Designing Wireless Enterprise Applications Using Java™ Technology

A Java BluePrints for Wireless White Paper

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

January 2002 (Revision 2)

Designing Wireless Enterprise Applications Using Java™ Technology

Today's businesses have an established standard for developing multitier enterprise applications: the Java™ 2 Platform, Enterprise Edition (J2EE™). One of the J2EE platform's advantages is its ability to accommodate many types of clients, including Web browsers, Java applets, and Java applications, which can easily be deployed onto a laptop, desktop or workstation.

The Java 2 Platform, Micro Edition (J2ME™) gives businesses the opportunity to accommodate a new set of enterprise clients: cell phones, two-way pagers, and palmtops. These Internet-ready devices can be programmed using the Mobile Information Device Profile (MIDP), a set of Java APIs which, together with the Connected Limited Device Configuration (CLDC), provides a complete Java runtime environment.

With the J2ME and J2EE platforms, you can create 100% Java technology-based wireless enterprise solutions. This paper describes how to use these technologies together, and the issues you will face when designing and implementing your own J2ME-J2EE solutions. From time to time, this paper will cite examples from the Java Smart Ticket Demo (available through the J2EE BluePrints for J2ME Wireless Web site at `http://java.sun.com/j2ee/blueprints/j2mewireless/`), a sample movie ticket reservation application. Once you understand the principles behind the Java Smart Ticket Demo, you can easily apply the same principles as you develop other mobile services.

This paper refers to many technologies across the J2ME and J2EE platforms. If you need to get up to speed on any of these technologies, don't worry; the Resources section at the end of this paper contains references you should find useful.

0.1 Overview of Java Technologies for Wireless Enterprises

Widely supported across the industry, the J2EE standard gives developers a rich technology stack on which to run their enterprise applications. This stack includes the Java Database Connectivity (JDBC™) API for access to tabular data sources, the J2EE Connector architecture for integration into legacy databases and information systems, and the Enterprise JavaBeans™ (EJB™) component architecture for modularizing business logic. On top of this base, JavaServer Pages™ (JSP™) and Java servlet technologies provide a powerful, flexible Web layer for accessing enterprise services.

Today, JSP pages and Java servlets are commonly used to support Web browser clients. Under this arrangement, a browser retrieves dynamically-generated HTML from the server and interprets it to render the user interface. You could use this approach to support wireless clients as well, creating JSP pages and Java servlets that generate Wireless Markup Language (WML) and Compact HTML (CHTML) for consumption by microbrowsers. Unfortunately, browser-based solutions have some key limitations, which are pronounced in the wireless space.

The J2ME MID Profile addresses many of the limitations of microbrowser-based solutions. Developed through the Java Community Process (JCP) program under the guidance of an expert group composed of 20 companies representing the wireless industry, MIDP addresses programming issues most relevant to mobile clients, such as user interface, networking, persistent storage, and application model. Compared to microbrowser-based solutions, MIDP provides the following advantages:

- *Lower network usage and server load*. In a microbrowser-based solution, the server is responsible for generating display markup. This requires a round-trip every time the interface changes. In contrast, a MIDP client's interface is contained within the device, so it can operate even when disconnected. On the occasions the device does interact with a server, it incurs less network traffic, because it downloads only application data, as opposed to application data plus interface markup.
- *A better user experience.* Markup languages such as WML and CHTML are, by design, restricted in the types of interactions they can offer. With the MIDP GUI APIs, it is easy to implement customized widgets and event-handling, opening up unlimited possibilities for mobile client interfaces that are easier and more interesting to use.

0.2 Reaping the Benefits of a 100% Java Technology-Based Solution

With the J2ME and J2EE platforms, you have everything you need to enable your enterprise for mobile access. Furthermore, by using Java technology to implement your entire end-to-end solution, you fully leverage its portability and scalability, as well as the ease of using the Java programming language.

On the server side, you can develop your J2EE applications against the J2EE Reference Implementation and deploy them on any J2EE-branded application server. The J2EE standards ensure that your business gets a choice of servers appropriate to the strategic purpose of your application.

On the client side, you can develop your MIDP application using the J2ME Wireless Toolkit and provision it onto any MIDP-compliant device, whether it's a cell phone, two-way pager, or palmtop. This allows you to serve a wider range of clients, making your enterprise even more accessible.

The bottom line benefits of using a 100% Java technology-based solution are increased programmer productivity, better strategic use of computing resources, and a greater return on an organization's technology investments.

0.3 Architecture of a Java Wireless Enterprise Application

The high-level architecture of a Java wireless enterprise application is similar to that of a canonical J2EE application.

In the Java Smart Ticket Demo, for example, the client tier consists of a MIDP application, or MIDlet, which provides the user interface on the mobile device. The MIDlet, in turn, communicates with Java servlets in the Web tier to access the business logic of Enterprise JavaBeans components in the EJB tier; both the servlets and enterprise beans are located on a J2EE application server. Finally, the enterprise information systems (EIS) tier supports access to the application database through the JDBC API.

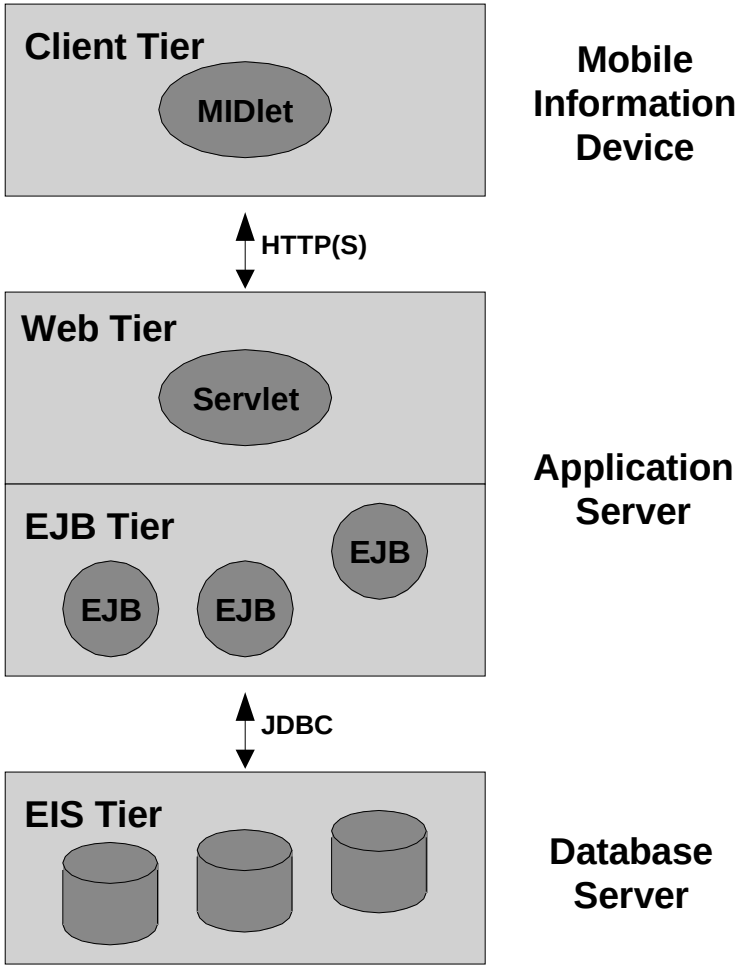

FIGURE 0-1 J2EE-J2ME Application Architecture

In addition to being divided into tiers, the architecture of the Java Smart Ticket demo follows a Model-View-Controller (MVC) organization. In a J2EE-J2ME application, the separation of model (data abstraction) from view (data presentation) is a natural fit since these parts map almost directly to server and client. The controller logic is distributed between the server and client.

One of the benefits of using MVC is that it separates core data access functionality from the presentation and control logic that uses this functionality. Such separation allows multiple views to share the same enterprise data model, which makes supporting multiple views--and by extension, clients--easier to implement, test, and maintain.

Enterprise JavaBeans components support this approach, because they are designed to be reusable. For example, to incorporate an HTML front-end, you only have to add a set of HTML views, which you can connect to your enterprise beans through a separate controller. You can also work in the opposite direction; your J2EE application that already serves Java applets and HTML browsers can be easily extended to serve mobile users.

In the sections that follow, you will find design guidelines on issues specific to J2ME-J2EE applications. You can find an explanation of general J2EE application architecture as well as guidelines on how to design the lower tiers of a J2EE application at the J2EE BluePrints Web site (`http://java.sun.com/j2ee/blueprints/`).

0.4 Messaging

HTTP provides the bridge between the MID Profile and the J2EE platform; because they both support HTTP, they can use this protocol to communicate with each other.

One of the J2EE platform's strengths is its ability to serve many types of clients over the Web, including browsers, plug-ins, Java applets, and even non-Java applications. The JSP and Java servlet technologies help in this regard, providing an extensive framework for communication over HTTP using a dynamic request/response-based paradigm.

MIDP includes standard support for HTTP 1.1, and APIs for generating GET, POST and HEAD requests, basic header manipulation, and stream-based consumption and generation of messages. Consequently, you can use the JSP and Java servlet technologies to serve MIDP clients.

Note that while you program against an HTTP networking API, what goes on behind the scenes may or may not occur over TCP/IP. Depending on the carrier network, messages moving between a MIDP device and a J2EE server may tunnel through a number of different protocols. On the server side, the deployment configuration of the carrier network ensures that such network connections are routed to and received by the J2EE server in the form of HTTP messages.

There are no requirements about the format of messages that flow between a MIDP client and a J2EE application. Recall that both JSP pages and Java servlets let you send any type of data in the body of an HTTP response, not only HTML, or markup

for that matter. Similarly, a MIDlet can send any type of data in the body of an HTTP request. So, the format could be something simple like a string of comma-separated values, or something more structured in the form of key-value pairs, or more formally, XML data conforming to a set of DTDs or schemas.

0.4.1 Design and Implementation Guidelines

This section describes the issues you should consider when designing and implementing messaging in a J2ME-J2EE application.

0.4.1.1 Choosing a Message Format

When choosing a message format, keep in mind that the size requirements the format imposes will have pronounced effects in a wireless production environment. Well-formed and self-descriptive XML messages can be orders of magnitude larger than simple text-based messages, consuming bandwidth and airtime for which the user of the application may be billed.

To be sure, wireless applications using XML do enjoy some advantages. For one, the messages are self-describing, making debugging and development easier. Another, perhaps more significant win is that many Web services are XML-based; if a J2EE application publishes such services then developing a MIDP client to use the very same services makes its integration almost seamless. The J2EE and J2ME platforms accommodate this strategy.

The J2EE platform provides a rich set of libraries for building XML-based Web services, through the Java APIs for XML Messaging (JAXM), XML-based RPC (JAX-RPC), and XML Registries (JAXR). The lower-level Java API for XML Processing (JAXP) allows developers to manipulate XML documents using the Document Object Model (DOM) and Simple API for XML (SAX), and to transform XML documents using Extensible Stylesheet Language Transformations (XSLT).

On the MIDP side of the equation, a few well-supported XML parsers exist. (At the moment, these are not native components of the profile.) These parsers do come at a cost, however, increasing a MIDlet's size by 15 to 30 kilobytes. Also note that since the parsers are designed for constrained environments, they may offer only SAX for processing XML documents, and may not include XML validation or sophisticated error-reporting mechanisms.

0.4.1.2 Ensuring Confidentiality

When designing an application such as the Java Smart Ticket Demo, which involves the transmission of sensitive user information such as passwords and credit card numbers, it is important to keep private the communication between client and server.

If your MIDP device supports the Secure Sockets Layer (SSL) protocol, you can protect your messages using HTTP connections over SSL (HTTPS connections). Furthermore, because of MIDP's Generic Connection Framework, making an HTTPS connection does not require any special or different code. A MIDP client just needs to open a connection to a server on an URL beginning with `https:`.

For your convenience, the MIDP device emulation in the J2ME Wireless Toolkit supports HTTPS. This capability lets you test, in your development environment, your applications that use HTTPS. (For more information, see the *J2ME Wireless Toolkit User's Guide*.)

0.4.1.3 Utilize Content Type and MIME Type

To aid debugging of messages sent from client to server, you should describe the type of data in the message using the `content-type` header in the HTTP request. In some deployments, this is absolutely necessary; some gateways through which the HTTP messages tunnel do not accept messages with undefined or non-standard `content-type` headers.

0.4.1.4 Aggregate Messages Before Sending

In wireless production deployments, a significant cost of an HTTP connection lies in establishing the connection in the first place. You can avoid this cost by aggregating messages before sending them, minimizing the number of messages the client sends.

0.4.1.5 Use Progress Indicators During Messaging

It is sometimes impossible to avoid sending large chunks of data either to or from mobile devices. Since these devices often encounter unexpected network delays and unreliable network performance, a visual indication of the progress of a data exchange can improve user experience.

0.5 Session and Personalization Management

A *session* is a sequence of service requests by a single user using a single client to access a server. *Session state* is the information maintained in the session across requests, such as the contents of a shopping cart which are updated as a user browses and chooses from a catalog.

Whereas session state spans service requests, *personalization data* is maintained between sessions. Typically, many aspects of a user, such as his or her address, zip code, or favorite color, would not change from session to session. Because such data is stable, an application can use it to personalize a user's experience.

This section describes how to maintain session state and personalization data in a J2ME-J2EE application.

0.5.1 Managing Session State

HTTP is, by design, a stateless protocol. It does not outline any mechanisms for determining if a series of requests is from one user or another user. The Java servlet API works around this limitation by providing an `HTTPSession` object, which represents each user's interaction with the Web layer. J2EE application servers use three methods to maintain the integrity of `HTTPSessions`:

- *Using cookies.* A cookie is a small chunk of data a server sends for storage on the client. Each time the client sends information to a server, it includes in its request the headers for all the cookies it has received (and stored).
- *Using URL rewriting.* This technique involves encoding every URL on a served page to include client-side session state. When the client selects an URL, the encoded session state is sent back to the server with the request.
- *Using HTTPS.* To ensure the privacy of an HTTPS communication, the client generates a session key that uniquely identifies the user.

MIDP devices can use all of these three methods. (However, HTTPS is not a requirement of MIDP.) The first two methods are relatively simple to implement. When using cookies, clients must always retransmit the last cookie sent by the server, which usually contains only a session ID, and for URL rewriting the client must append the session ID to each request.

The Java Smart Ticket Demo uses URL rewriting but it could also use cookies. It can be useful to implement URL rewriting as a fallback to cookie support since some gateways do not allow the transmission of cookies.

0.5.2 Managing Personalization Data

While session management can be characterized as transient, personalization data is by nature persistent. In a J2ME-J2EE application, the data can be persisted on the server, or the client, or both. The MIDP Record Management Store (RMS) API allows you to store data that persists on the device between uses of an application. On the other hand, J2EE application servers offer numerous ways of integrating with other information systems to store and retrieve user information and preferences.

A major benefit of personalizing an application is that it can lead to fewer user interactions for completing a task. This is especially important for wireless devices, whose inputs are constrained. In the case of the Java Smart Ticket Demo, it would become tedious to re-enter a zip code with every use of the service since the user is often interested in seeing movies close to home.

0.5.3 Design and Implementation Guidelines

The main consideration for session and personalization management is how much of these responsibilities should be distributed among client and server.

0.5.3.1 Let the Server Take Care of It

To reduce the size of messages sent from the client to the server, store session state and personalization on the server. For example, consider an application that uses a preference such as zip code or membership level to tailor the results of database queries. If this preference is managed by the server, the client portion of the application does not need to transmit the same preference with each request it sends.

Also, by letting the server manage sessions and personalization, it is easier to support multiple types of clients. For example, users who access an enterprise service from a cell-phone may want to access the same service through a Web front-end when they are at their laptop or workstation. When they do so, they would expect not to have to reenter information such as their address, and credit-card number; these items should already be on file so that all they have to do is sign in.

0.5.3.2 Distinguish Between Generic and Device-Specific Preferences

Not all data should be stored on the server. For example, a mobile user may prefer to display fonts in a certain size or style. These preferences are specific to the user's interactions with the MIDP client, not with the service; hence, when the user accesses the application through another client, these preferences would be inappropriate. Device-specific information of this sort is most conveniently stored directly on the MIDP device.

You can also use client-specific preferences to streamline workflows. For example, the Java Smart Ticket Demo remembers whether a user prefers to see a preview, a poster, or neither after choosing the movie to see. This preference may be influenced by the speed of the network or the performance of the device, and thus they should be stored on the device itself.

0.6 Deployment and Provisioning

When deploying a wireless enterprise application, you follow the usual steps for deploying J2EE and MIDP applications.

On the J2EE side, JSP pages and Java servlets are packaged inside a Web archive (WAR) file, while EJB components are packaged inside a Java archive (JAR) file. The WARs and JARs go inside a single enterprise archive (EAR) file, together with descriptor files that describe the deployment requirements of the components. J2EE application servers typically include graphical tools for assembling and deploying EAR files.

Deploying a MIDP application is slightly simpler. A MIDP application is packaged inside a JAR file, which contains the application's class and resource files. You can deploy this JAR file using a couple of methods:

- Pre-installing the JAR on the mobile device.
- Downloading the JAR from the J2EE server once onto the mobile device.

When deploying the application, keep in mind that wireless devices generally have limited memory and work on networks with limited bandwidth. Smaller applications are desirable.

Accompanying the JAR file is a Java Application Descriptor (JAD) file, which describes the application and any configurable application properties. You might find it useful to make the URL of your wireless service a configurable property, for example.

0.7 Conclusion

This paper has described how you can design, develop, and deploy complete wireless enterprise solutions, such as the Java Smart Ticket Demo, using the power of the J2ME and J2EE platforms. Such solutions can fully reap the benefits of Java technology: portability, scalability, and programming ease.

The high-level architecture of these solutions is simple, yet effective. Enterprise JavaBeans components provide business logic on top of enterprise data. Java servlets in the Web tier use these components to access and manipulate data on behalf of MIDP clients, which in turn present the data to the user. This division of responsibilities makes the application easier to implement, test, and maintain.

Most importantly, because EJB components are designed to be reusable, an enterprise application enabled for mobile clients can just as easily serve traditional desktop clients, such as Java applets and Web browsers. The net result is a truly accessible enterprise: anytime, anywhere, on anything.

0.8 Resources

For more information on designing wireless enteprise applications using Java technology, visit the J2EE BluePrints for J2ME Wireless Web site at `http://java.sun.com/j2ee/blueprints/j2mewireless/`.

You may also find the following references useful:

- Roger Riggs, Antero Taivalsaari, Mark VandenBrink. *Programming Wireless Devices with the Java 2 Platform, Micro Edition*. Addison-Wesley, Boston, MA, 2001.
- Nicholas Kassem, Enterprise Team. *Designing Enterprise Applications with the Java 2 Platform, Enterprise Edition*. Addison-Wesley, Boston, MA, 2000.

All the facilities described in this paper are available in J2ME MIDP 1.0.3, and the J2EE 1.3 platform. For more information on these technologies, visit the Java Software Web site at the following URLs:

- `http://java.sun.com/j2ee/`, and
- `http://java.sun.com/products/midp/`

At those locations, you will find reference implementations for these technologies, which are free for download.

Also consider joining the interest lists hosted at Sun for these technologies:

- `KVM-INTEREST@JAVA.SUN.COM`, and
- `WEBJAVA-WIRELESS@JAVA.SUN.COM`.

Details on how to subscribe to these lists are available at `http://archives.java.sun.com`.

Readers may send comments on this paper to:

`j2ee-j2me-blueprint@sun.com`