, common message-oriented middleware (MOM) has been extended for wireless situations (see the "Example: Message-Oriented Middleware for Mobility" in Section 4.3.5).

Naturally not all these features are needed for every mobile computing application. It is thus important to review the characteristics of mobile computing environments that are unique and then discuss the categories of applications that place different demands on the underlying network and middleware.

What is Middleware?

Middleware is the connectivity software that allows applications and users to interact with each other across a network (see the figure below). We will use the following definition:

> **Middleware** is a set of common business-unaware services that enable applications and end users to interact with each other across a network. In essence, middleware is the software that resides above the network and below the business-aware application software – it denotes a reusable, expandable set of services and functions that benefit many applications in a networked environment in terms of scalability, performance, security and interoperability.



According to this definition, the key ingredients of middleware are:

- It provides common business-unaware services.
- It enables applications and end users to interact with each other across a network.
- It resides above the network and below the business-aware application software.
- It supports scalability, performance, security and interoperability of applications.

These ingredients of our definition can be used to determine if a particular software package qualifies as middleware or not. According to our definition, the following software qualifies as middleware:

- Email (it is business unaware and interconnects users at different sites)
- Terminal emulators and file transfer packages (these business-unaware services connect users to resources)
- Web browsers (they are business unaware and support user access to resources on the Internet) As a matter of fact, as we will see in a later chapter, the World Wide Web is a collection of middleware services.
- Database drivers and gateways such as ODBC drivers and Sybase Omni Server (they provide access to remotely located databases).
- Object Management Group's CORBA (Common Object Request Broker Architecture) because it provides services for distributed object applications.
- IBM's MQSeries, a message-oriented middleware (MOM), because it provides store-and-forward capabilities for applications residing on different machines.

Here is a list of software systems that do not qualify as middleware, according to our definition:

- Operating systems such as Windows, Unix, and Linux (they operate only on local resources)
- Airline reservation systems (they are business-aware)
- Network routers (they do not reside above the network – they are part of network services)

For an extensive discussion of middleware, see A. Umar, *e-Business and Distributed Systems Handbook: Middleware Module*, 2nd ed., NGE Solutions, 2004.

4.3.2 Operational Issues in Supporting Mobile Computing Applications

Mobile computing applications, residing fully or partially on mobile devices, typically use cellular networks to transmit information over wide areas, and wireless LANs over short distances. Technologies needed to support digital applications over cellular networks designed for human users is only one aspect of mobile computing. There are several other issues that arise in supporting applications over cellular networks. Examples of these issues are:

- Network quality-of-service (QoS)
- Cost of service
- Location of users
- Characteristics of end-systems

Network QoS for cellular networks is much lower than the traditional fixed networks. While fixed LANs have been delivering data rates around 100 Mbps (million bits per second) since the mid 1990s, the wireless LANs have been struggling with data rates in the range of 11 Mbps (54 Mbps started around 2003). In wireless WANs, the data rates are typically around 19.6 Kbps, with plans for 112 Kbps and 2 Mbps. Another serious QoS issue is the error rate and the loss of packets in cellular networks. For example, TCP performance degrades significantly due to loss of packets in cellular networks. For this reason, specialized protocols that replace TCP have been proposed by WAP (Wireless Application Protocol).

The costs of network access are very different for mobile users as compared to fixed network users. Most users of a cellular network are charged based on time of day, tariffs, connection time, user location (city versus rural areas), etc. Thus casual browsing through databases and documents can be very expensive for cellular users. This fact should be taken into account while designing an application for mobile computing. However, 2.5G and 3G wireless networks solve this problem due to the use of packet switching.

The location of mobile computing users can change during an application session, thus impacting QoS and cost. It may be necessary for applications to check the location of users and the variations in QoS (for example, moving from a wireless LAN to wireless WAN range) continually during an application session. This is unheard of in fixed networks.

The end systems in many mobile computing applications are handheld devices. These devices, although improving rapidly, still differ dramatically from desktop computers in screen sizes and I/O devices that can operate with batteries. Thus screen design that may look quite impressive on a large screen may be terribly busy and difficult to operate in mobile applications. This consideration must also be taken into account when designing mobile applications.

4.3.3 Design and Development Issues in Mobile Computing Applications

Designers and developers of mobile computing applications also face many unique challenges. First, wireless networks pose many problems that most commonly available fixed networks do not have. As stated previously, wireless networks are typically slower, get congested frequently, and are more error-prone and susceptible to outages. Thus mobile computing application designers should have some knowledge of the underlying communication network. For example, database queries over wireless networks should not attempt to send thousands of rows because the network may not be available that long. This somewhat contradicts the commonly used practice in building distributed applications where the underlying communication network details are hidden from the application designers by middleware.

Besides wireless network weaknesses, the limitations of mobile devices also need to be considered. There is a potpourri of new mobile devices, such as cell phones, pagers, and personal digital assistants (PDAs). Developing applications for these devices is challenging for the following reasons:

- Different form factors for different devices. Existing mobile devices have varying numbers of display lines, horizontal or vertical screen orientation, and color or black-and-white displays.
- Different browsers and markup languages. For example, HTML is used by laptops and PDAs, wireless markup language (WML) by wireless application protocol (WAP) cell phones, and compact HTML (cHTML) by Japanese i-mode phones.
- Different device capabilities. Some devices can display images, some can make phone calls, and some can receive notification messages.

Finally, design of mobile computing applications depends on how extensively the applications use the underlying network. From this point of view, applications in mobile computing fall into the following three broad categories:

- Standalone applications
- Simple client/server (C/S) applications
- Advanced mobile applications

Stand-alone applications run entirely on mobile computers in disconnect (detached) mode. Some of the data needed by these applications is located remotely, accessed through wireless networks and typically transferred to local disk. Download of pictures and audio clips to the handset is a common example. This approach works very well if most remote data accesses are retrievals and currency of data is not a big concern. These applications do not require extensive use of wireless networks and typically require simple middleware services (e.g., file transfer over wireless networks).

Simple C/S applications are distributed between mobile computers as well as remote computers. Typically, client software runs on mobile computers and the database services are provided by remote sites. In these applications, there is a need to access remote “live” data interactively (as compared to the batch-extract method used by the standalone applications). Moreover, the amount of remote data accessed is small and must be current. Thus the connection time for C/S interactions is short. Common examples are Web browsing and m-commerce. These applications need more wireless network and wireless middleware support than standalone applications.

Advanced mobile applications involve a combination of groupware and distributed multimedia over the wireless networks. These applications typically require peer-to-peer group interactions that may use multimedia information. The information exchanged is time-critical (i.e., real-time) and the exchanged sessions may last for a long time. These applications demand a great deal of network and middleware services support.

Two of the leading middleware platforms for delivering Internet content to mobile telephones and other wireless devices – the Wireless Application Protocol (WAP) and i-mode – were designed to take into account the constraints of wireless communications: limited bandwidth and end-system processing as well as a constrained user interface. Each platform defines a standard markup language that permits an application's user interface to be specified independently of the end device. The delivery of these services is independent of the underlying networking technology, so applications can be used on different networks, just as Internet applications can. We discuss WAP in the Section (4.5).

4.3.4 Wireless Middleware Services Needed for Mobile Computing Applications

Middleware for mobile computing applications appears to follow two approaches:

- Information hiding
- Information providing

The "information hiding" wireless middleware attempts to smooth over the mobile computing issues, as much as possible, so that the same applications can run on wired as well as wireless networks. This is a conceptually desirable goal because application developers should not have to know the underlying network characteristics. This goal is met by the wireless middleware typically through specialized APIs that provide functionalities such as the following:

- Monitoring to determine whether a mobile device is powered on and within the range of wireless WAN coverage. The middleware provides this information to the applications as a status indicator and/or to end users as a screen icon.
- Optimization techniques to improve throughput by using a combination of data compression, intelligent restarts (i.e., restart at the point of disconnection and not from the beginning of session), pre-fetching, and data-caching (keeping data at local sites in case it needs to be accessed again) techniques. This middleware may also provide bundling of smaller packets to larger packets to save communication costs (most wireless systems charge by number of packets sent, reducing end-user costs).
- Monitor and limit the number of simultaneous wireless connections to be maintained by the applications.
- Provide agents that reside on the wireless servers. Agents can perform a variety of services such as bandwidth optimization, notifying the clients asynchronously if a special situation arises (e.g., a new customer arrives), automatically setting up sessions, contacting "dormant" users, and handling ad hoc queries.

This type of middleware is suitable for standalone and simple C/S applications. An example of "information hiding" wireless middleware is WAP (Wireless Application Protocol). We will discuss WAP later on in this chapter.

The "information providing" wireless middleware provides as much information about the underlying environment to the application as possible. In fact, this class of middleware exploits the network quality of service, cost, and location information for optimum performance. In particular:

- Network QoS can be used to modify application behavior. For example, the MOST project at Lancaster University uses a database application that determines the response to be sent to the user based on number of matches and the network QoS. Thus, if the application has many matches (i.e., long response to a query) but the network is slow, then the application dynamically sends only few selected matches [MOST 1992]. Similarly, different compression techniques can be used for different network QoSs (i.e., use more compression for slower networks). Another example is the system described by [Friday 1996] that uses different color schemes on the screen to interact with different classes of users (i.e., red for slow).

- Cost information can be used by the middleware to modify application behavior. For example, if a user is being charged per second, then messages can be batched up before transmission.
- Location information can be used by the middleware to direct application behavior. For example, location information can be used to route information to the nearest printers, computers, and other devices. This is used commonly in wireless sensor networks discussed in later chapters. Location information is also used by positional commerce and other location-based services (see Chapter 5).
- End-system characteristics can also be exploited by the middleware. For example, specialized user-interfaces can be displayed for mobile users. This approach is used by Microrsoft Mobile Internet Toolkit (MMIT) which generates WML, cHTML or HTML by detecting the type of mobile device. The mobile device can be also put in a “sleep” mode to conserve batteries while it is waiting for chunks of data from remote sites.

The basic philosophy of this class of middleware is to detect changes in the mobile environments and supply the change information to the applications so that they can modify their behavior changes in the mobile environments.

In many real-life situations, information-hiding as well as information-providing middleware is used by mobile computing applications. See, for example, the middleware considerations for wireless sensor networks (Section 4.8.2).

Table 4-2 shows the middleware characteristics needed for different classes of mobile applications.

Table 4-2: Middleware Characteristics to Support Mobile Applications

Application Type	Application Characteristics	Middleware Characteristics
Standalone Applications	1. Mobile computers run applications entirely on mobile computers in disconnect (standalone) mode. 2. Need to access remote files through wireless networks. 3. Most data accesses are reads and currency of data is not big concern.	Requirements are minimal. . Information-hiding middleware can do the job. Typical operation: 1. Cache copies of files at mobile computers over fixed networks and then disconnect. 2. Use the cached files in disconnect mode. 3. Reconnect to integrate the cached changes with master files.
Simple Client/Server Applications	1. Client software runs on mobile computers. 2. Need to access remote data interactively. 3. Amount of data transferred is small and not time-critical. 4. Connection time is short.	Require moderately sophisticated middleware Information-hiding is desirable. Typical functions needed are: 1. Establish connection over wireless. 2. Support client/server interactions for a short duration. 3. Disconnect.
Advanced Mobile	1. Peer-to-peer group interactions	Requires extensive middleware support.

Applications	2. Information is time-critical (i.e., real-time). 3. May use multimedia information. 4. Sessions may last for a long time.	Information-providing middleware is essential.

4.3.5 Example: Message-Oriented Middleware for Mobility

Message-oriented middleware (MOM) has been used to ensure the reliable transmission of information from one system to another. Users of MOM do not communicate directly with each other – instead they send messages through a message queue (see Figure 4-5). Basically, an application creates a message, labels it with destination information, and hands the message over to the MOM. The application is then relieved from the work of delivering the message to its destination. The receiver picks the message from the MOM service, processes it and hands the results back to the MOM service to be picked up later on. MOM thus provides asynchronous communication. Messaging middleware allows for loose coupling of software components and leads to greater flexibility – applications communicate with one another indirectly, by passing messages through a message service. MOMs can be valuable for developing sophisticated mobile applications because:

- MOM supports asynchronous processing and thus is suitable for *disconnected operation*; i.e., an application can send a message to another application, disconnect, and receive the answers later.
- MOM is recoverable from failures because the message queues on disks can be used to recover the system after failures. For example, once a message has been received and queued by MOM, these messages are not lost (the queues are used to restart applications, if needed).
- MOM can be used to link existing legacy applications with mobile clients easily without modifying any code at either side (i.e., it can redirect the output of application A to a disk queue and redirect the input of application B from the same disk queue).
- MOM supports both transactional interactions and real-time delivery of volatile information.
- MOM libraries can be very lightweight. Some MOMs such as Oracle in Motion and iBus//Mobile library described later require 50-100 kB of RAM and 70-100 kB of ROM.

MOMs for mobile computing may also support features such as: message time-to-live, (to limit the lifetime of messages) and an open bearer model, allowing you to choose whether messages should be delivered by SMS, WAP, GPRS, etc.

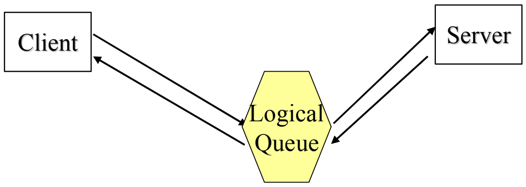


Figure 4-5: Conceptual Views of Message Oriented Middleware (MOM) Processing

Traditional MOM models use **message queuing**. Message queuing works with queues of messages. Suppliers and consumers use the queues to exchange messages. Queues can be persistent (on a disk) or non-persistent (in core). Typically, message queuing is applied to e-commerce-style interactions, where a mobile-originated purchase request has to be delivered to an e-commerce server in a highly reliable manner. When the user presses the “Buy” button, a message detailing the purchase request is put into the message queue and the MOM delivers the message to the e-commerce server.

MOM at present also supports the **publish/subscribe (P/S) model**. The main difference between message queuing and publish/subscribe is that the former implements a one-to-one communication model (much like a postal system), while the latter provides a many-to-many communication model (similar to radio transmission). In the P/S model, a producer publishes a message on a channel. The consumers subscribe to the channel of their interest and receive messages of interest to them. In contrast to message queuing, these messages are typically volatile. A topic subscriber receives only the messages published while the subscriber is active and running. Publish/subscribe can be used for delivering real-time information, such as stock quotes, sports news, or business alerts. For massive scalability, packet-oriented wireless protocols used in 2.5G and 3G cellular networks over broadcast/multicast networks will allow P/S to work between thousands or millions of publishers and subscribers.

The Sun **Java Message Service (JMS) API** specification presents a uniform interface addressing both message queuing and publish/subscribe messaging. Since its release, more than twenty vendors have endorsed the specification (including companies like IBM, Oracle, and BEA). The JMS API can be downloaded from Sun's website at http://java.sun.com/products/jms. JMS is a building block of the Java 2 Enterprise Edition (J2EE) platform.

SoftWired's iBus//mobile. An example of messaging middleware for wireless devices, SoftWired's iBus is a middleware product family aimed at supporting Internet-based distributed systems. The iBus software and documentation can be downloaded from http://www.softwired-inc.com/. iBus is implemented in the Java programming language, and is fully JMS compatible. More recently, SoftWired has introduced a version of iBus for mobile devices. This product is called iBus//Mobile, and supports Symbian's EPOC, PalmOS, and other platforms. SoftWired's iBus//Mobile consists of a lightweight JMS library to run on mobile devices, and a message server. Applications written with the mobile library can communicate with other mobile devices, or with JMS applications on PCs and servers.

On the device, the library provides a local message queue holding messages to be delivered to the message server. This is to ensure guaranteed delivery in case the device cannot establish a connection with the message server, the message server is down, or the device needs to be reset before the message is delivered to the server. The library spawns a background activity when connected that synchronizes the on-device message queue with the queue on the message server. The queue on the device typically holds just a few messages, and it is implemented using a file system or an embedded database (in flash memory). iBus//Mobile also provides an API allowing commercial embedded database products to be incorporated easily. Optionally, a device-to-device configuration is supported which does not require the presence of any message server. However, with this configuration only the JMS publish/subscribe model is available. In order to benefit from JMS features such as store-and-forward, a message server configuration is required. The key technical aspects of SoftWired's iBus//Mobile are:

- Fully implements the Java Message Service (JMS) standard. The JMS publish/subscribe model provides an event notification mechanism. This makes iBus appealing for delivering real-time information such as stock quotes and alerts to wireless information devices.
- Provides guaranteed delivery – each mobile-originated message is delivered to its destination(s) exactly once, in spite of network outages and failures of devices or message servers.

- Session persistence. The message server keeps track of what topics and queues mobile devices are subscribed to. If either the message server or the device crash, this state information is read from a database and the client state is reinitialized.
- Security. iBus//Mobile provides a pluggable security mechanism, allowing sessions and messages to be protected using symmetric or asymmetric (that is, public key) encryption.

The iBus message server, shown in Figure 4-6, serves as the central hub by performing activities such as protocol translation, message queuing and forwarding, access control to queues and topics, message encryption, and transaction management. Thus the users of iBus/Mobile can use SMS (Short Message Service), Bluetooth, TCP/IP over Ethernet or other media, or GPRS to exchange messages. The iBus server contains one protocol plug-in for each wireless bearer or transport protocol. iBus software and documentation can be downloaded from http://www.softwired-inc.com/.

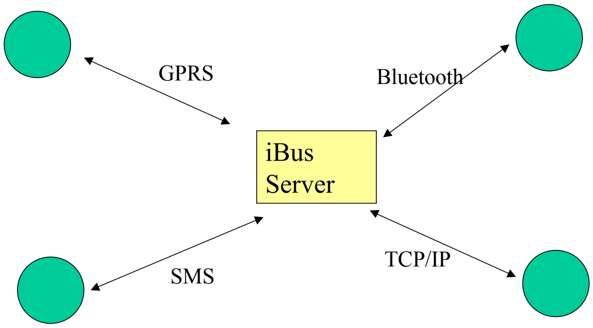


Figure 4-6: iBus/Mobile

4.4 Wireless Gateways and Mobile Application Servers

Since mobile computing applications commonly need more than one wireless middleware service (e.g., connectivity, security, compression), these services are being packaged together as:

- Wireless gateways that combine many individual services together to provide *run-time* support.
- Mobile application servers that go beyond the gateways and include mobile application *development and deployment* features.

Although this packaging does not work perfectly in all situations, it provides a conceptual framework for discussing and categorizing the wide range of products that are becoming available to support mobile computing.

4.4.1 What is a Wireless Gateway?

Simply stated, a gateway is a converter – it converts contents and protocols for disparate parties to talk to each other. Wireless gateways package several wireless middleware services that perform conversions between two distinct worlds: the Internet world and the wireless phone/data network world. The wireless gateway shown in Figure 4-18 translates between the Web server and the mobile devices and uses a three-tiered approach for mobile computing applications. The gateway is the middle tier that contains many middleware services and thus supports a thin client model by allowing the handset to be simple and inexpensive. As a middleware package, a wireless gateway provides a wide range of services such as establishment of sessions between client processes and server processes, security, compression/decompression, and failure handling. Specifically, a wireless gateway offers some of the following services:

- Connectivity services that allow the remote partners on a network to locate each other (through a directory or naming service, typically), open a connection with the remote partner, and transfer information between the remote partners
- Protocol adapters that translate requests from the wireless network protocols to the Web protocol stack
- Content encoders and decoders that translate Web content into compact encoded formats to reduce the size and number of packets traveling over the wireless data network
- Directory, naming, and location services for remote partners
- Security services such as identification, authentication, confidentiality, authorization and access control
- Performance enhancements (such as caching and compression)

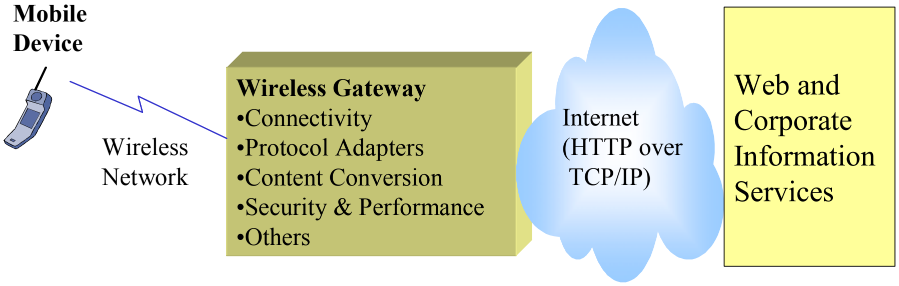


Figure 4-7: Wireless Gateway

The main advantage of a wireless gateway is that a vendor provides an integrated set of commonly used services. The services provided by the gateway depend on the differences between the two sides of the wireless gateway. For example, if the wireless network side is an 802.11 LAN supporting laptops, then the gateway may not need to provide hardly any functionality because TCP/IP runs on both sides and the laptops can handle normal Web interactions with HTML over HTTP. But if the wireless side is a 1G cellular network with handsets, then many different types of conversions are needed by the wireless gateway. An example of a wireless gateway is the iBus MOM server that performs many conversions. Other examples of wireless gateways are the WAP Gateway from Motorola, the i-mode gateway from DoCoMo, and the VoiceXML gateway from IBM. In general, XML and its variants are playing an increasingly important role in the future of wireless gateways.

4.4.2 What is a Mobile Application Server?

The growing demand for wireless applications dictates that the mobile applications must be developed quickly and be scalable, secure, robust, and flexible. This has several implications:

- Quick deployment means that an integrated development environment (IDE) is needed.
- Scalability means that a mobile application may be used by thousands or even millions of users.
- Security means authentication of users, confidentiality, non-repudiation of transactions, etc.
- Robustness implies that a transaction issued on a mobile device needs to be executed exactly once, in spite of poor network coverage or failures.

- Finally, mobile applications ought to be flexible in order to accommodate different bearers (transmission technologies: SMS, GSM, GPRS, Bluetooth, Infrared) and interaction styles (synchronous, asynchronous, transactional, one-to-one, or many-to-many).

Mobile application servers go beyond the gateways to provide the infrastructure and the development environment needed to satisfy these requirements. Figure 4-8 shows the conceptual components of a Mobile Application Server (MAS). A MAS is also part of a multi-tier (mostly three-tier) architecture that typically consists of a thin client tier residing on handheld devices, a middle tier consisting of mobile applications and a set of middleware and network services, and a back-end tier consisting of mission-critical databases and applications. Simply sated, MAS = wireless gateway + development and operational facilities. The key components of a MAS are mobile applications, back-end and wireless middleware services, network transport services, and application development environments.

Mobile Application. The purpose of a MAS is to support a mobile application. This application contains the business functions specifically developed for the mobile users and typically needs integration with back-end database or business application systems such as mainframe financial accounting systems, manufacturing systems, inventory, ERP (Enterprise Resource Planning) and CRM (Customer Resource Management) systems. This application may be a MEBA or M-P-T-V (Mobile, Positional, Television, Voice) Commerce.

General Purpose (Back-end) Middleware. To support the back-end systems, common back-end middleware services such as database and transaction processing are needed. We discussed these and other general purpose middleware services in the "Middleware" Module.

Mobile Devices. This may be a cellular phone, notebook, handheld computer, pen computer, PDA, PalmOS compatible PDA, Windows CE/Pocket PC device, or a two-way interactive pager. These devices operate under different mobile operating systems that reside in the mobile device – it may be Windows98/2000/NT, PalmOS, Symbian OS, Win CE (or Pocket PC), EPOC, a specialized OS like Blackberry, or a voice/Web browser.

Wireless Network. This may be a wireless LAN, WAN or MAN. These networks are provided by companies such as Cingular (formerly Bell South Wireless Data), Verizon, Sprint, Metricom, Nextel, Bell Mobility (Canada), AT&T (Canada), BT in UK, and NTT DoCoMo (Japan). To connect computing devices to the wireless network, you need a suitably-configured wireless WAN modem, wireless LAN adapter, or wireless MAN (metro-area network) adapter. While wireless network provides true mobility, you may utilize a wireline network for those mobile users who need occasional connection from hotels, motels, or airport lounges. Some of these airports are now offering wireless LAN connectivity to wireline back-end networks.

Wireless Middleware. This middleware, also known as mobility middleware, is responsible for handling mobile devices and is the heart of a MAS. This middleware takes raw data from database applications/queries and transforms the data to a specific thin client (or a thick client like a PC) considering its presentation space characteristics and limitations. It breaks the messages into smaller chunks, filters redundant information, and optionally compresses the data, etc. This middleware also provides mobility-specific services such as location, and supports application programming level interfaces (APIs) with specialized communications protocols for wireless. Wireless middleware in most MASs takes the form of a wireless gateway as discussed previously.

Wireless Software Development and Monitoring/Control Facilities. Special facilities, such as Software Development Kits (SDKs) are needed to build the mobile applications of the future. These SDKs provide GUI development tools to build mobile application modules that use the wireless middleware APIs. In addition, mobile applications need to be monitored and controlled at run time for errors and load balancing. Wireless software development and monitoring/control facilities are very closely tied to the wireless middleware and are usually packaged with them. For example, the Nokia WAP Server has an SDK and monitoring control commands for WAP.

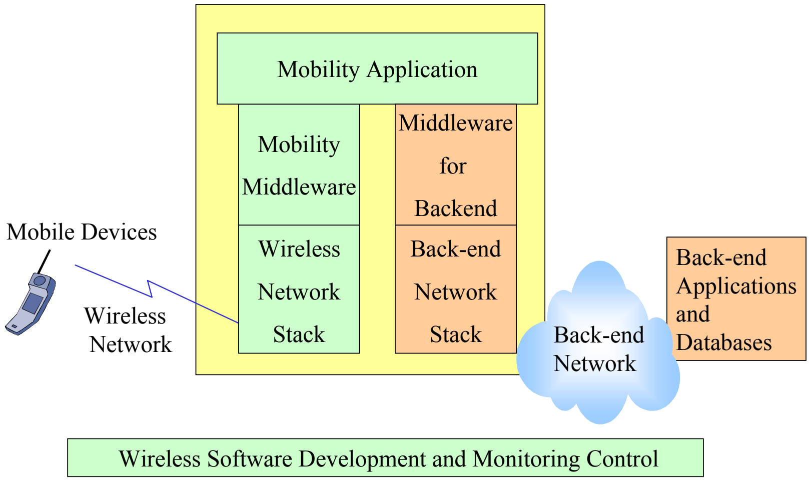


Figure 4-8: Mobile Application Server – Conceptual View

4.4.3 Examples of Mobile Application Servers

Within the conceptual model presented above, a wide range of MASs from different vendors are available commercially. Some mobile application servers are generic web servers with an SDK (Systems Development Kit) or API capability to pull data from back-end systems and send it to a browser-based client software in a handheld device. Other application servers provide a business application with customization capability for wireless access. Still other mobile application servers are extensions of wireless gateways where the vendor decided to add development capabilities to the gateways. Depending on the heritage of the vendor and their core expertise, you can categorize application servers in the following broad classes:

- Generic application servers with a web-based SDK with support for handheld devices and wireless networks. Examples are the Netscape, Microsoft, Sun, and BEA application servers with support for wireless devices.
- Generic database servers with support for mobile devices. Oracle MAS is an example.
- eBusiness/eCommerce Application Servers that have been built specifically to support EC/EB applications. Mobility has been added as a feature to these severs. IBM's WebSphere is an example. We will discuss these servers in the next two chapters.
- Mobility-specific application servers that were built primarily for mobile applications. Nokia's WAP Server with SDK is an example.

Naturally a given MAS is stronger in those aspects that are its core features and its heritage. For example, the mobile application server from Oracle may be stronger on integration with databases but not necessarily as strong on interfaces with a variety of handheld devices and wireless networks. However, a MAS from Nokia may be stronger on wireless and mobile device support than on database integration. You have to know your requirements clearly before selecting a MAS (so what is new!). A partial list of commercially available MAS's is shown in the following sidebar (consult the websites for details). In the next few sections, we will review the core technologies that support mobile application servers (e.g., WAP, i-mode, Wireless Java).

Examples of Commercially Available Mobile Applications Servers

- Aligo M-1 Mobile Application Server (www.aligo.com) uses the wireless Java – specifically the Sun J2ME (Java 2 Micro Edition) – environment to provide the resources and services that developers and architects need to design, build and deploy Java applications for mobile devices.
- IBM's WebSphere Application Server (www.ibm.com) has the facilities for serious mobile EB application development for large enterprises. However, this is a large and complex system and you must be ready to allocate enough resources to implement solutions based on this MAS.
- Jacada Presentation Server for Palm (www.jacada.com) addresses the needs of developing mission-critical information from enterprise systems to the web through handheld devices.
- Nokia's WAP Application Gateway (www.nokia.com) uses WAP and has an extensive array of facilties for mobile device support with connections to back-end applications.
- Oracle's Mobile Application Server (www.oracle.com) allows access to Oracle's databases and application suite from mobile devices.
- Sybase Mobile Application Studio (www.sybase.com) provides a studio of tools for wireless access to Sybase and other databases.
- @Hand Mobile Application Server (www.@hand.com) provides core infrastructure, administrative services and a rapid application development environment for creating mobile applications.

Time to Take a Break

✓ • Local Services and Wireless Middleware

• WAP, i-mode, J2ME, MMIT, BREW

• Voice XML, Examples and Case Studies

4.5 The Wireless Application Protocol (WAP)

4.5.1 Overview

WAP is a set of protocols to enable the presentation and delivery of wireless information and telephony services on mobile phones and other wireless devices. Two main constraints cause this market to be different from the wireline market. First, the wireless links are typically constrained by low bandwidth, high latency, and high error rates. Second, the wireless devices are constrained by limited CPU power, limited memory and battery life, and the need for a simple user interface.

WAP specifications address these issues by using the existing standards where possible, with or without modifications, and also by developing new standards that are optimized for the wireless environment where needed. The WAP specification has been designed such that it is independent of the air interface used, as well as independent of any particular device.

The key elements of the WAP specification, discussed later in this section, are:

1. A WAP programming model, shown in Figure 4-9. The model is based heavily on the existing WWW programming model; i.e., a WAP gateway translates between the Web server and the WAP clients.

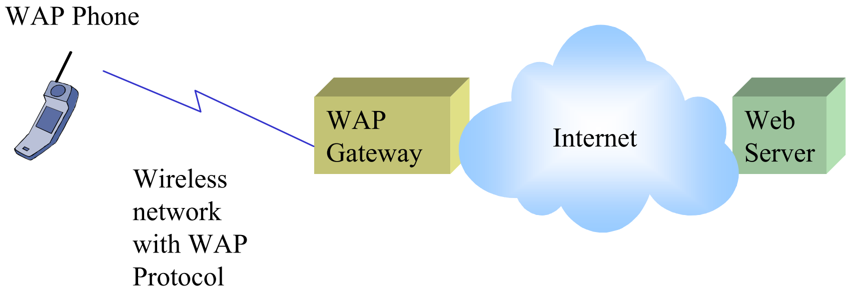


Figure 4-9: WAP Architecture

2. A Wireless Application Environment (WAE) for creating WAP applications and services. The main elements of WAE are:

- A markup language called Wireless Markup Language (WML) that is similar to XML but that has been optimized for wireless links and devices. A scripting language (WMLScript) is also provided.
- Specification of a microbrowser in the wireless terminal. This is analogous to the standard Web browser – it interprets WML and WMLScript in the handset and controls presentation to the user.
- A framework, the Wireless Telephony Applications (WTA) specification, to allow access to telephony services such as call control, messaging, etc. from within the WMLScript applets.

3. A protocol stack designed specifically for the wireless environment, as shown in Figure 4-10. WAP supports a lightweight protocol stack to minimize bandwidth requirements, guaranteeing that a variety of wireless networks can run WAP applications. The WAP protocol stack is similar to Internet protocols but is optimized for wireless information pull and push. It supports WAE and consists of (we will discuss these later):

- Wireless Session Protocol (WSP)
- Wireless Transport Layer Security (WTLS)
- Wireless Transaction Protocol (WTP)
- Wireless Datagram Protocol (WDP)
- Wireless network interface definitions

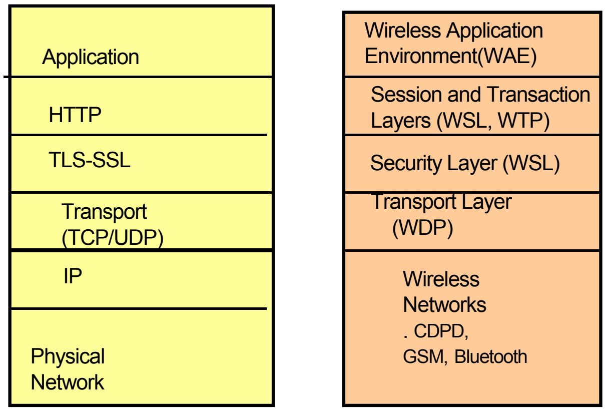


Figure 4-10: WAP Protocol Stack

Basically, WAP supports development of applications appropriate for handheld devices that are tailored specifically for the wireless Internet. The WAP Forum, now part of the Open Mobility Alliance (OMA), has taken technology elements from TCP/IP, HTTP and XML, optimized them for the wireless environment, and is submitting these optimizations to the W3C standards process as input for the next generations of (XHTML) and HTTP (HTTP-NG).

Before looking at details, let us briefly review how it works at a high level. Wireless devices communicate through the wireless network to a WAP server. A WAP server converts data or Web pages between WAP and TCP/IP. This conversion lets conventional Web servers send WML pages to wireless devices, which use microbrowsers that let users surf the Web. Tools are emerging that will automate the ability to author content for multiple devices: cell phones, palmtops, and desktops. XML will help this situation by separating information into pure XML content and pure XML style sheet language (XSL)-based presentation. The point is to design an XML document architecture that separates presentation method, which varies by device, from content. In this way, the XML-based content can be translated to HTML for conventional browsers and to WML for microbrowsers by using different XSL scripts.

4.5.2 Why WAP is Needed

Wireless Application Protocol (WAP) is a result of continuous work to define an industry-wide standard for developing applications over wireless communication networks. The WAP Forum (www.wapforum.org), originally founded in 1997 by Ericsson, Motorola, Nokia, and Unwired PlanetWML, was formed to create the global wireless protocol specification that works across differing wireless network technology types. The first specification of WAP was released in June 1999. According to the WAP forum:

> "WAP is an open, global specification that empowers mobile users with wireless devices to easily access and interact with information and services instantly."

At the time of this writing, the WAP Forum has merged into the Open Mobile Alliance (OMA) and no longer exists as an independent organization. The specification work from WAP continues within OMA and can be found on the OMA website at: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html. Henceforth, any reference to the WAP Forum should be redirected to OMA.

Why is WAP needed? The main idea is to provide an open and global specification that addresses the wireless network, devices, interoperability, and market issues in a cohesive manner.

a) Network Differences: Wireless networks and terminals have specific needs not addressed by existing Internet technology, e.g.:

- TCP is highly inefficient over wireless links since it interprets packet errors as congestion.
- HTTP uses text format, not compressed binary, headers.
- HTML is not suitable for small screens (e.g., the 4-line, 20-character displays of cellular phones).
- TLS (Transport Layer Security) requires several message exchanges between client and server. This is not suitable for wireless networks.

In general, wireless data networks present a more constrained communication environment compared to wired networks. Because of fundamental limitations of power, available spectrum and mobility, wireless data networks tend to have:

- Less bandwidth
- More latency
- Less connection stability
- Less predictable availability

Furthermore, as bandwidth increases, the handset's power consumption also increases, which further taxes the already limited battery life of a mobile device. Therefore, even as wireless networks capitalize on higher bandwidth, the power of a handset will always be limited by battery capacity and size, thus challenging the amount of data throughput. Deployment of the WAP standard will accommodate more users per MHz since it uses the available bandwidth at an extremely efficient level. The result of placing more users on a given amount of spectrum can yield lower costs for both the network provider and the customer. A wireless data solution must be able to overcome these network limitations and still deliver a satisfactory user experience.

b) Device Differences. Similarly, mass-market, handheld wireless devices present a more constrained computing environment compared to desktop computers. Because of fundamental limitations of battery life and form factor, mass-market handheld devices tend to have

- Less powerful CPUs
- Less memory (ROM and RAM)
- Restricted power consumption
- Smaller displays
- Different input devices (e.g., a phone keypad, voice input, etc.)

Because of these limitations, the user interface of a wireless handset is fundamentally different than that of a desktop computer. The limited screen size and lack of a mouse requires a different user interface metaphor than the traditional desktop GUI.

These conditions are not likely to change dramatically in the near future. The most popular wireless handsets have been designed to be lightweight and fit comfortably in the palm of the hand. Furthermore, consumers desire handsets with longer battery life, which will always limit available bandwidth, and the power consumption of the CPU, memory and display.

Because there will always be a performance gap between the very best desktop computers and the very best handheld devices, the method used to deliver wireless data to these devices will have to effectively address this gap. As this gap changes over time, standards will have to continually evolve to keep pace with available functionality and market needs.

c) Interoperability Issues. Equipment and software offered by different suppliers needs to work together. The WAP specification has been designed to encourage interoperability between its key components. Any solution component built to be compliant with the WAP specification can interoperate with any other WAP-compliant component. WAP assures handset manufacturers that if their device complies with the WAP specification it will be able to interface with any WAP-compliant server, regardless of the manufacturer. Likewise, the makers of a WAP-compliant server are assured that any WAP-compliant handset will interface correctly with their servers.

d**) Internet Market Issues**. The WAP specification is designed to bring Internet access to the wireless mass market. By building open specifications, and encouraging communication and technical exchanges among the industry players, the WAP Forum began to open the wireless data market in new ways. The revolution is under way to bring information access to any handset, at a reasonable price and in an easy-to-use form factor.

4.5.3 The New WAP – WAP 2.0

Currently the best-known example of WAP is known as the WAP 1.x stack, which introduced the new protocols such as WDP, WSL, WTP, etc. These layers were designed for slow WANs. However, 2.5G and 3G cellular networks are now becoming a reality. A new version of WAP (known as WAP 2.0) was announced in January 2002 to take advantage of the new cellular networks. As shown in Figure 4-11, the WAP 2.0 essentially eliminates the WAP 1.x-specific layers (WSP, WTP, WSL, and WDP). It only retains the WAP Application Environment (WAE) which resides directly on top of the typical TCP/IP stack.

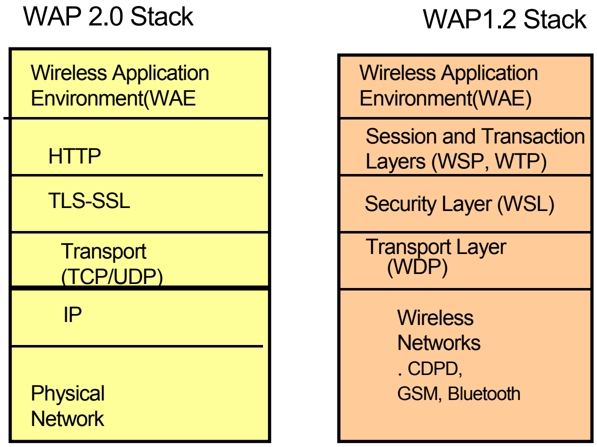


Figure 4-11: WAP 2.0 Versus WAP 1.x Stack

There are several implications of this. The main implication is that the thick WAP gateway is replaced by a proxy that translates between WAP2.0 to back-end Web servers. Although WAP 2.0 may have significant impact, we devote most of the discussion to WAP 1.x in this chapter because most WAP systems use WAP1.x. If WAP2.0 becomes important, we will take a closer look at it.

4.5.4 Wireless Application Environment (WAE)

The Wireless Application Environment (WAE) is a complete environment for creating WAP applications and services. The Wireless Application Environment (WAE) consists of three main parts:

- WML and WML Microbrowser (see sections 4.5.5 and 4.5.6)
- WMLScript including a virtual machine and set of support libraries (see Section 4.5.7)
- Wireless Telephony Application (WTA) and the API (WTAI) (see Section 4.5.8)

WAE is naturally of prime interest to developers of wireless applications. To minimize training of developers, the WAE mimics to a large extent the Web programming model. Thus anyone familiar with Web programming can quickly learn to develop WAP applications.

WAE also consists of WAE User agents (software that executes in the wireless devices) and content generators (applications that produce standard content formats in response to requests from user agents in the mobile terminal). Standard content encoding is defined to allow a WAE user agent to navigate Web content.

Several documents specify WAE. Examples are:

- Wireless Application Environment Specification (WAE) – the root document in the WAE document hierarchy that specifies and references core WAE elements.
- Wireless Markup Language Specification (WML) – describes the mark-up language, WML, including its semantics, its XML document type definition (DTD) and its encoding extensions.
- Wireless Telephony Application Specification (WTA) – specifies the technologies included in the Wireless Telephony Application reference architecture.
- Wireless Telephony Application Interface (WTAI) – describes standard telephony-specific extensions to WAE, including WML and WMLScript interfaces to such items as call control features, address book and phonebook services.
- WMLScript Specification (WMLScript) – describes the scripting language, WMLScript, including its lexical and syntactic grammar, its transfer format and a reference bytecode interpreter.
- WMLScript Standard Libraries Specification (WAEStdLib) – describes standard libraries available to WMLScript programs, including a language library, a string library, a dialog library, a floating-point library, a browser library and a URL library.
- WAP Binary XML Format Specification (WBXML) – describes the XML document encoding and transfer framework used by WAE.

The first generation of WAE (specified in WAP 1.1) defines the basic architecture and its components (e.g., WML, WMLScript). The second generation of WAE (WAP 1.2) specifies a push model and some performance enhancements. See Section 4.5.9 for an overview of the WAP push model.

4.5.5 Wireless Markup Language (WML)

4.5.5.1 WML Overview

WML is a markup language that is very similar, in principle, to XML. WML is basically a flavor of XML for creation of information that handheld computers, Palm Pilots, smart cell phones, pagers, and other wireless devices can read. WML is designed with the constraints of small mobile devices in mind. These constraints include: 1) small display and limited user input facilities; 2) narrowband network connection; 3) limited memory and computational resources.

Unlike the flat structure of HTML documents, WML documents are divided into a set of well-defined units of user interactions. One unit of interaction is called a ***card.*** Services are defined in terms of user navigations between cards from one or several WML documents. WML provides a smaller, telephony-aware set of markup tags that are more appropriate than for handheld devices. From the WAP Gateway, all WML content is accessed over the Internet using standard HTTP 1.1 requests. WML includes four major functional areas:

- Deck/card organizational metaphor – all information in WML is organized into a collection of cards and decks. A deck is the page of markup data that gets sent to the phone when it requests a given URL.
- Inter-card navigation and linking – WML includes support for explicitly managing the navigation between cards and decks.
- Text presentation and layout – WML includes text and image support, including a variety of formatting and layout commands.
- String parameterization and state management – all WML decks can be parameterised, using a state model.

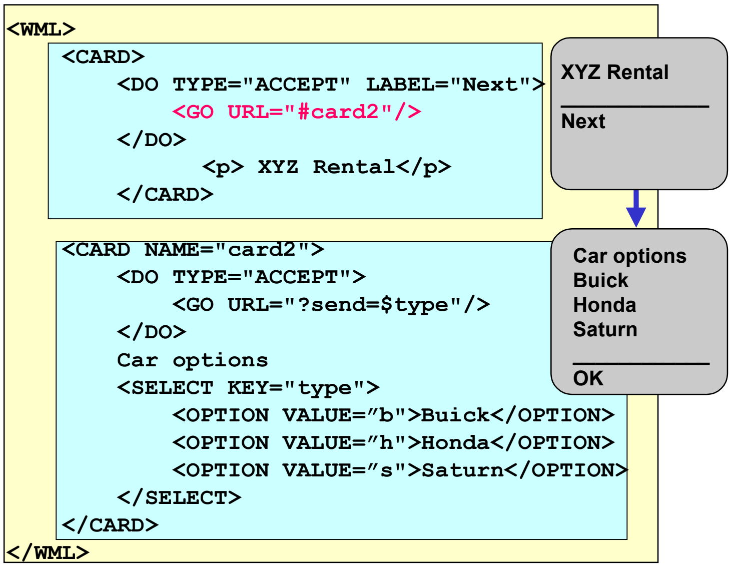
4.5.5.2 WML Examples

WML specifies card decks (identified through <card> and </card> tags) that can be displayed on a cellular phone. The card element can have an ID and title. The following is an example of WML that can be used to display “Hello World” on your cellular phone:

```
<wml>
<card id=‘main” title=“Example1”>
  <p>
                    Hello World;
 </p>
</card>
</wml>
```

A WML file must be enclosed with <wml> and </wml> tags. The WML statements are stored in a file with the .wml extension.

Figure 4-12 shows another example of WML in which a user can select a car from a car rental company. The example shows two WML cards, with outputs shown to the user on the right. The first card basically displays the company logo “XYZ Rental” and passes control to the next card “card2” through the “go URL” statements. The “go” statement is used to transfer control. .The control can be passed to a card (e.g., the “go URL” statement) or an external site through a “go http:” statement. Within each card, you can insert programming constructs (with <do> </do> tags) to do specialized processing (e.g., going to next card). The “accept” on the do statement indicates a yes (or OK) button on the cellular phone. The main selection takes place in the second card. In this card, the user is given three options, indicated by <option> and </option> tags. The options are surrounded by <select> and </select>. If a particular option is chosen, then a value (e.g., “h” for Honda) is passed to the “go URL statement” which passes control to the appropriate card for processing the chosen car.



```
<WML>
    <CARD>
        <DO TYPE="ACCEPT" LABEL="Next">
            <GO URL="#card2"/>
        </DO>
               <p> XYZ Rental</p>
        </CARD>

    <CARD NAME="card2">
        <DO TYPE="ACCEPT">
            <GO URL="?send=$type"/>
        </DO>
        Car options
        <SELECT KEY="type">
            <OPTION VALUE=”b">Buick</OPTION>
            <OPTION VALUE=”h">Honda</OPTION>
            <OPTION VALUE=”s">Saturn</OPTION>
        </SELECT>
    </CARD>
</WML>
```

Figure 4-12: WML Example

WML has many features for WAP application developers. The features combine HTML and XML. Examples of the features are (you can assume that each <> tag is ended by a </> tag unless otherwise indicated):

- Advanced display features such as tables (through <table> tag) and images (through <img> tag)
- User interaction features to support option selections (through <select> and <option> tags), templates (through <template> tag), events (through <onevent> tag), timers (through <timer> tag), defining and using variables (through <setvar > tag), input handling (through <input> tag), and posting data (through <post> tag)
- Text formatting such as <br/> for breaking lines, <b> for bold, <I> for italic, <u> for underline, etc.
- Navigation through anchors (e.g., <a href="…">) and traveling back to previous cards through <prev/>

Details of WML programming are beyond the scope of this book. Many books (such as *Professional WAP* by Arehart et al., Wrox Books, 2000) discuss programming details at great length.

4.5.5.3 Benefits of Using WML

Application developers can reach a large audience when they write their applications in WML because WAP is supported heavily at present. Additional benefits of WML for developers include:

- WML is an XML-based language – thus it is an easy markup language for existing Web developers to learn.
- Content written in XML-defined markup languages can be automatically translated into content suitable for either HTML or WML by using an XSLstyle sheet (see Figure 4-13). In addition, content written in well-formed XML can also be translated to other XML-based markup languages, using a different XSL style sheet.
- WML has been designed to be an integral part of universal content with internationalization features.
- Since WML is part of an open standard, and was developed by an independent organization, there no unique access to APIs or special functionality.

- By writing in WML, a developer's work becomes available to any network and device that is WAP-compliant.
- WML supports development of powerful user interfaces. Applications can map soft keys for easy user input and use special features to maximize the effect of displaying text on a limited screen.
- An application written in WML will look good on any device that is WAP-compliant. If one device is able to display more lines of text than another, the microbrowser will do so automatically, making the best use of the device's and application's capabilities.
- WML allows the use of icons and bitmapped graphics, for devices that support them. One application will work equally well on a phone with or without graphics by offering alternate text to the phone that is not capable of displaying images.
- An application can be customized to take advantage of a particular device's capabilities, by using standard HTTP header mechanisms to learn about the device's capabilities.

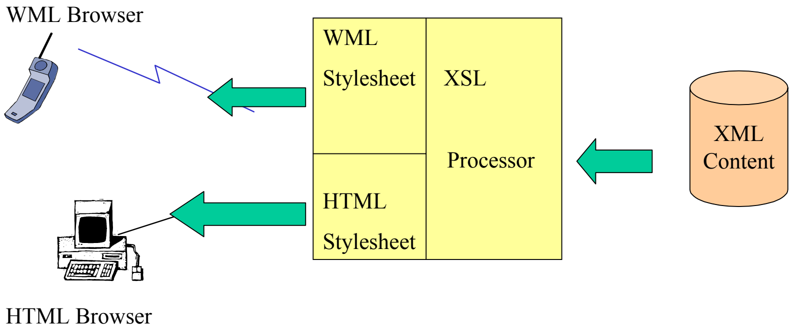


Figure 4-13: The Future of Content Description

4.5.6 WAP Microbrowsers

WML works with a WAP microbrowser that resides in the handheld device and renders the WML content. It controls the user interface and is analogous to a standard Web browser. The microbrowser is a very thin client, able to fit in a limited amount of memory in the handheld device. The microbrowser interprets WML and WMLScript in the handset and renders it according to the charatcteristics of the handset. The Web gateway combined with compression in the network interface reduces the processing load at the handheld device so that an inexpensive CPU can be used in the handset with low power consumption.

The microbrowsers are designed for wireless handsets for compact yet flexible user interfaces. Users navigate through cards with up and down scroll keys instead of a mouse. Soft keys allow the user to perform specific operations appropriate to the application context, or to select menu options. A traditional 12-key phone keypad is used to enter alphanumeric characters, including a full set of standard symbols. The microbrowser allows devices with larger screens and more features to automatically display more content, just as a traditional browser does on a PC when the browser window is expanded on the screen. Navigation functions such as Back, Home, and Bookmark are also provided, in keeping with the standard browser model.

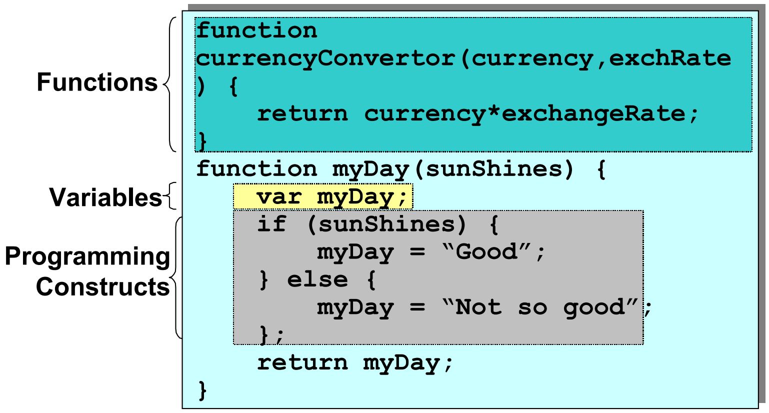
Handset manufacturers can integrate a microbrowser into their product line to be a player in mobile commerce. By offering a WAP-based solution, handset manufacturers can integrate a microbrowser into the handset at low cost, since the WAP specification calls for a low-profile browser that does not demand large memory or expensive CPU requirements.

4.5.7 WMLScript

WMLScript is a scripting language derived from JavaScript that has been optimized for use with small-CPU and small-memory devices. It omits some functions of JavaScript and integrates easily with WML. Unlike JavaScript or VBScript, which can run on client or server machines, WMLScript is designed solely to run on the client (i.e., handset). WMLScript:

- Assumes a bytecode-based, stack-oriented virtual machine
- Assumes compiler is in network for reducing network bandwidth and terminal memory/CPU usage
- Provides libraries for URL processing, simple dialog (UI) processing, math and string processing, and WML browser interface

Figure 4-14 shows an example of WMLScript. All WMLScript is done with the use of functions. The following example shows two functions. The first converts currency and the second returns a day (good or bad depending on sunshine).



```
function
currencyConvertor(currency,exchRate
) {
    return currency*exchangeRate;
}
function myDay(sunShines) {
    var myDay;
    if (sunShines) {
        myDay = “Good”;
    } else {
        myDay = “Not so good”;
    };
    return myDay;
}
```

Figure 4-14: WMLScript Example

4.5.8 Wireless Telephony Application (WTA)

Wireless Telephony Applications (WTA) allows access to telephony functionality such as call control, phone book access and messaging from within WMLScript applets. WTA allows operators to develop secure telephony applications integrated into WML/WMLScript services. For example, services such as Call Forwarding may provide a user interface that prompts the user to make a choice between accepting a call, forwarding it to another person or forwarding it to voicemail. WTA adds to the WAP architecture a user agent on the client side and a WTA server on the other side for mobile telephony. It adds extensions to the WML/WMLScript browser and provides security by having a separate WTA browser and port.

WTA uses the Wireless Telephony Application Interface (WTAI) specification that describes standard telephony-specific extensions to WAE. The extensions include WML and WMLScript interfaces to such items as call control features, address book and phonebook services. WTAI is basically an API to facilitate functions such as the following:

- Making a mobile-originated call
- Receiving a call
- Sending and receiving short messages
- Examining call logs

- Pressing keys on the keypad
- Call control (call forwarding, call hold, call transfer)
- Voicemail
- Phone book interface
- Event processing

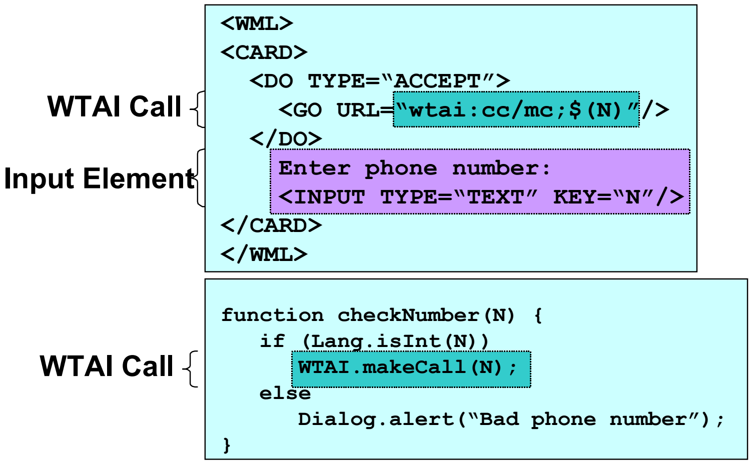
Telephony applications can be written by using WML or WMLScript. For that reason WTAI functions can be accessed from both. For example, making a call to a number (555-1111) would look like the following in WMLScript:

```
WTApublic.makecall("5551111);
```

The same call can be made by using the following WTAI WML statement:

```
wtai:/cc/mc;5551111;
```

Figure 4-15 shows how WTAI can be used to place a phone call. The first WML card prompts user for a phone number. This number ($N) is passed to WTAI to make the call (wtai:/cc/mc is the WTAI command for activating a phone call). The second card shows the WMLScript function that checks the phone number.



```
<WML>
<CARD>
  <DO TYPE="ACCEPT">
    <GO URL="wtai:cc/mc;$(N)"/>
  </DO>
    Enter phone number:
    <INPUT TYPE="TEXT" KEY="N"/>
</CARD>
</WML>
```

```
function checkNumber(N) {
   if (Lang.isInt(N))
      WTAI.makeCall(N);
   else
      Dialog.alert("Bad phone number");
}
```

Figure 4-15: WTAI Example – Placing a Phone Call

The Wireless Telephony Application (WTA) allows application developers to initiate phone calls from the browser and respond to network events as they occur. The WTA API accomplishes this by providing an interface to the local and network telephony infrastructure. The local interface allows WML and WMLScript to access a specific set of telephony functions, such as a function call to dial a phone number from the mobile handset. The network interface allows an application to monitor and initiate mobile network events, so that the application can take action or update information based on these events. This functionality can be used to keep an updated list of the phone numbers dialed into an active conference call. These network and local APIs are powerful features that no other standard provides.

4.5.9 WAP Push Model

The **push model** is used to deliver information to the users at certain times (e.g., the evening news) or due to certain events (e.g., earthquake news). This model is very different from the **pull model** in which the users can ask for information whenever they need by running a query. The push model is based on the publish/subscribe model, in which publishers "push" information onto a channel, while the consumers subscribe to the channel of interest and the information is automatically pushed to them. See the example in Section 4.8.6 that shows how Drexel University uses the WAP push model to send information to students without them asking for it.

Standard HTTP has no support for "push" functionality. The WAP specification defines a push mechanism that will allow any Web server to send information to the client. This is an extremely important feature because it allows applications to alert the subscriber when time-sensitive information changes. There are a number of applications that make use of this functionality, such as traffic alerts and stock quote triggers, or email and pager notifications. The WAP push framework consists of a Push Access Protocol (PAP) and Push Over the Air (POTA) protocol. WAP push has found several applications (see, for example, the discussion of WAP in SMS and MMS in the next section). For a detailed discussion of WAP Push, see "Professional WAP" by Arehart et al., Wrox Books, 2000, Chapter 16).

4.5.10 The Role of WAP in SMS and MMS

SMS (Short Message Services) and MMS (Multi-media Message Service), introduced in Chapter 2, are popular messaging applications for mobile users. WAP is playing an important role in these services. WAP as well as messaging services have several similarities. In particular, they both try to tailor the content to the limitations of the phone – mainly the small text-only display, and the restrictive keyboard and navigation keys. So parts of WAP as well as SMS/MMS are concerned with sensible data formatting and navigation appropriate to the handset limitations. In addition, sending data over mobile air interfaces poses problems with delays and slow links. Thus, an efficient and optimized transport protocol for cellular phones is an important aspect of WAP. Naturally, handset applications may use SMS/MMS, WAP, or both.

4.5.10.1 WAP and SMS

SMS, as discussed in Chapter 2, is the short messaging service for GSM and other digital cellular networks. It provides 2-way short messages to be sent between GSM subscribers and also supports message exchanges with other systems such as Internet email and the Web through gateways. Since SMS provides a transport service for messages (the messages may be anything), it can be used to exchange WAP-compliant data. In other words, instead of using the entire WAP stack, the WML messages could be transported over SMS. There are basically three scenarios:

- SMS by itself – no WAP
- SMS to carry WAP data
- Pure WAP – no SMS

In the first scenario, SMS is used to exchange data without any WAP. For simple messaging applications, the simple send-and-receive primitives of SMS are sufficient because there is no context to maintain on an ongoing basis. For applications like sending a letter or an email where no immediate, or even any, response is required, SMS works quite well as a connectionless transport mechanism. Many SMS messages are alerts of one kind or another, used to notify the recipient of an event. Connectionless applications that require no followup action can, of course, be built by using WAP – but this is not necessary because WAP is not widely available, and there are millions of phones that can handle SMS but not WAP.

WAP messages, just the WML and images, can be transported over SMS. Why? Suppose you just want to send WAP messages to a phone that does not have the entire WAP stack but has SMS support. The SMS protocol consists of simple reliable 2-way messaging with a very basic command set such as “Send Message” and “Receive Message.” Thus WAP traffic can be carried over SMS for simple applications, but for anything more sophisticated, the SMS protocol is very limited. However, a more sophisticated protocol could be added on top of SMS, with more commands that are sent using the Send and Receive of SMS. For example, instead of WAP data messages, the WDP (Wireless Datagram Protocol) packets can be transported over SMS. WDP, as discussed later, is part of the WAP stack.

The third scenario of using WAP exclusively is particularly useful for conversational services on the handset. Although interactive services can be built by using native SMS, this is not as elegant as WAP. Using WAP, the user can be prompted for information and guided along various conversational paths, but SMS-based conversational applications force the user to remember how to respond with preset commands. As the doctors say, this could cause “minor discomfort.”

In reality, SMS and WAP complement each other and can be used to support different applications on the handset by using mixtures of the above scenarios. SMS is particularly good for pushing out information to mobile phone users, such as new emails or data changes that occur on a corporate database. These alerts can be followed up by a variety of actions. These may include SMS replies of one form or another. In addition to alerts, SMS can also be used to pull data from a corporate database. For example, a mobile worker could get the phone number of a customer, their address, fax number, etc. For many of these types of applications, the quick alert or prompt/pull operations of SMS are ideal. Indeed, an advantage of SMS is that it is quick. WAP, on the other hand, provides greater interactivity with the data source and is especially useful for operations such as navigating through a catalog, making travel plans. and traversing an email Inbox. The two could be combined so that a phone could receive an SMS alert (e.g., an emergency meeting), based on which the user may connect to a remote server to view needed documents online before attending the meeting.

4.5.10.2 WAP and MMS

MMS, as discussed in Chapter 2, is used to carry messages with multimedia elements instead of the plain text of SMS. The MMS messages can be sent either to a phone number or email address, just as SMS messages can. So most of the SMS discussion applies. But MMS messages are content-rich and are created typically by using a template that specifies the relative position of multimedia elements in the message. The message with all this information is sent to the user’s MMS server using the WAP PUT command. From the MMS server, the message can either be sent to a phone or sent to the recipient’s email account.

4.5.11 The Traditional WAP Protocol Stack

4.5.11.1 Overview

The WAP 1.x protocol stack is similar to Internet protocols but is optimized for wireless information pull and push. The WAP Protocols support the Wireless Application Environment (WAE) that we have discussed so far. The stack consists of several layers shown in Figure 4-16.

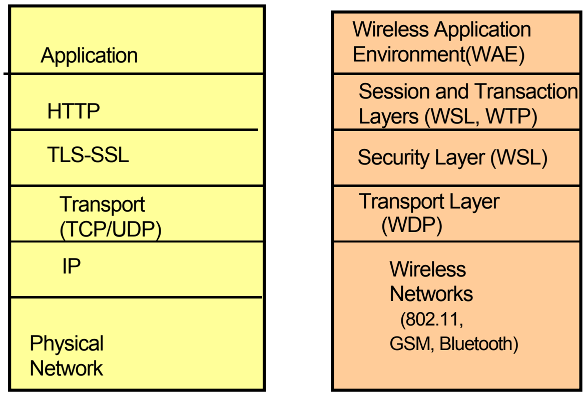


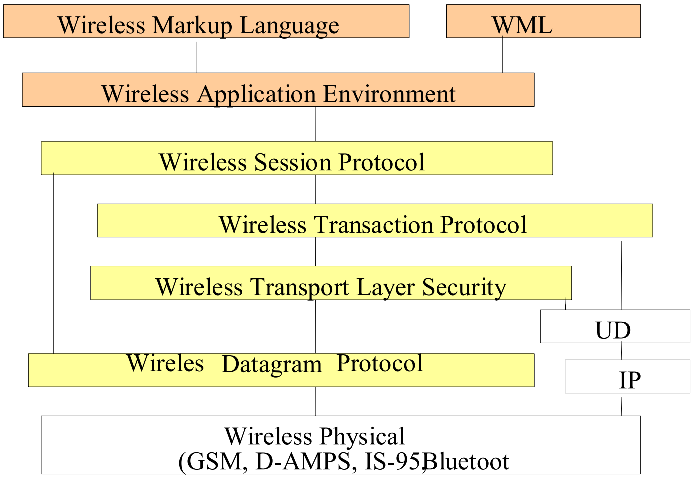
Figure 4-16: WAP stack and the Web.

The WAP standard works with cellular digital packet data (CDPD), code division multiple access (CDMA), time division multiple access (TDMA), global systems for mobile communications (GSM), and other wireless standards. WAP optimizes standard Web protocols, such as HTTP, for use under the low-bandwidth, high-latency conditions often found in wireless networks. The WAP stack introduces a number of enhancements to the session, transaction, security and transport layers. Here are a few examples of these improvements:

- The plain text headers of HTTP are translated into binary code that significantly reduces the amount of data that must be transmitted over the air interface.
- A lightweight session re-establishment protocol has been defined that allows sessions to be suspended and resumed without the overhead of initial establishment. This allows a session to be suspended while idle to free up network resources or save battery power.
- WTP means that a TCP stack is not required in the phone, which allows for significant savings in processing and memory cost in the handset.
- The improvements made in the WAP protocol stack lead to significant savings in wireless bandwidth.

According to some experiments described by the WAP Forum, the WAP protocol uses less than half the number of packets that the standard HTTP/TCP/IP stack uses to deliver the same content. This improvement is of vital importance given the limited wireless bandwidth available.

Let us now take a closer look at the WAP protocol stack. Figure 4-17 shows the interdependencies between the WAP protocols. As can be seen, several WAP protocols can be bypassed. For example, WTLS can be bypassed if security is not needed (I wonder why?) and WTP can directly use the UDP/IP services, thus bypassing WDP. However, the WAP higher-level services such as WSP and WAE are always needed for WAP-compliant systems. Let us now look at the WAP protocol stack in a little more detail.


Wireless Markup Language
WML
Wireless Application Environment
Wireless Session Protocol
Wireless Transaction Protocol
Wireless Transport Layer Security
UD
Wireles Datagram Protocol
IP
Wireless Physical
(GSM, D-AMPS, IS-95Bluetoot

Shaded areas: WAP Specific (light shade: WAP1.2, Darker: WAP 2.0)

Figure 4-17: WAP Protocols

4.5.11.2 Wireless Session Protocol (WSP)

Wireless Session Protocol (WSP) enables services to exchange data between applications in an organized way. Basically, this layer establishes connections between WAP clients and servers. WSP is a transaction-oriented protocol based on the concept of a request and a reply that is quite similar to HTTP. WSP supports connectionless as well as connection-oriented services, and the PDUs (packet data units) exchanged in WSP contain WML, WMLScript, images and headers.

Connection-oriented session services in WSP operate above the reliable transport protocol WTP. These services include the following activities: a) establish reliable session from client to server and release, b) agree on common level of protocol functionality using capability negotiation, c) exchange content between client and server using compact encoding, d) suspend and resume a session, and e) push content from server to client in an unsynchronized manner. Connectionless session services operate above unreliable transport protocol WDP. These services do not establish and maintain sessions between WAP clients and servers.

WSP supports a variety of transaction types such as the following:

- Session establishment – client WSP user requests session with server WSP user.
- Session termination – client WSP user initiates termination.
- Session suspend and resume – initiated with suspend and resume requests – saves state (due to roaming, fading).
- Transaction – data is exchanged between a client and server.
- Non-confirmed data push – used to send unsolicited information from server to client.
- Confirmed data push – server receives delivery confirmation from client.

4.5.11.3 Wireless Transaction Protocol (WTP)

Wireless Transaction Protocol (WTP) provides services to accomplish reliable and non-reliable WAP transactions. It operates over the WDP layer or the optional security layer, WTLS. When operating over WDP, WTP provides reliable transport for the WAP datagram service (an unreliable protocol). WTP provides reliability similar to traditional TCP, but without limitations of TCP that make TCP unsuitable in a wireless network. For example, TCP transmits a large amount of information for each request-response transaction, including information needed to handle out-of-order packet delivery. Since there is only one possible route between the WAP proxy and the handset, there is no need to handle this situation. WTP eliminates this unnecessary information and reduces the amount of information needed for each request-response transaction.

WTP is a lightweight protocol suitable for “thin” clients over low-bandwidth wireless links for e-commerce transactions. The main WTP features are:

- Three classes of transaction service: Class 0 for unreliable invoke message with no result message (unreliable push), Class 1 for reliable invoke message with no result message (reliable push, get acknowledgement), and Class 2 for unreliable invoke message with one reliable result message (supports a request-reply model with some data from server).
- Optional user-to-user reliability: WTP user triggers confirmation of each received message.
- Optional out-of-band data on acknowledgments
- PDU concatenation and delayed acknowledgment to reduce the number of messages sent
- Asynchronous transactions

WTP uses the following PDU types to communicate between WAP peers:

- Invoke PDU – used to convey a request from an initiator to a responder.
- ACK PDU – used to acknowledge an Invoke or Result PDU.
- Result PDU – used to convey response of the server to the client.
- Abort PDU – used to abort a transaction.
- Negative acknowledgment PDU – used to indicate that some packets did not arrive.

4.5.11.4 Wireless Transport Layer Security (WTLS)

Wireless Transport Layer Security (WTLS) is an optional layer that is designed specifically for wireless systems. WTLS is based on TLS (Transport Layer Security), which in turn is based on SSL (Secure Socket Layer). WTLS ensures data integrity, privacy, authentication and denial-of-service protection. For Web applications that employ standard Internet security techniques with TLS, the WAP Gateway, discussed later, automatically and transparently manages wireless security with minimal overhead. WTLS security includes support for:

- Data integrity – ensures that data sent between client and gateway are not modified, using message authentication.
- Privacy – ensures that the data cannot be read by a third party, using encryption.
- Authentication – establishes authentication of the two parties, using digital certificates.
- Denial-of-service protection – detects and rejects messages that are replayed or not successfully verified.

The WTLS Protocol Stack supports two layers of protocols: a) WTLS record protocols that provide basic security services to various higher-layer protocols, and b) higher-layer protocols that provide a variety of protocols such as the Handshake Protocol, the Change Cipher Specification Protocol, and the Alert Protocol. Discussion of these protocols is beyond the scope of this chapter; however, let us look at the Handshake Protocol Exchange for some insights.

Before any application data is transferred, WTLS goes through several phases of the Handshake Protocol Exchange. This protocol is quite complex and is used for client and server to authenticate and negotiate encryption keys and message authentication codes. The phases of this protocol are:

- First phase – used to initiate a logical connection and establish security capabilities (agreement on cryptography and compression).
- Second phase – used for server authentication and key exchange (symmetric, asymmetric keys through server-hello-done).
- Third phase – used for client authentication and key exchange (for server-hello-done, client verifies if certificates are valid, client-key-exchange).

- Fourth phase – completes the setting up of a secure connection (cipher-change).

We will take a second look at WTLS when we revisit WAP security in Section 4.5.13.

4.5.11.5 Wireless Datagram Protocol (WDP)

Wireless Datagram Protocol (WDP**)** is the bottom layer of the WAP stack and is roughly equivalent to the IP layer in the Internet stack. Thus the portability of the WAP protocol relies heavily on this layer. WDP at present supports a very wide range of bearer services (the bearer services are the physical communications between the mobile phone and the Base Stations (the antenna). Examples of bearer services supported by WDP are SMS, CDMA, TDMA, GSM, CSD, and DECT. Thus WAP is supported by many bearers.

WDP is used to adapt the higher-layer WAP protocol to the communication mechanism used between the mobile node and WAP Gateway. WDP hides details of the various bearer networks from the other layers of WAP. Adaptation may include:

- Partitioning data into segments of appropriate size for the bearer
- Interfacing with the bearer network

Wireless Control Message Protocol (WCMP) performs the same support function for WDP as ICMP does for IP (i.e., it provides feedback on problems such as not reaching destination, or router buffer problems). WCMP is used in environments that don't provide the IP bearer and don't lend themselves to the use of ICMP. It is also used by wireless nodes and WAP gateways to report errors encountered in processing WDP datagrams. WCMP can also be used for informational and diagnostic purposes.

4.5.12 WAP Gateway

WAP Gateway is the bridge between two distinct worlds: the Internet model and the wireless phone/data network model. It converts the WAP protocol stack (discussed in previous section) to the Internet stack. The WAP Gateway, shown in Figure 4-18, translates between the Web server and the WAP clients.

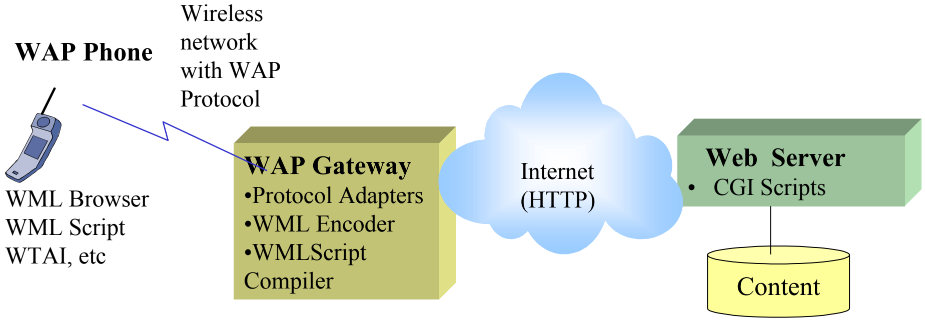


Figure 4-18: WAP Gateway

The WAP specification uses standard Web proxy technology to connect the wireless domain with the Web. The WAP Gateway, a software module, plays a key role in WAP architectures – it supports a thin client model by allowing the handset to be simple and inexpensive. For example, a WAP Gateway can take over directory processing, data conversions, fraud management, and network provisioning. The Gateway thus offloads these computing tasks from the handset. WAP Gateway typically includes the following functionality:

- Protocol Adapters – the protocol adapters translate requests from the WAP protocol stack (WAE, WSP, WTP, WTLS, WDP) to the WWW protocol stack (HTTP, TCP, SSL, IP).

- Content Encoders and Decoders – the content encoders translate Web content into compact encoded formats to reduce the size and number of packets traveling over the wireless data network.

This architecture ensures that mobile terminal users can browse a variety of WAP content and applications regardless of the wireless network they use. Application developers can build content services and applications that are network- and terminal-independent, allowing their applications to reach any audience. The content and applications are hosted on standard WWW servers and can be developed using Web technologies such as CGI scripting, servlets, and Java Server Pages.

The WAP Gateway decreases the response time to the handheld device by aggregating data from different servers on the Web, and caching frequently used information. The WAP Gateway can also interface with subscriber databases and use information from the wireless network, such as location information, to dynamically customize WML pages for a certain group of users.

4.5.13 WAP Security

Applications on the Web typically require a secure connection between the client and the application server. The WAP specification ensures that a secure protocol is available for these transactions on a wireless handset. The Wireless Transport Layer Security (WTLS) protocol is based on the industry-standard Transport Layer Security (TLS) protocol, more popularly known as Secure Sockets Layer (SSL). WTLS is intended for use with the WAP transport protocols and has been optimized for use over narrow-band communication channels.

WTLS ensures data integrity, privacy, authentication and denial-of-service protection. WTLS does not support non-repudiation. The WTLS specification is designed to work even if packets are dropped or delivered out of sequence – a more common phenomenon in some wireless networks. Another issue is that some WTLS messages can be sent without authentication of origin [Van Der Heijden 2000].

Many initial implementations of WAP will have a client – proxy – server model architecture where the proxy can be used to present a simplified view of familiar websites. An important security function performed by a proxy is that it unwraps the WAP WTLS secure data from the client and then re-wraps it into SSL/TLS before passing it to a Web server. For Web applications that employ standard Internet security techniques with TLS, the WAP Gateway automatically and transparently manages wireless security with minimal overhead.

WAP can provide end-to-end security between WAP protocol endpoints. End-to-end security is achieved through two approaches: a) when a browser and origin server directly communicate using the WAP protocol, or b) if a WAP proxy is trusted, or for example, located at the same physical secure place as the secure origin server.

4.5.14 WAP Software Development Kit (SDK), Toolkits, and Infrastructure

Web developers can easily develop WAP applications since the WAP programming model closely follows the existing WWW development model. WML is a tag-based document language specified as an XML document type. As such, existing XML authoring tools, as well as many HTML development environments, can be used to develop WML applications.

Since the WAP specification uses standard HTTP 1.1 protocol to communicate between the WAP Gateway and Web servers, Web developers can deploy their applications on any off-the-shelf Web server. WML developers can use standard Web tools and mechanisms such as Cold Fusion, CGI, Perl, ASP and others to generate dynamic WML applications. Developers can either use separate URLs for their HTML and WML entry points, or use a single URL to dynamically serve either HTML or WML content according to the requestor's browser type.

Although it is possible to translate HTML into WML using an automated system, in practice the best applications use WML to tailor the interface to the specific needs of the wireless user. This allows for the best possible use of the handset features, such as soft keys, and provides the best user experience. The most valuable parts of any Web application are typically the unique content it provides and the back-end database interaction, not the particular HTML that was written to interact with the user. Therefore developing a corresponding WML front-end leverages previous engineering effort, while providing significant user interface benefits.

Although most Web content at present is in HTML, a great deal of future content will be created in XML. As stated previously, content written in XML-defined markup languages can be automatically translated into content suitable for either HTML or WML by using an XSLstyle sheet (this was illustrated in Figure 4-13).

How to get started in WAP? The quickest way appears to get the SDKs (software Development Kits) that include WAP emulators. These emulators allow you to develop and test WAP applications without needing a WAP phone or a wireless network. Such emulating toolkits are available from vendors such as the following:

- Nokia WAP Toolkit that can be downloaded from the Nokia Forum website (http://forum.nokia.com/)
- Phone.com (previously known as Unwired Planet) toolkit, known as UP.Simulator, that can be downloaded from www.phone.com
- Ericsson's *WapIDE* for client and server-side development, that can be downloaded from the website www.ericsson.com/developerszone

Several other toolkits for WAP are also becoming available. Some examples are:

- Melody Interactive (Sweden): *MyWap* WAP site and script tool
- Baltimore Tech: *W/SECURE* WAP security SDK for WTLS
- Oracle (USA): *Project Panama*, translated DB content for WAP devices

Several products to support the WAP Infrastructure are also becoming available. Examples are:

- WAP Gateways – such as Ericsson's *WAP Gateway/Proxy,* Nokia's WAP Server, and the ApiON's *Binnian* (Ireland)
- WAP Terminals and Phones – from Ericsson, Motorola, Nokia, and Samsung
- WAP Microbrowsers – from Nokia, IDO (Japan), and Phone.com (UK)
- Other infrastructure – such as Lucent SMS Center: vmail, fax, paging delivery of WAP messages.

In addition, several **WML editors** have become commercially available to generate WML code. An example is the WML Editor 3.2, which provides Coding Autofills and templates for various WAP conversations. This editor supports a dual view so that you can view the results by PALM OS browser or cell phone micro-browser. Another example is the I-WAP WML Generator that allows you to build a file structure by naming folders and putting them into a tree format. The title of the folder dictates what the "card" will do. For example, the title of folder "contact us at 555-11111" will generate appropriate WML code.

4.5.15 WAP Applications

Several WAP-based banking applications are being developed and will be developed in the future. Typical examples include checking account balance and fund transfers. Examples of other applications are Aspiro, for Stock purchasing; ServiceHub, a mobile field personnel dispatcher; and multi-player wireless games. In addition, a WAP application has been developed to pick songs from a database of music. Another example is www.xwap.net, a pure XML-based news portal designed to deliver content to a variety of wireless devices by leveraging XML and XSL. The portal has been developed by e-content and uses the e-content company's X-WAP (XML Wireless Application Product) that uses XML and XSL to deliver XML-based content to wireless devices.

An interesting example is the Limousine tracking system used by several limo companies. Limousine owners are somewhat suspicious of their drivers – they may be driving around and having a good time instead of waiting to pick up the next customer. Limousine tracking systems are now available that put a GPS device inside a limo. Other examples of general applications are: :

- Intelligent Information Inc.: Personalized info
- MapQuest.com: Travel information (hotels, restaurants, etc.)
- MAZ (Germany): Location-based services
- Peramon (UK): Email notification and access
- Sonera (Finland): News, Account balances, Directory enquiries
- Virtual Inc. (Taiwan): Mobile banking

Many more examples are being reported almost daily. Many WAP applications follow the architecture shown in Figure 4-19 – i.e., the WAP devices access the same content (through the WAP Gateway) that is accessed from the regular Web browsers.

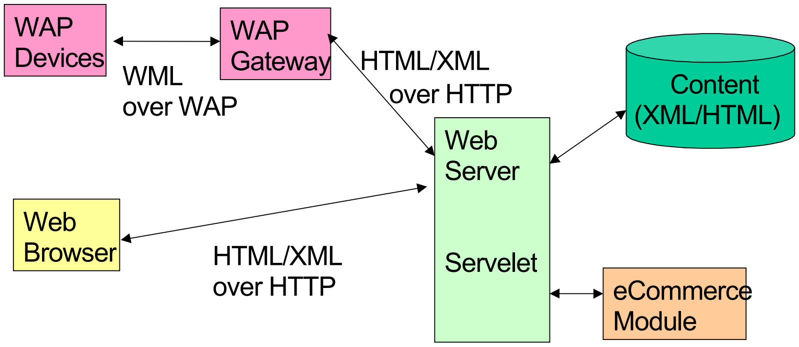


Figure 4-19: Typical WAP Application Architecture

WAP Certification

In March 2000 the WAP Forum launched a certification testing program for all components in the WAP chain. The testing, done in coordination with a third-party testing facility, helps assure future interoperability of WAP-enabled products. Only those products that pass the certification testing process are permitted to use the official certification logo of the WAP Forum, currently part of OMA.

4.5.16 Example of WAP and Bluetooth Together

Some really interesting wireless applications involve WAP and Bluetooth technologies working together. Envision that you arrive at Newark Airport on your way to Denver. After landing, you turn on your mobile phone which supports both WAP and Bluetooth. As you are walking through the airport, a tone from your phone alerts you to a message on the screen. The message states, "Newark Airport Information Service – OK to connect?" The airport devices are talking to your devices. You make the connection and are given information that your Denver flight has been cancelled. Then it proceeds to tell you about nearby bars (you may really need it), duty-free shopping specials, airport map and nearby restaurants.

This is how it works. For the two devices to use WAP protocols over Bluetooth links, you need to make sure that:

- The devices are able to recognize each other as being WAP-capable, giving some indication if they are able to act as a client or proxy/server.
- The devices are able to exchange a short text description of the type of service provided by each such as "Newark Airport Information Service."
- Each device must indicate that it supports the baseline protocol stack as defined by the Bluetooth Specification.
- The proxy/server should indicate if it has overridden the default UDP port assignments for the various WAP protocols.

4.5.17 WAP Summary

WAP is becoming an important element of the wireless and mobile middleware space. WAP penetration is greater in Europe and the Far East than in the US and elsewhere – but is gaining ground rapidly. However, some questions are being raised:

- How many content providers will generate WAP/WML content? How well will automatic HTML/WML translators work?
- Will existing Internet technology mature fast enough to reduce the impact of WAP?
- Will sophisticated e-commerce services for mobile users (e.g., stock purchases and transactions) really become a significant market?

The debate on the success potential of WAP is well underway at present (see, for example, the sidebar, "The WAP Debate." It is too early to tell. However, it is possible that at best WAP will evolve, with several revisions, to address the criticisms. At worst, it will be a good transitional technology that will give us some insights into how mobile commerce will really be supported.

The WAP Forum has done its best to stay abreast and has evolved its standards, plus developed new standards, as new handset technologies, network protocols and Internet standards have developed. But, as stated previously, the WAP Forum is currently part of the Open Mobile Alliance (OMA) and no longer exists as an independent organization. For recent developments in WAP, visit the OMA website at: http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html.

Suggested WAP Links and Books

- WAP Forum – the definitive source of WAP information (white papers, WAP specifications); part of OMA (http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html)
- www. Phone.com: for a popular WAP emulator

- www.ericsson.com\developers – Ericsson WAP Developer's Zone for downloading WAP emulator and other WAP tools
- http://mobileinternet.ericsson.se – Ericsson Mobile Internet
- http://www.ericsson.se/WAP/ – Ericsson WAP
- http://www.motorola.com/mix/ – Motorola MIX
- http://www.forum.nokia.com/wap_developer/index.html – Nokia WAP Developer Forum
- www.nokia.com – Nokia's WAP Resources
- http://www.digimob.com/Digital Mobility – UK and Scandinavia WAP operator
- http://www.seiren.net/ – products including Waplook for Exchange
- http://www.waptunnel.com/index.html – HTML-to-WAP conversion
- http://www.wapmap.com/ – WAPMAP.COM
- *Professional WAP* – book by Arehart et al., Wrox Books, 2000
- *Understanding WAP* – book by M. Van Der Heijden and M. Taylor, Artech Publications, 2000

4.6 i-mode, Wireless Java, MMIT, and BREW

4.6.1 Overview

WAP is not the only approach for mobile application servers. There are several competitors to WAP. Examples of the competitors discussed in this section are i-mode from KDD-DoCoMo, wireless Java that includes J2ME (Java 2 Micro Edition) from Sun, and BREW (Binary Runtime Environment for Wireless) from QualComm. Some other initiatives such as Microsoft's stringer are not discussed here because they are still under development.

These wireless platforms offer different presentation services and are popular in different parts of the world, as shown in the following table. It is difficult to give a detailed comparative analysis of all wireless middleware services due to space limitations. In addition, the marketplace is in flux, with new features being added to the various middleware services on a regular basis. You can find market and technology analysis from sources such as the Giga Information Group (http://www.giga.com/), ResearchPortal.com, MobileWeek, www.obileinfo.com, Gartner and others. The following discussion presents highlights of the various wireless platforms. It is best to visit the sources listed in the following table for additional details and for the latest developments in this highly volatile area.

Table 4-3: Wireless Middleware Highlights

Wireless Platform	**WAP**	**i-mode**	**Wireless Java**	**Microsoft Mobile Internet Toolkit**	**BREW**
Presentation	WML	cHTML	Java	WML, cHTML HTML	BREW
Main popularity	Europe	Japan	US	US and Europe	US
For additional information	www.wapforum.org	www.docomo.com	wireless.java.sun.com	www.microsoft.com	www.qualcomm.com/brew

4.6.2 i-mode[3]

4.6.2.1 What is i-mode?

While WAP (Wireless Application Protocol) is a standards-based approach for wireless Internet, i-mode, a mobile phone service, also offers continuous Internet access. However, this service is restricted to Japan as of now, with some attempts to make it available to other countries. For example, AT&T's M-life initiative is based on i-mode.

I-mode (information-mode) was launched in February 1999 by NTT DoCoMo (DoCoMo in Japanese means "anywhere") – a leading cellular phone operator in Japan. Access to websites compatible with i-mode can be achieved at the touch of a button. In addition to phone calls, you can receive email, exchange photographs, receive news and stock quotes, shop online, receive weather forecasts, play online games and access music files online. Every subscriber is given an email ID, which is his or her cellular phone number with the extension "@docomo.ne.jp."

4.6.2.2 Key Features of i-mode

Components required for i-mode services are: (see Figure 4-20):

- An i-mode cellular phone, i.e., a phone capable of voice and packet transmission along with a browser installed. I-mode phones are specialized phones with larger windows for multimedia.
- I-mode phones are well-known due to their highly graphic user interfaces.
- Content in cHTML, passed to the browser residing in the i-mode phone.
- An i-mode gateway that translates the cHTML content to the back-end server.
- A packet-switching network that is "always on" – i.e., the end user can turn the phone on and receive the email without having to call a number.

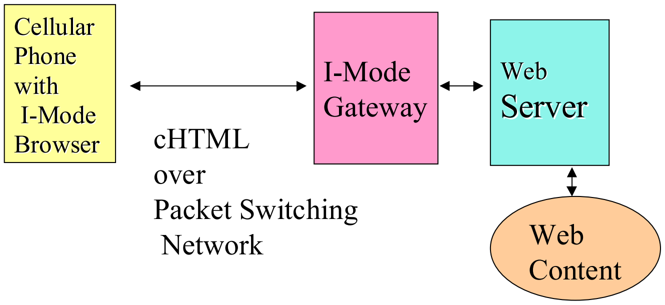

Cellular Phone with I-Mode Browser
cHTML over Packet Switching Network
I-Mode Gateway
Web Server
Web Content

Figure 4-20: i-mode Components

cHTML Markup Language. The markup language that i-mode uses is cHTML (or compact HTML) which is a subset of HTML. It is designed for devices with slower connecting speed. One main difference between the two languages is that some of the more resource-intensive areas of the code such as tables and frames have been bypassed in cHTML. This reduces the download time to mobile devices. An i-mode-enabled website utilizes pages written in cHTML which are easy to create, for Web designers familiar with HTML.

i-mode Gateway: The cHTML content needs to pass through a cHTML gateway before users can access it on their mobile phones using the i-mode browser. When an i-mode-

[3] Also written as I-mode or i-Mode.

compatible wireless device makes a wireless request, the gateway translates this to the server and back from the server to the wireless device.

i-mode Browser. The micro-browser in an i-mode phone is lightweight software designed to run on a handheld device and used to access a cHTML page.

Packet-Switching Network. I-mode uses a PDC-P (Personal Digital Cellular-Packet) method of data transmission over the existing PDC network used for ordinary voice traffic. PDC-P is a packet-switched network that is well suited for wireless communication, mainly in Japan. As discussed previously, in a packet-switched network, data is broken into small units called packets and routed over the Net. This mode of transmission, where communication is broken into packets, allows the same data path to be shared among many users in the network. Currently, the rate at which data is being transmitted is 9.6Kbps. But DoCoMo is developing the W-CDMA (Wideband-Code Division Multiple Access technology) a 3G communication system with speeds of 384 Kbps or more. W-CDMA allows high-speed data transmission of video and large-volume data.

Security. For security, DoCoMo has formed an alliance with Sun Microsystems to incorporate Sun's Java, Jini and Java Card technologies into i-mode cellular phones. This provides security for critical applications like online banking and trading, and greater functionality allowing for game downloading and interactivity with other devices.

4.6.2.3 i-mode Versus WAP

The basic difference between WAP and i-mode is that WAP is standards-based while i-mode is not. WAP has strong industry backing from Motorola, Nokia, IBM, Intel, Microsoft, Ericsson, etc. while i-mode is mainly backed by NTT DoCoMo. Other differences are:

- Markup languages are different. I-mode uses cHTML, which is a subset of HTML, while WAP uses WML, which is a subset of XML. Although cHTML is similar to HTML and easier for Web designers to use, XML is the Internet language of the future as HTML has limited capabilities. XML is more tuned to growing standards than cHTML, so growth will be towards XML. WML might incorporate more and more capabilities from XML and will be far stronger than cHTML.
- The level of graphics supported by the two are different. i-mode supports more graphics than WAP. Although WAP supports some amount of graphics, it is not suited for online gaming supported by i-mode phones. The main reason for this is that WAP uses a circuit-switched network at present, while i-mode uses a packet-switched data network, which is more suited to transferring data than circuit-switched networks are. However, i-mode uses the PDC-P network which is mainly used in Japan, while other parts of the world have circuit-switched networks. This means that even if i-mode were to expand its services it would still have to depend on the existing network in that country; so the major drawback of circuit-switched networks remains.
- Intended audiences may be different. WAP is based on the requirements of people on the move, who would like to access their mail, to get stock quotes, weather reports, and traffic information, or to conduct banking transactions. People on the move would generally not be expected to browse the Net or read large amounts of data or require extensive graphics. Considering the cost of WAP-enabled phones and airtime as of now, it seems to be slotted for the business people, managers, CEOs and the like. I-mode seems to cater to the younger crowd with interest in online games and video. These two audiences could merge as the costs drop and bigger display sizes start appearing on the handsets.

Some mobile computing platforms are bridging the gap between WAP and i-mode. For example, Microsoft Mobile Internet Toolkit generates WML and cHTML so that WAP as well as i-mode devices could use the application.

4.6.3 Wireless Java and J2ME (Java 2 Micro Edition)

4.6.3.1 Overview

Wireless Java is the general umbrella name under which Sun is supporting its Java platform for developing wireless applications. As shown in Figure 4-21, the Java platform itself encompasses many technologies, including Java 2 Standard Edition (J2SE), which provides core applications programming functionality, Java 2 Enterprise Edition (J2EE) technology for writing server-side enterprise applications, and Java 2 Micro Edition (J2ME) technology for writing applications that run on small handheld devices. There are also other technologies that give support for specific wireless communication needs, such as JavaCard. However, in most cases J2ME is considered as the main platform for wireless Java. In reality, wireless Java is not confined to J2ME alone – it includes many other Java technologies such as JavaCard and Personal Java. As a matter of fact, any Java application that runs on a mobile device could be considered as part of wireless Java. For example, a laptop computer can run a J2SE application, making network connections via a wireless Ethernet LAN. Not to generalize the discussion too much, we will confine wireless Java to the following definition:

> Wireless Java = J2ME + JavaCard + Personal Java + any other Java technologies developed specifically for wireless devices (e.g., consumer electronics, cellular phones)

It is interesting to note that Java technologies seem to have come full circle in less than a decade. Java was initially designed in the mid-1990s to support consumer electronics devices because the prevalent programming languages at that time did not fit well in such small devices. Buoyed by its popularity, Java technologies grew into large-scale enterprise applications and resulted in the Sun J2EE (Java 2 Enterprise Edition) with support for Enterprise Java Beans for corporate transactional support. With J2ME, it appears to be solidifying its initial base. The following discussion gives a brief overview of wireless Java. Additional details can be found at the Sun site: wireless.java.sun.com.

		Wireless Java	
	J2SE, J2EE	J2ME	Other Java Technologies for Mobile Devices
Profiles and Configurations	None	Games, MIDP, several others	None
Runtime Environments	Java Virtual Machine (JVM)	Compact and Kilo Virtual Machine (CVM/KVM)	JavaCard Virtual Machine, Others
Devices	PCs, Servers, Workstations	Consumer Electronics, Handhelds	Smart cards, Handhelds, other Mobile devices

Figure 4-21: Java Platforms (for a more detailed view, see wireless.java.sun.com)

4.6.3.2 Why Use Java for Wireless Applications

As we have seen, several development environments for mobile applications such as WAP-WAE and i-mode are currently available. Why should you use Java as the platform to develop wireless applications? Some people argue that Java was designed for small portable devices for consumer electronics – thus the question is invalid. The other reasons to favor Java for mobile applications are:

- **Java language can handle sophisticated and robust programs.** Developers can create more sophisticated applications that interact with back-end systems using Java technology, compared with the browser-based environments (e.g., WAP or i-mode).
- **Java applications can be written once and run anywhere.** Applications written in Java can run on similar types of systems (servers, desktop systems, mobile devices) independent of the underlying operating system and processor. Assuming a JVM (Java Virtual Machine) is available, the same Java app can run on a Nokia Communicator running the EPOC operating system, a Compaq iPAQ running PocketPC, a PDA powered by the Linux operating system, or on mobile phones on the variety of mobile phone processors and operating systems.
- **Java applications are network aware.** Java applications can also exchange data with a back-end server over any network protocol such as TCP/IP, WAP, i-mode, and different bearers, such as GSM, CDMA, TDMA, CDPD, Mobitex, and others.
- **Java addresses safety and security concerns.** Java code always executes within the confines of the Java Virtual Machine (JVM) and limits the application crashes to possible death of the Virtual Machine. Your cell phone or PDA would not crash due to a Java application crash. For security, Java uses what is known as a **sandbox**. A sandbox surrounds the Java code so that Java applications cannot access system resources outside of the sandbox. Java also supports the standard encryption solutions (e.g., SSL) on a packet-based network.

4.6.3.3 A Closer Look at Wireless Java and J2ME

As mentioned previously, the Java Platforms contain core and optional packages that are defined in part by a specific set of APIs. The main platforms are the Java 2 Platform for Enterprise Edition (J2EE) that provides a range of tools for developing enterprise applications, the Java 2 Platform Standard Edition (J2SE) for standard Java applications, and the Java 2 Micro Edition (J2ME) that is a modified and independent version of J2SE for small devices. Wireless Java is the following family of technologies:

- J2ME technology is a platform for writing applications that run on small handheld devices. We will discuss this in more detail because wireless technology is the main focus of this edition.
- JavaCard platform lets developers write wireless or wired smart card applications that are standardized on a common smart card platform.
- PersonalJava technology is designed for network-connectable consumer products in the areas of communication, entertainment, and mobile computing.
- Java Phone is an API that is an extension of the PersonalJava platform targeted at Internet phones and wireless smart phones.

Java Docs, although not connected directly to wireless Java, are a valuable resource for Java programmers. Java Docs describe the specific syntax for a set of APIs associated with a given platform or edition and thus are the main programmer interfaces for developing wireless or wired Java applications.

Let us now look at the key players in the Wireless Java family in some detail. The attention is focussed on J2ME and Java card shown in Figure 4-22 (Personal Java and Java Phone are older and not that much in the limelight). As shown in Figure 4-22, J2ME itself can be subdivided into high-end consumer electronic devices and handheld devices. It can be seen that there are several layers that define the wireless applications and the technology stack to support these applications (by now, you should be used to these types of stacks).

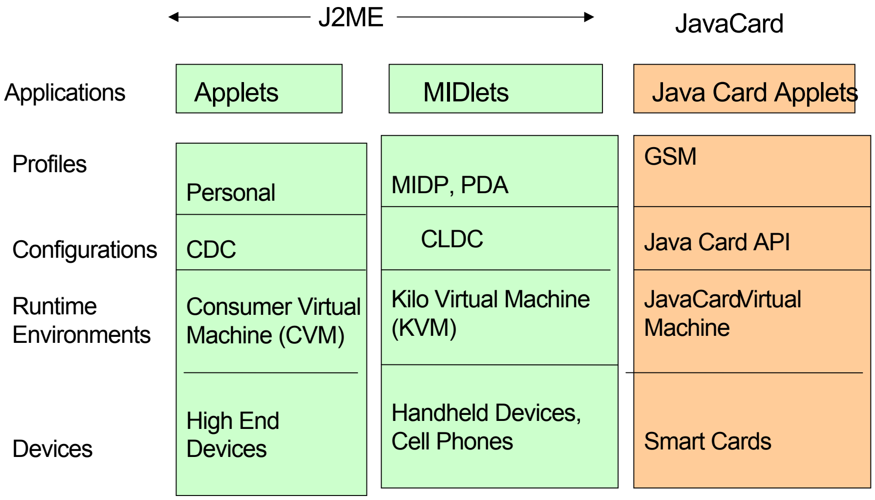


Figure 4-22: J2ME Stack

Application Software. J2ME's applications are developed as applets or MIDlets. A **J2ME applet** runs in a Compact Virtual Machine (CVM) installed on a high-end wireless device, and has an inherent life cycle that is managed by the Application Manager also running in the CVM. The Application Manager is responsible for starting, initializing, cleaning up after, and stopping the applet. A **J2ME MIDlet** (Mobile Information Device-let) runs in a KVM (Kilo Virtual Machine) installed on a handheld wireless device. MIDlets are managed by the Application Manager also running in the KVM. The Application Manager is, once again, responsible for starting, initializing, cleaning up after, and stopping the MIDlet. We should also mention the **Java Card's applets** that run in a Java Card VM that is installed on a smart card. These applets are managed by the Java Card Runtime Environment (JCRE) that also run in the JavaCard VM. The JCRE is responsible for starting, initializing, cleaning up after, and stopping the JavaCard applet.

Profiles: A profile defines the set of APIs available for a particular family of devices. Profiles are built on top of configurations. Some of the J2ME profiles are:

- **Personal:** The J2ME Personal profile repackages the PersonalJava Application Environment. It provides the J2ME specification for devices that need a high degree of Internet connectivity.
- **Game:** The Java Game Profile supports game development targeting high-end consumer game devices and desktops.
- **PDA:** The J2ME Personal Digital Assistant (PDA) profile provides user interface and data storage APIs for PDAs. It uses CLDC (see below).
- **MIDP:** The J2ME Mobile Information Device Profile (MIDP) provides a standard platform for small, resource-limited, wireless-connected mobile information devices. MIDP requires CLDC. MIDP is the first finished profile and has the first installed base of devices.

Configurations: A configuration defines the minimum set of class libraries available for a range of devices. Configurations include specific virtual machines for full-featured as well as small devices.

- **CDC**: The J2ME Connected Device Configuration (CDC) supports a full-featured runtime environment and is intended for larger wired or wireless consumer electronics devices. It uses the CVM.
- **CLDC:** The J2ME Connected Limited Device Configuration (CLDC) is intended for small wireless devices with simplified user interfaces, minimal memory, and intermittent network connections. It uses the KVM.

Runtime Environments and Virtual Machines: Programs written in Java Programming Languages are interpreted on top of the Virtual Machines (VMs). Because application code runs inside a VM, code that is downloaded from the network can be run safely. The most familiar virtual machine for Java is JVM (Java Virtual Machine), used for the J2EE and J2SE platforms. To support J2ME, which is intended for handheld devices, there are two VMs: a Kilo Virtual Machine (KVM) for small handhelds and a Compact Virtual Machine (CVM) for larger consumer electronics devices.

Devices: Sun is trying to cover all the existing devices as much as possible. Each Java Edition (J2SE, J2EE, J2ME) is created to give specific support for specific devices, since each device has its own characteristics. For example, the J2EE platform is designed for server-side programs that reside on servers and communicate with wired and wireless clients and access databases. The following devices are supported for wireless Java through J2ME and its friends (you can think of it as a “Friends and Family” program):

- **Consumer Electronics:** There are many consumer devices with embedded chips. These range from small devices that you can carry around in your hand or pocket to large devices operating in houses.
- **Handsets:** These include handheld devices such as Personal Digital Assistants (PDAs), organizers, and mobile phones.
- **Smart Cards:** A smart card is a plastic card, about the size of a credit card, with an embedded microprocessor. It can store and process data, but requires no power because it works in tandem with a smart card reader or other Card Acceptance Device (CAD).

There is a great deal of activity in wireless Java, especially in J2ME. For the latest developments, see the Sun site: wireless.java.sun.com. Several mobile application servers such as the Aligro Mobile Application Server (www.Aligro.com) are based on wireless Java.

4.6.4 Microsoft Mobile Internet Toolkit (MMIT)

The Microsoft Mobile Internet Toolkit (MMIT) is an extension of the .NET framework for building Wireless applications. We will briefly discuss .NET in a later chapter (Chapter 11), but briefly, .NET provides a set of common services which can be invoked from a number of entities (people, machines, programs) over standard Web protocols. In particular, Web Services (discussed in the previous chapter) is at the core of .NET – the services defined by .NET are Web Services that can be invoked through WSDL, UDDI, etc. from mobile or stationary devices.

MMIT has two interesting features. First, it automatically generates code (WML, cHTML, and HTML) for different type of mobile devices. Thus different applications do not need to be built for different mobile devices. Second, it is built around .NET and Visual Studio – popular platforms for application development at present – thus many developers can leverage their existing desktop skill set to produce mobile applications. A conceptual view of MMIT is given in Figure 4-23.

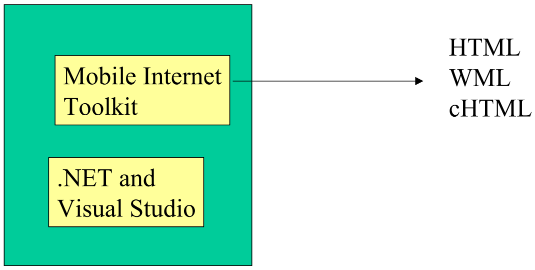


Figure 4-23: Microsoft Mobile Internet Toolkit

MMIT consists of three major tools:

- **Mobile Web Forms Controls.** This tool generates markup language for different devices. These are server-side controls that provide user interface elements (list, command, call, etc.) that are used to generate ("render") the correct markup for the device that makes the request. This code renders on different devices. The code generated can be WML for a cell phone running a WAP browser, cHTML for an i-mode phone, and HTML for a Pocket PC. The generated markup can be customized for a specific device.
- **Mobile Internet Designer.** This tool works with the Visual Studio .NET integrated design environment (IDE) to provide a drag-and-drop mobile development environment. This tool allows you to create and develop your mobile application in the same way that you would develop a Windows Forms or Web Forms application. The Mobile Internet Designer makes it fast and easy to build and maintain mobile Web applications. In addition, it enables today's desktop developers to quickly learn how to create mobile applications by using Visual Studio .NET.
- **Device Capabilities.** This tool captures information about the display capabilities of the target mobile device and stores it in machine.config file. This information is used by the mobile web form controls to generate appropriate markups for the mobile devices. Typical information about a device includes the Markup language (HTML, WML, cHTML), the type of browser, number of display lines, and screen size.

The Mobile Internet Toolkit also includes device adapters that read the device capabilities information in the machine.config file and optimize the markup accordingly. You can create new device adapters for additional devices by using the device capabilities mechanism, thus increasing the number of devices supported by the MMIT.

From an operational point of view, a device accessing the mobile application launches several processes on the Web Server. First, device capability is detected and information such as browser type, supported mark-up language, etc. is collected. Then the response is generated based on the device type. MMIT currently supports the most common device types using cHTML, WML, and HTML. This rendering is possible with a machine.config file. The client device type is matched against the file and its capabilities are automatically associated with it.

To use MMIT, first the Microsoft .NET Framework and then the Microsoft Mobile Internet Toolkit has to be installed. A great deal of information about MMIT for developers, including download and documentation, is available at www.microsoft.com/downloads/details.aspx and www.asp.net/mobile/.

4.6.5 QualComm's Binary Runtime Environment for Wireless (BREW)

QualComm created the BREW platform to provide application developers with a platform in which to develop their products, and to allow the same features as J2ME, including cross-device platform capabilities. The problem is that BREW-enabled applications are mainly dependent on devices with QualComm CDMA chipsets running on a CDMA network. So what is so good about BREW?

BREW addresses a problem that has hampered the wireless industry for a while – nearly every cell phone sold today is expected to be thrown away. Handset manufacturers must load each phone's applications onto the handset at the factory. Further, each application must be custom-built for each individual handset. So how can the applications installed on your current phone be transferred to your new one quickly when you toss the current phone away? Without a way to install new applications on existing phones or to transfer existing applications to new phones, the carriers face a difficult and time-consuming business problem. BREW is intended to simplify application development on CDMA handsets, albeit from QualComm, and to make it possible for end users to download new applications as binary code. In addition, BREW encompasses a distribution method for certifying, downloading, and charging that could speed up the introduction of wireless applications.

The main feature of QualComm's BREW platform is that it is very thin, with a very small footprint. It runs on top of .NET Frameowrk and Visual Studio. BREW code is many times smaller than other platforms due to the sole focus of QualComm on the wireless handsets (rather than a scaled-down version of a product developed for PCs and PDAs). Thus it is more efficient than J2ME code. Due to its focus on binary code, it enables rapid development of a wide variety of downloadable applications. Once developed, these applications can be quickly downlaoded whenever you change your phone or want to get new apps on your existing phone. However, there are several limitations of BREW also. First, BREW applications are at present dependent on QualComm's CDMA chipsets running on CDMA networks. QualComm claims that it intends to branch out to other products also. Second, unlike Java, BREW is not a standard – it is a proprietary platform from QualComm. Finally, QualComm's competitors are not likely to support a QualComm initiative which will increase QualCom marketshare.

Despite some questions about BREW, carriers in the US, Japan, and Korea are rolling out or are planning to provide application download services using BREW. Examples of the carriers that have adopted BREW are Verizon (US), Leap Wireless (US), KTF (Korea), and KDDI (Japan). Additional information about BREW can be found at www.qualcomm.com/brew/.

Time to Take a Break

✓ • Local Services and Wireless Middleware

✓ • WAP, i-mode, J2ME, MMIT, BREW

• Voice XML, Examples and Case Studies

4.7 Voice Communications – Voice Browsers and Voice XML

4.7.1 Overview

Developments such as WAP and i-mode concentrate on keyboard-display for cellular phones and other wireless devices. However, it is much more natural in many cases to keep your eyes and hands free by just talking to your telephone. Use of voice communications for conducting business is common for companies. A menu traversed using the phone's keypad is well known to most of us thanks to the IVR (Interactive Voice Response) units that make us all walk through scores of voice menus before we talk to a human being.

Development of voice browsers is key to voice communications (see Section 4.7.2 for a discussion of voice browsers). In addition to the voice browser, a collection of markup languages are needed to represent different aspects of voice communications. The World Wide Web Consortium's Voice Browser Working Group is defining several markup languages for applications supporting speech input and output. These markup languages are intended to enable IVR applications across a range of hardware and software platforms. Specifically, the Working Group is designing markup languages for dialog, speech recognition, grammar, speech synthesis, natural language semantics, and a collection of reusable dialog components. These markup languages make up the ***W3C Speech Interface Framework***. Details about this Framework can be found at the W3C website (http://www.w3.org/).

Although many markup languages have been introduced for voice communications, the best-known so far is VoiceXML, created by the VoiceXML Forum (http://www.voicexml.org/). We discuss VoiceXML in Section 4.7.3. The main goal of VoiceXML is to simplify the task of building IVR applications by automating the menu selection process. How does the W3C Voice Browser Working Group relate to the VoiceXML Forum? Basically the two groups work very closely with each other. The VoiceXML Forum developed the dialog language VoiceXML, which it submitted to the W3C Voice Browser Working Group. The Voice Browser Working Group used those specifications as a model for the Dialog Markup Language. In addition, a great deal of discussion is going on at present between WAP, VoiceXML Forum, and the W3C Voice Browser Working Group to integrate different views.

4.7.2 Voice Browsers

According to W3C, "a ***voice browser*** is a device (hardware and software) that interprets voice markup languages to generate voice output, interpret voice input, and possibly accept and produce other modalities of input and output." The voice browsers enable users to speak and listen using a telephone or cell phone to access information available on the World Wide Web. These voice browsers accept spoken words as input, and produce speech or replay prerecorded speech as output. In addition to cellular phones and PCs, voice browser hardware processors may be embedded into appliances such as VCRs, TVs, radios, alarm clocks, soda machines, remote controls, ovens, refrigerators, coffeepots, doorbells, and practically any other electronic or electrical device. Thus voice browsers can be used in an extremely diverse array of mobile as well as non-mobile devices to access Web information through speech.

Voice browsers allow people to access the Web using speech synthesis, prerecorded audio, and speech recognition. Due to the popularity of the Web as a vast information source, voice access to this information creates interesting applications. Possible software applications include a wide range of voice-based access to information and services such as automated telephone order taking and tracking, weather, news, airline arrival and departure information, cinema and theater booking services, home banking services, address and telephone lists, to-do lists, shopping lists, and sending and receiving voicemail messages).

A voice browser handles scripts written using voice markup languages. The W3C Voice Browser Working Group (www.w3.org/voice/) was chartered by the World Wide Web Consortium (W3C) in May 1999 to prepare and review markup languages that enable voice browsers. Participants of the Group include the four founding members of the VoiceXML Forum, plus many companies in telephony and speech recognition applications, web portals, telcos and appliance manufactures. The Group has developed a ***Dialog Markup Language (Dialog ML)*** that is based on the Voice XML language. Attempts to combine the Dialog ML with WML (Wireless Markup Language) are under way. In addition, W3C has developed Synchronized Multimedia Integration Language (SMIL, pronounced "smile") as a presentation language that coordinates the presentation of multiple visual and audio output to the user. How does it relate to Dialog ML? Dialog Markup Language coordinates input from the user and output to the user. Eventually the presentation capabilities of SMIL should be integrated with the output capabilities of Dialog Markup Language.

Can HTML be used for voice browsers instead of inventing a new language for voice-enabled web applications? The short answer is no. HTML was designed as a visual language with emphasis on visual layout and appearance. Voice interfaces are much more dialog-oriented, with emphasis on verbal presentation and response. Rather than adding HTML with additional features and elements, new markup languages were especially designed for speech dialogs. Thus HTML is not supported by W3C-specified voice browsers. However, some vendors are creating voice-enabled HTML browsers that produce voice instead of displaying text on a screen display. A voice-enabled HTML browser must determine the sequence of text to present to the user as voice, and possibly how to verbally present non-text data such as tables, illustrations, and animations. A voice browser, on the other hand, interprets a script which specifies exactly what to verbally present to the user as well as when to present each piece of information.

Voice browsers enable Web-based services from any phone and any device with a voice browser hardware chip. Voice browsers can be a key component of the next generation of call centers and Web portals that combine Internet access with phone access. Users can choose whether to respond by a key-press or a spoken command. Basically, the Web content can be created in XML and can be rendered through XSL for voice browsers, HTML browsers, or WAP browsers (Figure 4-24).

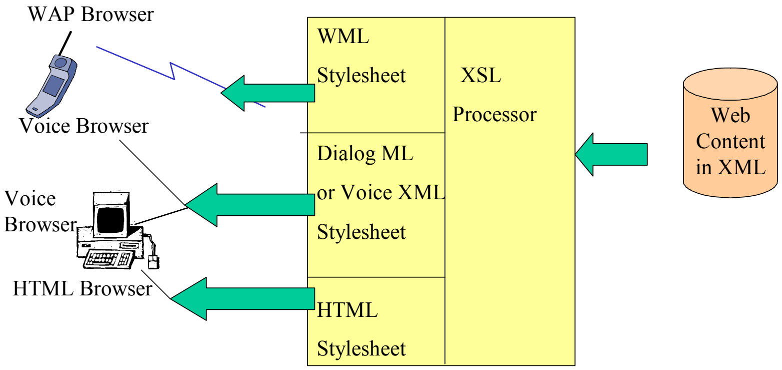


Figure 4-24: Integrating Voice with Other Media

Finally, a word about multimodal (i.e., voice plus data) interfaces. We are moving towards multimodal interfaces which will need to adjust themselves to the user's current environment and functional abilities. Work on multimodal browsing will address this in the context of user and device profiles.

4.7.3 VOICE XML

4.7.3.1 Overview

A voice markup language is needed to support voice browsing. Different speech companies have historically created their own voice markup languages with different names. For example, IBM created a language called SpeechML. AT&T and Lucent both developed PML (Phone Markup Language), and Motorola's original language was VoxML. HP Research Labs created TalkML. The World Wide Web Consortium Voice Browser Working Group has specified Dialog ML.

The Voice Extensible Markup Language (VoiceXML), developed by the VoiceXML Forum (www.voicexml.org), is the best-known markup language at present. The VoiceXML Forum was formed by IBM, AT&T, Lucent, and Motorola (VoiceXML was briefly known as VXML when initially created). The W3C Dialog ML uses VoiceXML as a model. The VoiceXML is designed for creating audio dialogs that feature synthesized speech, digitized audio, and recognition of spoken and digitized voice mail. Its major goal is to bring the advantages of Web-based development and content delivery to interactive voice response applications.

The discussion of VoiceXML in this section is admittedly brief. For more information, refer to the VoiceXML Forum site (www.voicexml.org) – the VoiceXML specifications on this site are surprisingly readable.

4.7.3.2 Key Ideas Through Simple Examples

Example 1: Here is a simple Voice XML example for "Hello World":

```
<?xml version=“1.0”?>
<vxml version=“1.0”>
 <form>
  <block>Hello World!</block>
 </form>
</vxml>
```

The top-level element <vxml> is mainly a container for *dialogs*. Form elements present information and gather input. This example has a single form, which contains a block that presents "Hello World!" to the user. Since the form does not specify a successor dialog, the conversation ends.

Voice XML supports two types of dialogs: *forms* and *menus*. Form elements present information and gather input; menus offer choices of what to do next. We used form in the previous example. The next example shows the use of both dialog types.

Example 2: This VoiceXML code asks the user for a choice of car rentals and then submits it to a server script:

```
<?xml version=“1.0”?>
<vxml version=“1.0”>
 <form>
 <block> Welcome to Friendly Car Rental </block>
 <field name=“car”>
   <prompt>Would you like to rent Honda,
    Corsica, Buick, or Cadilac?</prompt>
   <grammar src="rentals.gram"
    type="application/x-jsgf"/>
 </field>
 <block>
   <submit next=
    "http://www.rentals.example/car2.asp"/>
 </block>
 </form>
</vxml>
```

This sample Voice XML will generate the following interaction:

C (computer): Welcome to Friendly Car Rental
C: Would you like to rent Honda, Corsica, Buick, or Cadilac?
H (human): Camry.
C: I did not understand what you said.
C: Would you like to rent Honda, Corsica, Buick, or Cadilac?
H: Buick
C: (continues in document car2.asp)

Let us briefly go through how this will work. Basically VoiceXML concentrates on what to do with the content instead of how to convey the content. A **field** in VoiceXML is an input field. The user must provide a value for the field before proceeding to the next element in the form. In this example, a user must provide a valid value for the car. Each dialog has one or more **grammars** associated with it which define the allowable inputs submitted by the user. Grammars provide a context and simplify the task of voice recognition enormously. Some grammars are pre-defined (e.g., the currency grammar) but you can define your own grammar also. In the car rental example, the rentals.gram is the grammar that checks against valid inputs that may look like this:

```
Cadillac | Blue, Green;
Corsica | Red, Silver, Black;
Buick | Black;
Honda | Grey, White;
```

This grammar shows the valid car entries and the colors available. Notice that Camry is not a valid input because it does not exist in the grammar. Once a valid entry is found, then the dialog could ask the user to choose colors of rental cars. The "type" statement just indicates the type of grammar. For example, "jsgf" indicates Java Speech Grammar Format – a specific grammar format. Once the correct input has been received the control is passed to the car2.asp program for further processing. VoiceXML defines a mechanism for handling ***events*** that are raised under a variety of circumstances, such as when the user does not respond.

4.7.3.3 Architectural Model

Figure 4-25 shows the architecture of VoiceXML. At the core of VoiceXML is the VoiceXML browser that consists of an interpreter, a context, and an implementation platform (e.g., a cellular phone). The browser receives VoiceXML documents from the voice document server and sends requests to the document server. The browser also uses two external components: an automatic speech recognition (ASR) system that parses the speech and a text-to-speech (TTS) processor for conversion between voice and. data.

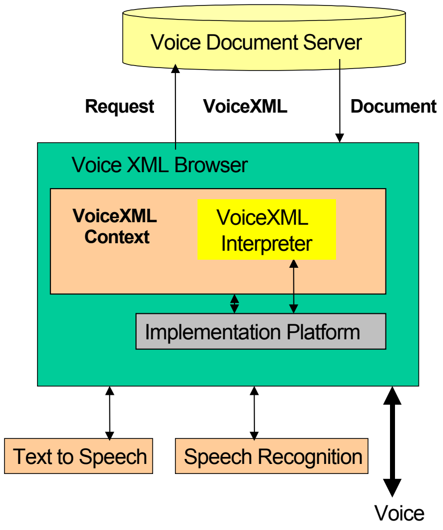


Figure 4-25: VoiceXML Architecture

Let us look through the internal workings of a browser. The document server (e.g. a web server) processes requests from the browser VoiceXML Interpreter, through the VoiceXML interpreter context. The server produces VoiceXML documents in reply, which are processed by the VoiceXML Interpreter. The VoiceXML interpreter context may monitor user inputs in parallel with the VoiceXML interpreter. For example, one VoiceXML interpreter context may listen for a special escape phrase that activates a personal assistant, and another may detect an incoming call. The implementation platform generates events in response to user actions (e.g., spoken or character input received, or disconnection) and system events (e.g. timer expiration).

VoiceXML enables integration of voice services with data services using the familiar client-server paradigm. A voice service is viewed as a sequence of interaction dialogs between a user and an implementation platform. The dialogs are provided by document servers, which may be external to the implementation platform. Document servers maintain overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document specifies each interaction dialog to be conducted by a VoiceXML interpreter.

The VoiceXML browser basically converts VoiceXML to voice and resides in end devices. An alternative is to build a VoiceXML gateway that converts VoiceXML to voice (this gateway acts as a large browser). Figure 4-26 shows a physical architecture for a voiceXML application – it shows the two different ways of producing voice from VoiceXML: a) through a VoiceXML browser that produces voice directly from VoiceXML, and b) through a VoiceXML gateway that translates VoiceXML to voice. In this example, the VoiceXML browser resides on a PC and converts VoiceXML to voice for the PC speakers. Thus a VoiceXML gateway is not needed on the PC. The main advantage of the VoiceXML gateway is that it can support many voice devices including old telephones .

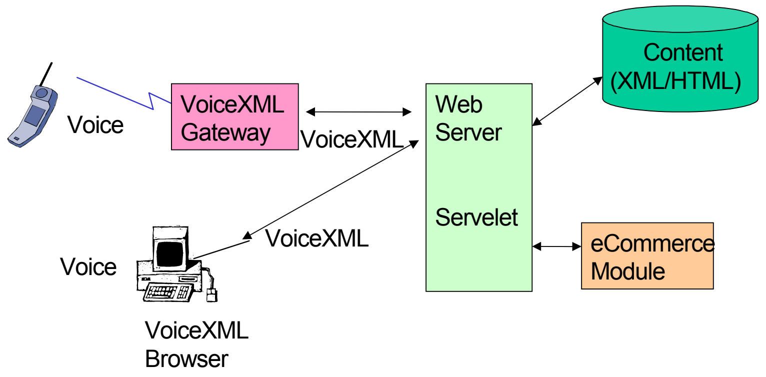


Figure 4-26: VoiceXML Physical Application Architecture

Figure 4-27 shows a more detailed view of the Voice XML gateway. The gateway consists of a voice browser, a text-to-speech (TTS) processor and an automatic speech recognition (ASR) unit. Let us use the car rental example to illustrate how this gateway could be used. The steps to use this gateway are:

1. The VXML document described earlier is stored on the website
2. The user calls the site, the car rental site, by typing the URL of the site.
3. The site server sends VXML the following welcome messages to the Gateway:
Welcome to Friendly Car Rental
Would you like to rent Honda, Corsica, Buick, or Cadillac?
4. The voice browser at the Gateway receives this VXML and uses TTS to translate it to speech that is sent to the user. The user gets voice "welcome … " and "would you like … " messages.
5. The user listens to these messages and says the choice "Camry."
6. This voice response is sent to the Gateway that invokes ASR to convert voice to VXML text.
7. The browser now sends VXML to the site. The site now looks into grammar and generates a VXML response that is sent to the Gateway.
8. The VXML is translated to voice by the Gateway and dialog continues by using the steps 4-7.

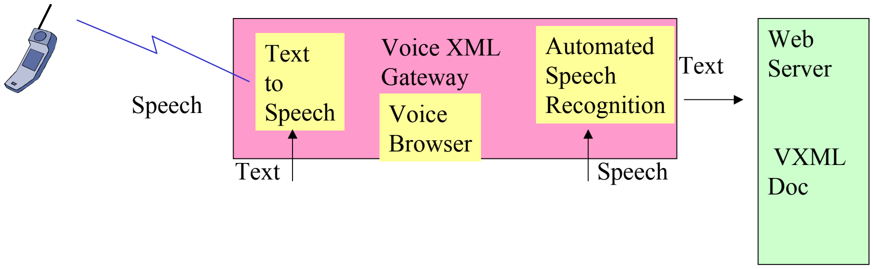


Figure 4-27: Voice XML Gateway

4.7.3.4 Implementations

Several venders have implemented VoiceXML 1.0. Examples of the implementations are:

- Motorola has the Mobile Application Development Toolkit (MADK), a freely downloadable software development kit that supports VoiceXML 1.0 (as well as WML and VoxML). See http://www.motorola.com/MIMS/ISG/spin/mix/.

- Tellme Studio allows anyone to develop their own voice applications and access them over the phone just by providing a URL to your content. Visit http://studio.tellme.com/ to begin. The Tellme Networks voice service is built entirely with VoiceXML.
- IBM Voice Server SDK Beta Program is based on VoiceXML Version 1.0 available at http://www.alphaworks.ibm.com/tech/voiceserversdk.
- Nuance offers graphical VoiceXML development tools, a Voice Site Staging Center for rapid prototyping and testing, and a VoiceXML-based voice browser to developers at no cost. See the Nuance Developer Network at http://extranet.nuance.com/developer/ to get started.

4.8 Examples and Case Studies

4.8.1 Texas Instruments' OMAP – Platform for Building 3G Applications

Texas Instruments has developed OMAP^{TM} (Open Multimedia Application Platform) specifically for the development of 3G wireless applications on handsets. Figure 4-28 shows the architecture of OMAP. The system uses a dual-processor architecture for two different type of processes. One side is the ARM9 processor that can be used by many popular mobile device operating systems such as Symbian, WinCE, and Linux. The other side is the C55xx digital signal processor (DSP) used in many cellular handsets. The inter-processor communication (IPC) is handled through hardware components that support multiple data transfers and multiple messaging capabilities.

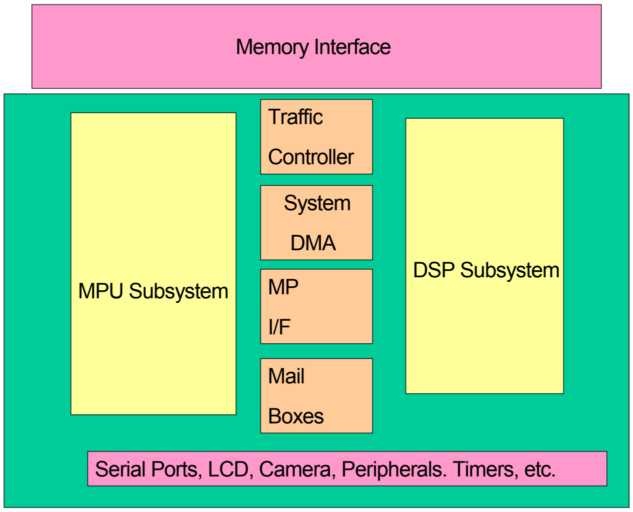


Figure 4-28: Texas Instruments OMAP

The core of the OMAP system is the System Dynamic Memory Access (DMA), which has multiple data paths to various memories and peripherals. The purpose of DMA is to transfer large blocks of data between various memories and peripherals for both internal and external resources. Complementing the System DMA are various hardware blocks. The Traffic Controller block helps routing the data traffic as well as arbitrating the traffic paths based on a priority scheme. The Mail Boxes is a window for either processor to send short data packages to each other. This mechanism is used for inter-processor communication via interrupts. OMAP is a system-on-chip (SoC) device. Many peripherals are built in, like serial ports, Liquid Crystal Display (LCD) controller, real time clock (RTC), and others. There are over 50 various peripherals built into the same SoC.

To make the task of software development easy, TI has provided two different technologies to help the system designers:

- DSP-Link: A simple IPC mechanism which allows any generic host processor to communicate with any TI DSPs by using APIs for data transfer and message passing. System designers can also build system intelligence and system management on top of the DSP-Link protocol.
- DSP-Bridge: A more sophisticated IPC mechanism that runs only on OMAP and provides a middleware system solution that provides many APIs to automatically use OMAP hardware features. It is designed for mobile device OSs: Symbian, WinCE, Linux, and Palm.

To simplify the implementation of software that runs on this device, an operating system is used to schedule various applications running with different priorities. For example, phone conversations may have a higher priority than downloading a picture. Since OMAP has two processors (ARM9 and C55-DSP), it also has two different operating systems. ARM side OS, as stated above, can be Symbian, WinCE, or Linux. While the DSP side uses a DSP-BIOS, an operation system created by TI supports the DSP. Figure 4-29 shows a life cycle of an application under a multi-tasking OS environment supported by OMAP. The OS treats each application (e.g., phone conversations, playing MP3, using digital camera, etc.) as a thread with the following three phases;

- Create phase allocates system resources for the application.
- Execute phase runs the application at its appropriate priority.
- Delete phase returns resources back to the system.

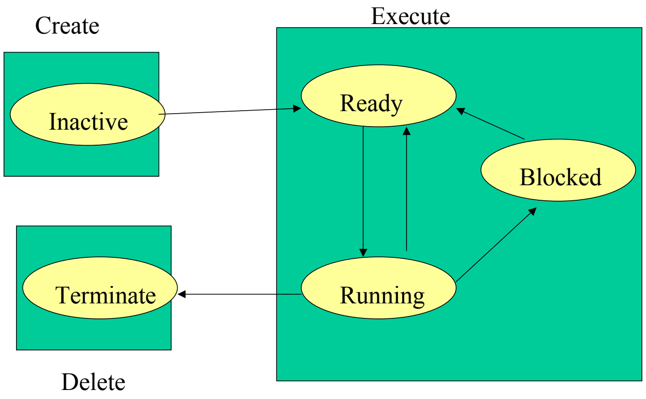


Figure 4-29: States in OMAP

Consider, for example, a user wanting to play an MP3 music file. As the user launches the MP3 application, the OS creates a new thread as an “inactive state.” During this state, the OS allocates system resources required for the application to run. After the user selects the MP3 file and clicks on the play button, the thread moves to the execute phase as a “ready state.” While in the execute phase, the thread needs to have two main resources: space to hold the data buffer from the input device and the output buffer to be sent to the output device, and time to execute the processing. In a handset with multiple applications running simultaneously, space and time are needed for each thread to become "ready," otherwise they go into a "blocked" state. Whenever data and time become available to a blocked thread, the OS moves the thread to the "ready state." If no higher priority tasks are ready, then the MP3 thread goes into "running" state. The OS monitors and schedules the tasks between various states. When the user quits the MP3 application, then the thread is moved to the delete phase. During the delete phase, the OS takes back all resources used by the application and then deletes the thread.

The DSP-Link and DSP-Bridge APIs provide mechanisms to set up and efficiently manage the system memory and other resources. Mobile devices typically are small, thus there is no room for large chunks of main memory. A typical system has 16 to 64 MBytes of SDRAM (synchronous dynamic random access memory). To prolong battery life, a mobile device generally does not have a hard disk drive; instead it uses 16 to 128 MBytes of flash memory to store compressed data and applications. Applications running from external memory run slower than applications running form internal memory. Running from internal memory speeds up the application processing time and reduces the device's power consumption. Thus it is highly desirable to run as many applications as possible with internal memory.

Sources:

- Texas Instruments Wireless OMAP website – http://www.ti.com
- S. Tran and P. Arasalingam, *Multi-media Implementation on 3G Hand set*, Project Report, University of Pennsylvania

4.8.2 Example: Platforms and Middleware for Wireless Sensor Networks (WSNs)

WSNs, as stated in Chapter 2, are formed when tiny sensors communicate with each other through wireless networks. WSNs shuffle the information collected through thousands of sensors and transfer it to higher level nodes with data aggregation, data storage and processing capabilities. WSN applications need specialized platforms that include the middleware services, local services, and network transport services. The main purpose of platforms for sensor networks is to support the development and execution of sensor-based applications. However, this is a non-trivial task, as WSNs have some unique characteristics:

- Sensor nodes are tiny devices with very limited energy they can store. This translates into resources (CPU performance, memory, wireless communication bandwidth and range).
- Sensor nodes do not have IP addresses. Thus they are not directly addressable by IP routing.
- Nodes are subject to failures due to depleted batteries, captures, or environmental influences.
- There is a high degree of dynamics in WSNs such as frequent network topology changes and network partitions (i.e., the network is subdivided so that the subdivisions do not communicate with each other).
- WSNs are heterogeneous, consisting of different types of sensor devices that are interconnected to more powerful back-end systems
- WSN nodes have to operate unattended since it is impossible to service a large number of nodes that may reside in remote and hard to access locations.

The WSN platforms, by using the framework in this chapter, should include three types of services: local services, network transport services, and middleware services. There are challenges in all 3 services. First, local services, even the OS are not clear. Due to the early stage of WSN technology, it is not clear what type of OS is suitable for WSNs. Network transport services are also not clear because the sensor nodes are not IP addressable.

However, most WSNs at present are using the mobile ad hoc network (MANET) model discussed in later chapters (chapter 6, 7 and 10).

The most serious challenges are posed by the WSN middleware that should support the implementation and basic operation of WSN applications. First, how can the WSN middleware hide the underlying uniqueness of the WSNs discussed previously from the applications. Second, how can the middleware services be built without knowing the OS on which the services will be built. Third, how can WSN middleware support automatic configuration and error handling because WSN nodes operate in an unattended mode. Fourth, how can the support for time and location management integrated into a middleware infrastructure for the WSN applications that need time, location, and real-time support. Finally, how can this middleware support not only the sensors but also the devices and networks that are connected to the WSN. In essence, middleware for WSN must be of "information providing" type to inject application knowledge into the infrastructure and the WSN.

Research in WSN middleware and platforms is continuing at present at different universities such as Berkeley, Cornell, and Rutgers. The Cougar Project at Cornell, for example, adopts a database approach where sensor readings are treated like “virtual” relational database tables. An SQL-like query language is used to issue tasks to the WSN. The prevalent thinking favors a hierarchy of services that range from low level sensors to higher level data aggregators, and data storage and analysis capabilities (see Figure 4-30). This approach, presented by [Hill 2004] provides a general framework for developing WSN platforms.

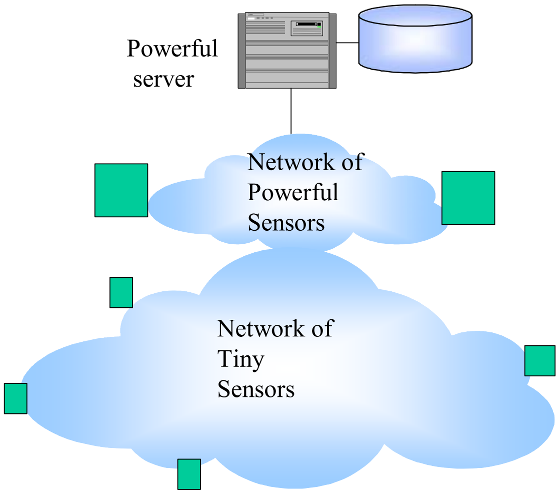


Figure 4-30: Hierarchy of WSN Platform Services

Sources:

- Omer, K., Kasten, O., and Mattern, F., "Middleware Challenges for Wireless Sensor Networks", Mobile Computing and Communications Review, Volume 6, Number 2, October 2002, www.inf.ethz.ch/vs/publ/papers/wsn-middleware.pdf.
- Hill, J., et al, "The Platforms Enabling Wireless Sensor Networks", CACM, June 2004, pp. 39-46.
- Cougar Project. www.cs.cornell.edu/database/cougar
- Sensorwebs Project. basics.eecs.berkeley.edu/sensorwebs.
- Smart Messages Project. www.rutgers.edu/sm

4.8.3 AIRTIS Wireless Traffic and Weather Solution

TransCode's AIRTIS wireless traffic and weather solution is a traveler information system that utilizes wireless technologies. The AIRTIS services include:

- **Traffic Service**: It sends traffic messages to wireless devices providing news about traffic conditions and current construction zones as they occur. It also allows subscribers to customize the service by selecting times and days of the week as well as the traffic severity for which they wish to receive alerts.
- **Weather Service**: It provides continual weather information throughout the day and notifies subscribers of severe weather conditions such as tornado warnings or snow and ice conditions. It also provides weather forecasts and allows subscribers to specify up to 10 counties for weather alerts and forecasts.
- **Email Service**: It provides subscribers wireless email access from any Internet email system and allows subscribers to use their PC email address and mailbox.

The AIRTIS traffic and weather application is built using the Nettech's RFlink™ middleware. It is available on the AIRTIS website, www.airtis.com, where subscribers can customize the service. The application can be accessed through RIM 850 Wireless Handheld™ devices. The application consists of traffic and weather profiles. The Traffic Profiles are used to set up "commute profiles" which are matched against their real-time traffic database to inform members of traffic tie-ups via pager or other wireless device. The Weather Profiles allow members to set up counties, days, and times to receive weather forecasts on their wireless device. In addition, members can access messages sent to their wireless devices online and can update their account information if they change wireless service providers, change addresses, etc.

Sources:

- http://www.mobileinfo.com
- http://www.airtis.com

4.8.4 City of Seattle Public Utilities Chooses Wireless Middleware[4]

Seattle Public Utilities provides more than 1.3 million customers with a reliable water supply, as well as essential sewer, drainage, and solid waste services for the City of Seattle. To deliver these basic services, the company relies on a system of pipes, reservoirs, and disposal and recycling stations.

To provide an efficient service, the company needed to track technicians' whereabouts, assign resources or schedule appointments optimally, adjust technicians' assignments in real time to accommodate unforeseen change of schedule, and coordinate between dispatcher, technicians and customers. In addition, the technicians were not able to update work status, track details of parts used, place orders for parts, or update inventory in real time because they had to fill out paper work and re-key data in the office. Because of the time lag between paper submission and re-keying of data, the invoices could not be generated quickly. The Work and Inventory Management (WIM) application that runs on the Oracle database was adequate for corporate office access – the office workers installed the application software on their desktop machines to access the Oracle database. But this did not provide adequate real-time access from the field technicians.

The company chose portable computers, running Windows OS, as their mobile devices. The portable computers were chosen because they can be used in demanding field conditions that

[4] Suggested by P. Arasalingam.

the technicians encounter. In addition, they can use a multitude of applications that keep appearing in the Windows environment. The company wanted a packet-based service in the wireless network for the frequent communications with the technicians on the field; they chose CDPD (cellular digital packet data) and Metricom Richolet.

Beyond the mobile device and wireless network, various software options on the portable computers needed to be considered. These options are shown in Figure 4-31. The first choice was to install the WIM application software on these portable computers. The main advantage of this approach is that the user interface stays the same whether the technicians run the WIM application in the corporate office or remotely. In addition, the IT department would set up the portable computers the same way they set up any normal desktop computer. However this approach does not address the network connection issues because the cellular wireless networks are not as fast and reliable as a normal LAN connection in the office. Tests showed that the WIM application would take a long time to open a window and perform basic operations. The main reason for this was that the Oracle database was accessed directly by using SQL, and every SQL request returned packets with huge sizes that took a long time to travel over the slow cellular network.

Another option was to use the client-server software called Cytrix Metaframe. In this case, the WIM application is subdivided into two parts: the WIM server application that is installed on the main server, and the client software, a thin client, that is installed on the mobile devices – in this case the portable computers. This approach improved the response time (windows opened quickly and query response was faster) because the application ran on the server connected on a high speed LAN in the office. This eliminated the running of SQL queries over the wireless network. The application just sent the needed output (presentation) to the laptops.

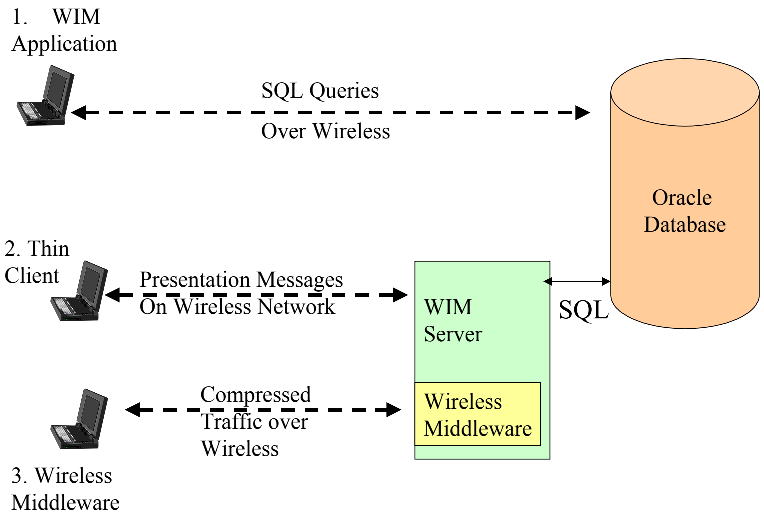


Figure 4-31: Platform Choices for Seattle Public Utilities

The third approach was to use a wireless middleware package. The company chose middleware called Smart IP because it compresses data and uses its own wireless optimized protocols to transmit it over wireless networks. This results in fewer transmissions of packets. Tests showed that this middleware makes download of applications and update of databases much faster. The company decided to use a combination of the above approaches to address their application needs.

By using the combination of an application server that only sends presentation messages over wireless networks and a wireless middleware package that compresses these messages, the application runs very well. This has allowed the technicians to gain all information needed while in the field. While working on their assignments, the technicians can track labor and parts.

Sources:

▪ Knowledge Base – A Case Study	www.rysavy.com
▪ Cable and Wireless Case Studies	www.cw.com/solutions/case_studies/
▪ CNET – White Papers and Case Studies	www.itpapers.news.com

4.8.5 M-Commerce: Nokia and Amazon.com Collaboration

Amazon.com and Nokia have formed a partnership aimed at delivering mobile commerce services to users of Internet-enabled mobile phones. The two companies are rolling out worldwide Internet-based mobile commerce services based on the WAP (wireless application protocol) specification.

Amazon.com launched a mobile section at its UK website (www.amazon.co.uk) designed to let users of WAP-enabled cell phones read reviews and purchase products from the online vendor's catalog of books, compact discs, and other products. The main features of this service are:

- Built for high capacity. Can handle sharp traffic bursts without missing a beat.
- Links directly to partners and suppliers for seamless integration.
- Easily scalable, from small deployments to large.
- Allows sophisticated consumer profiling and pinpoint targeting of offers.
- Compatible with digital cable and digital satellite broadcasting systems.
- Completely safe, secure and private.

4.8.6 Drexel University Wireless Network[5]

Drexel University embarked on a major wireless initiative to become the first major university go wireless. As a result, within its campus limits in the city of Philadelphia, Drexel has installed around 300 antennas and provided the ability for mobile computers with pre-approved access to connect to the university networks. But, prior to February 2003, Drexel had two separate computing infrastructures – its proven wired backbone and its newer university-wide wireless network. Drexel needed to link the two networks together and provide its 16,000 students and 3,000-member workforce a uniform access across wireless and traditional wired networks across the campus. This merger of networks was placed on a high priority as the university created the college of medicine by acquiring the MCP Hahnemann University. The university required its medical students to have a wireless computing device, or laptop, to complete their schoolwork on campus.

It was soon realized that just connecting wireless and wired networks was not enough. A higher-level service at middleware was needed to provide the users a seamless view of the computing services. Drexel chose the Microsoft .NET platform to merge the services provided by the two disconnected networks into a seamless campus-wide mobile computing network. By using the components of the .NET Framework, – especially the Mobile Internet Toolkit and Visual Studio®, the core environment of DrexelOne Mobile was developed by

[5] Suggested by Jeff Hager

February of 2003. The DrexelOne portal allows the students to retrieve personalized information from virtually any Web-enabled mobile or handheld device.

The DrexelOne Mobile used the WAP push model with good results. By using the WAP push model, the university was able to push personal information to any student on campus that needed immediate attention – not just when the student finally logged on to view his/her account. In addition, university specials or team sport information could also be disseminated to a group or specific college within the university. The following figure shows a simple WAP Gateway that handles all mobile devices that may typically be used on campus to retrieve information from the DrexelOne portal. Note that desktops can also access the portal through the wired network.

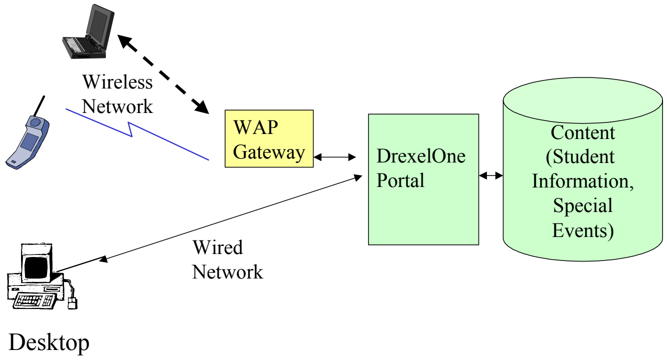


Figure 4-32: DrexelOne Portal

Sources:

http://www.drexel.edu/IRT/news/archives/d1mobile.html

http://www.sct.com/Education/Clients/Case_Studies/CS_Drexel.html

http://www.asp.net/Default.aspx?tabindex=6&tabid=44

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmitta/html/mmitvxslt.asp

4.9 Concluding Comments

Mobile computing platforms support the development as well as operation of mobile applications. These platforms consist of local platform services that run on the mobile device (clients), the wireless middleware that interconnects remote applications and databases, and network transport services that transfer message over the wireless networks. The most vital part of mobile computing platforms is the wireless middleware that is being packaged as wireless gateways and mobile application servers. Examples of wireless middleware are WAP (Wireless Application Protocol) from WAP Forum, i-mode from NTT DoCoMo, Wireless Java from Sun Microsystems, Mobile Internet Toolkit (MMIT) from Microsoft, BREW from QualComm, and VoiceXML from the VoiceXML Forum.

4.10 Suggested Review Questions and Exercises

1) What are the main components of a mobile computing platform? Give examples of each.

2) What are the typical local services needed by mobile computing devices? Give examples.

3) List the factors you will use to analyze and evaluate the middleware products to support mobile computing applications.

4) Suppose you need to support video conferencing on your mobile devices. List *all* mobile computing platform services you need to consider.

5) Identify and evaluate the middleware services needed for m-commerce.

6) What is the difference between wireless middleware, wireless gateways, and mobile application servers? Explain through an example.

7) What do you consider to be the main pros and cons of WAP?

8) Compare and contrast WAP with i-mode, Wireless Java, and BREW. Base your comparison on the latest information.

9) Sun and Microsoft compete in many areas. In light of this, compare and contrast them.

10) Nokia has a Mobile Application Server (MAS) based on WAP. Find other MASs and compare/contrast them with the Nokia WAP Application Server.

11) How is VoiceXML different from WAP? Is VoiceXML necessary? Could it have been handled through WAP, i-mode, J2ME, MMIT, BREW or others?

12) You have been asked to classify and compare and contrast the various wireless middleware services in their various forms (wireless gateways, mobile application servers). What factors will you use? Prepare a table that compares and contrasts the available systems based on your factors.

4.11 References

Arehart et al. *Professional WAP*. Chapter 16. Wrox Books, 2000.

Bates, R. *Wireless Broadband Handbook*. McGraw Hill, 2001.

Barnett, N., et al. “M-commerce: An operator’s manual.” *McKinsey Quarterly*, No. 3 (2000), pp. 163-175.

Bates, R. *Wireless Broadband Handbook*. McGraw Hiil, 2001.

Davies, N. “The Impact of Mobility on Distributed System Platforms.” Invited paper, *ICDP96 Proceedings*, Dresden, Feb. 1996.

Diehl, N., Grill, D., Held, A., Kroh, R., Reigber, T., and Ziegert,T. “System-Integration for Mobile Computing and Service Mobility.” *ICDP96 Proceedings*, Dresden, Feb. 1996.

Fingar, P., et al. *Enterprise E-Commerce.* Meghan Kiffer, January 2000.

Gralla, P. *How Wireless Works*. Que, 2001.

Hill, J., et al, "The Platforms Enabling Wireless Sensor Networks", CACM, June 2004, pp. 39-46.

Kalakotta, R. and Robinson, P. *M-Business.* McGraw Hill, 2002.

Lange, D. and Oshima, M. "Seven Good Reasons for Mobile Agents." *Comm of ACM*, March 1999, pp. 88-91.

Maes, P., et al. "Agents that Buy and Sell." *Comm of ACM*, March 1999, pp. 81-82.

Milojicic, D., et al., eds. *Mobility: Processes, Computers, and Agents*. Addison-Wesley, April 1999.

Omer, K., Kasten, O., and Mattern, F., "Middleware Challenges for Wireless Sensor Networks", Mobile Computing and Communications Review, Volume 6, Number 2, October 2002, www.inf.ethz.ch/vs/publ/papers/wsn-middleware.pdf.

Preguica, N., et al. "Mobile Transaction Management in Mobisnap." *Proceedings of ADBIS-DASFAA 2000.*

Richter, K., Rudolf, S., and Irmscher, K. "Mediator Services for Mobile Clients." *ICDP96 Proceedings*, Dresden, Feb. 1996.

Sampei., S. *Applications of Digital Wireless Technologies to Global Wireless Communications*. Prentice Hall, 1998.

Schilit, B., and Theimer, M. "Disseminating Active Map Information to Mobile Hosts." *IEEE Network*, Sept/Oct. 1994, pp. 22-31.

Sreetharan, M. and Kumar, R. *Cellular Digital Packet Data*. Artech House, 1996.

Stallings, W. *Wireless Communications and Networks*. Prentice Hall, 2001.

Umar, A. e-Business and Distributed Systems Handbook: Middleware Module. 2nd ed. NGE Solutions, 2004.

Van Der Heijden, M. and Taylor, M. *Understanding WAP*. Artech Publications, 2000.

Varshney, U. and Vetter, R. "Emerging Mobile and Wireless Networks." *Comm of ACM*, June 2000, pp. 72-81.

Veijalainen1, J. "Transaction Management for M-Commerce at a Mobile Terminal." Available at: http://cosco.hiit.fi/Articles/hiccs36.pdf

Vichr, R. and Malhotra, V. "Middleware Smooths the Bumpy Road to Wireless Integration." September 2001. Available at IBM Developer Site: http://www.106.ibm.com/developerworks/wireless/library/wi-midarch/

Walker, J., ed. *Advances in Mobile Information Systems*. Artech House Mobile Communications Library, 1998.

Walters, B. *Computer-Mediated Communications: Multimedia Applications*. Artech House, 1995.

Wittman, A. "Brush Up on Bluetooth." *Network Computing*, April 1999.

Wong, D., et al. "Java-based Mobile Agents." *Comm of ACM*, March 1999, pp. 92-96.

Woolf, B. P., and Hall, W. "Multimdeia Pedagogues: Interactiave Systems for Teaching and Learning." *IEEE Computer*, May 1995, pp. 74-80.