Beyond the Wire

Developing WebLogic® Software for Many Devices

A BEA White Paper

Copyright

Copyright © 2001 BEA Systems, Inc. All Rights Reserved.

August 16, 2001

Restricted Rights Legend

This document may not, in whole or in part, be photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc. Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems, Inc.

Trademarks

BEA and WebLogic are registered trademarks; BEA WebLogic Server, BEA WebLogic Integration, BEA WebLogic Personalization Server, Built on BEA, and BEA WebLogic E-Business Platform are trademarks of BEA Systems, Inc.

All other company and product names are trademarks of their respective owners.

CWP0297E0801-1A

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131 U.S.A.
Telephone: +1.408.570.8000
Facsimile: +1.408.570.8901
www.bea.com

Contents

Overview

As phone carriers and other independent providers increasingly add wireless Web services to their networks, virtually any Internet-enabled device within reach of a cellular base station can initiate and maintain a connection to the Internet – without the limitations of a physical wire. In essence, the tether to the Internet has been cut.

Wireless network coverage has grown quickly, largely due to the trend toward highly mobilized workforces. Although the most common consumer devices enabled for wireless Internet access are cellphones and personal digital assistants (PDAs), many enterprises engage special purpose mobile devices as well, to perform single tasks for their mobilized work force.

Because these devices vary greatly in form, user input interfaces, and use of non-standard markup languages and over-the-air (OTA) protocols, Web applications are being challenged to interact with many more client environments than just the desktop browser.

The good news is that any existing application built on BEA WebLogic® Server is ready for the wireless Web. In fact, an application's entire back-end is already equipped for use by Internet-enabled devices. Without making any significant code modifications, an application developer needs only to write a new multi-device client and add a transformation service to a BEA WebLogic Server™ cluster to make an application available for any Internet-enabled device.

Thanks to the "browser" interface, the Internet caught fire in the early 1990s. A far cry away from the DOS environment, without the simplicity of the browser interface the Internet may never have become as popular as it is today. The browser makes it possible to quickly develop applications both on the Internet and within corporate intranets.

Prior to the introduction of the browser, an application developer, using a two- or three-tiered application architecture, developed the business logic and data access layers only once, but had to develop a separate client graphical user interface (GUI) for each platform on which the application needed to run.

As the Internet began to mature and users settled on specific client platforms, many Internet applications became single-purposed or specialized (such as real-time stock quotes or real-time audio/video applications). These applications, often written as standalone applications and not for the browser environment, allowed for better control over their visual presentations and data communications, but the applications were once again being developed separately for each client platform. However, with the introduction of Java and its graphical class libraries, developers began using a standardized development environment in which the GUI is developed once and deployed on numerous client platforms.

Until now, an application development model using Java has been sufficient to deploy applications from a single source code base. However, the increasingly popular data-enabled mobile devices are posing a challenge similar to that of the special-purposed or specialized application. These devices have various operating environments, and there is no standard, such as HTML or Java, to tie them together. Furthermore, the browsers are so significantly diverse that even browser-based applications must diverge significantly for each type of browser. Even if the browsers were standardized, the devices contend other differences, such as screen size, that must be considered when developing applications. All of these factors make the task of using a single source code base more difficult when trying to accommodate data-enabled devices.

Application Challenges

Screen Size

The fact that current desktop browser-based applications are written to provide users with a full and rich experience on a large screen makes duplicating that experience on a small screen nearly impossible. In fact, much of the content presented on a desktop browser-based application will not translate well to a small screen. For example, consider a Web site using a large image map for navigation to different parts of the application. Most likely, the image map would be too large to fit on a small screen, and reducing the image to fit would result in text too small to read.

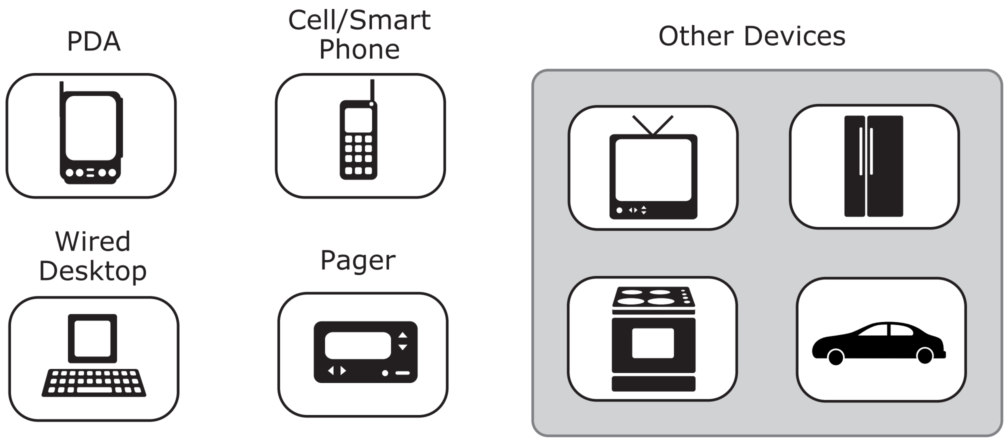

Figure 1: The Proliferation of Data-Enabled Devices.

Data-enabled devices are available in many different shapes and sizes, ranging from a wristwatch to a device about the size of a deck of cards, and are fixed in size for the specific device. Even if a reasonable representation of the image could be produced for a small screen, most data-enabled devices offer no means by which a user can point to areas of the screen or select within specific locations of an image.

Since many mobile devices can only display four or five lines of text with ten to twenty characters per line, the challenge is to present information on small screens in such a way that is simultaneously useful and pleasing. Even though these devices offer page-scrolling capabilities, the content needs to be scaled to make the experience more captivating for users.

Visual Capabilities

A desktop computer typically utilizes a large, color, graphic-enabled screen. Many mobile devices still use black and white screens; and some have no means to display graphical images. If a Web application relies heavily on color or graphics, it will not translate well on smaller data-enabled devices.

Because of this, application developers need to consider creating different representations of an application so that the application presents an optimal interface to devices not having the same capabilities as a desktop computer.

Data Entry Mechanism

Entering information on a cellphone can prove to be a slow and tedious process; users must press a key several times to cycle through the letters or characters associated with the key. Using the numeric keypad to enter data discourages many users from entering data, especially those who fumble around trying to manipulate the small keys.

A PDA (such as a Palm Pilot), on the other hand, generally uses a stylus for data entry. Touted as being as easy as to use as writing with a pen, on many devices, however, users are required to use a special character set (such as Graffiti on the Palm OS), rather than the character set of their native language. Moreover, even after mastering the new character set, users are often confronted with errors in the translation of the entered keystrokes. In this regard, entering data with a stylus proves only slightly better than entering data from a numeric keypad.

The lesson to be learned is that application developers need to consider the data entry mechanism as well as the amount of data to be entered when designing and writing applications. If possible, developers should separate an application's required data entry points from its optional data entry points and only present the required data fields to mobile devices. Generally, services are not usable if a user cannot get to the needed information in three mouse-clicks.

Navigation Keys

Referred to as “power keys,” many cellphones use unlabeled keys that an application may use to assign operations. Commonly assigned operations are <PREV>, <BACK>, <OK>, <SUBMIT>, <SELECT>, <DONE>, <NEXT>, <ABORT>, and <HOME>. Some phones have only one power key, while others have four or more.

“Soft keys” on many cellphones allow an application to reassign the operation of fixed keys. An application often reassigns the soft keys so that the keys can be used to select the first nine items in a list of links. Being able to assign power keys and soft keys to replace buttons and links present in the browser-based desktop version of an application is a very useful navigation and data entry tool for the user.

Likewise, PDA devices allow applications to reassign the use of buttons imprinted on the silk as well as the physical pushbuttons. Even though the stylus is an efficient mechanism to select buttons and links, an application becomes even better by using this reassignment capability.

Desktop browser-based applications typically contain buttons and links coded onto each page. To take advantage of the buttons on data-enabled devices, application developers need to separate the application’s navigation links from each page and apply them on a per-device basis to achieve the optimal use of each device’s capability.

Page Data Size Limitations

An advantage to a desktop browser-based application is that it is able to present virtually as much information on a page as it desires, almost without limitation. While it may not be a practical use of space and a user with a slow Internet connection may not care for this capability, the capability is nonetheless available should an application choose to use it.

In the wireless world, the size of a page received by a device depends on two factors:

- The amount of data that the wireless gateway sends in a message payload.

  The telecommunications carrier from which the device receives its data service can limit the amount of data that can be sent for a page.

- The amount of data that the device, itself, allows in a message payload.

  One popular class of devices currently limits the amount of data to 1.4K of compressed binary data per WML deck.

When servicing mobile devices, an application needs to either reduce the amount of information presented to the wireless subscribers or “chunk up” the data and link to previous and next chunks. An application must divide large forms and still maintain context of a single form from page to page. Similarly, an application must either reduce the size of selection lists or chunk up selection lists to retain the context.

Microbrowsers

Additional issues for data-enabled devices that are not generally issues for desktop applications pertain to the microbrowsers running on the data-enabled devices. Whereas desktop users can easily update their desktops with the latest browser technology, wireless subscribers cannot update their mobile devices with the latest microbrowser technology as easily.

Application developers need to consider microbrowser issues when designing and writing their Web applications. Whereas all desktop browsers accept HTML as the standard markup language, most microbrowsers do not support HTML as their markup language. The reasons are many, but the most compelling reason is that HTML does not support the special features needed by data-enabled devices, such as the capability to assign application operations to power and soft keys.

Microbrowsers use one of several markup languages designed specifically for mobile devices. Three general use markup languages used in data-enabled devices today include HDML, WML, and cHTML. Newer devices are no longer being produced exclusively with HDML because it has been replaced by WML in the OpenWave microbrowser. An effort is currently under way to consolidate WML and cHTML into XHTML (Basic), but this endeavor is still several years from conclusion.

Additionally, as much as the companies producing desktop browsers contend with minor differences in their interpretation of the HTML standard, the companies producing microbrowsers differ significantly on their interpretations of the standards on which the microbrowsers are based. For example, one company's microbrowser supports a WML card title by anchoring it to the top of the screen and the rest of the card scrolls beneath the title, while another company's microbrowser completely ignores the card title.

Cell phones have microbrowsers burned into flash memory. A microbrowser update is often possible with a flash upgrade, but the telecommunications carriers have no incentive to provide this service. The carriers' business model is to continuously sell new cellphones with the updated data capabilities already loaded. Since subscribers cannot easily upgrade their microbrowsers, Web applications must continue to support older versions of the most popular microbrowsers for a longer length of time than they would support older versions of a desktop browser. In some cases, the difference from one microbrowser version to the next is so significant that the versions appear to be unrelated products.

Hardware Limitations

Most data-enabled mobile devices have limited hardware resources, a very small amount of memory relative to desktop computers, and no physical persistence (storage) capability. Due to the limited storage capability, a device generally does not store cookies, and only caches a few (if any) of the most recently visited Web pages.

The lack of cookies on the device means that features such as session persistence, automatic login, forms pre-filling, and electronic wallet information that are taken for granted on a desktop are not usually directly available from a data-enabled mobile device. In addition, many more “round trips” may be required between the device and the application to acquire Web content.

Application developers need to be aware of these hardware limitations when developing Web applications. Some of the functionality not possible on the devices can be accomplished through server-side assistance.

Security

Data-enabled devices not having persistent storage usually cannot store PKI certificates, a barrier to complete security/privacy between the device and the application. Sites requiring high levels of security need to develop alternative security mechanisms for these limited devices.

Connectivity

Application developers need to consider certain connectivity issues unique to data-enabled wireless devices. First, wireless connectivity is not ubiquitous. On many continents, including North America, general coverage is spotty at best. Even in regions hosting full coverage, dropout areas do exist; tunnels and valleys are responsible for most dropout areas. Larger urban areas usually have good wireless coverage, while rural areas do not. Because the next generation of networks is initially being installed in the higher density areas, achieving ubiquity may take awhile.

Second, the speed of wireless networks is significantly slower than that of wired networks. While third generation (3G) and subsequent networks will improve speed, the bandwidths are shared, so in high density areas, the marginal increase in speed may be much lower than expected. Even as wireless and wired network speeds increase over the years, a gap in the speeds between the two networks will always exist. Application developers need to write applications that compensate for the slower speeds of the wireless network.

	Markup Language	Screen	User
Wired Desktop	HTML	Window, Size Varies, Color	Keyboard, Mouse
PDA	Web Clipping, HTML, WML	Medium, Square, Color or B&W	Stylus, Buttons, Keyboard
Cell Phone	WML, cHTML, HDML, XHTML	Small, Square, B&W [Some are Text Only]	Numeric Keypad, Power Keys, Soft Keys
Smart Phone	WML, HTML, XHTML	Medium, Rectangle, B&W	Stylus, Numeric Keypad, Keyboard
Pager	HTML (RIM)	Small, Rectangle, B&W [Most are Text Only]	Buttons or Mini Keyboard
Voice Client	VoiceXML	N/A	Human Voice, Numeric Keypad
Television	HTML	Frame on Screen, Size Varies, Color	Remote Control, Keyboard, Mouse

Table 1: Some Application Challenges.

Multiple Data Channels

Many mobile devices have one or more channels that are not Internet-based. The most prevalent is an asynchronous message channel utilized to communicate with wireless subscribers using short data messages.

Messaging Channel

Common terms for the asynchronous messaging carried on the messaging channel are “SMS messaging,” “Instant Messaging (IM),” “Paging,” and “Email.” There are many protocols for asynchronous messaging, including SMPP, CIMD2, UCP/EMI, WAP Push, AOL-IM, SNPP, and SMTP. Some of these protocols allow acknowledgements and tracing of the status of a message, while others only allow for the delivery of datagrams.

Messages can range from simple alerts to requiring an acknowledgement from the recipient. In the case of a message requiring an acknowledgement, if the recipient does not acknowledge the message, the application can escalate the alert to another user or try to deliver it through another channel or device. Alerts can also be actionable, in which the receiver of the message selects from a number of choices for a reply, and the sender of the message receives the selection once it has been made. Applications can be written to take some action based on the recipient's selection (or lack of selection).

When writing an application, an application developer needs to consider whether using the messaging channel would enhance the application for those devices having the messaging channel capability. It is poor etiquette, and illegal in some countries, to deliver unwanted messages to a user's device without first giving them the opportunity to decide which messages they want to receive. A wireless subscriber should be given the choice to receive or refuse asynchronous messages, and elect to explicitly give the device address to a party as a prerequisite to receiving messages. However, messaging is increasingly seen as another advertising medium and users may have to pay a premium to filter out unsolicited advertisements.

Voice Channel

All telephones, not just cellphones, are already enabled for voice interaction with an application. Voice-over-IP (VoIP) is not yet readily available over most networks making direct integration of voice with the microbrowser data channel not yet feasible. However, a user dialing into a voice-enabled server containing voice recognition and text-to-speech (TTS) capabilities should be able to access an application enabled for interactive voice-response (IVR) communication.

The challenge of the voice channel is to create a responsive and easy-to-navigate application. An application with a deep hierarchy of static selection lists hosting more than a few choices is doomed to failure. An application that recognizes unique keywords and provides customized content based on a voice inquiry will make the experience compelling enough to retain the user's interest. The "three click" rule applies to voice portals as well. Users should be able to get to the information they seek within three levels of hierarchy.

Location

Wireless carriers are making information about the location of mobile devices available for application use. Although privacy issues about making such information available are being addressed in some parts of the world, these issues should be resolved within a few years, at which time location information will be available globally.

When developing new applications, developers should consider how the location of a device could be factored into their application designs.

A New Development Paradigm

No matter what device is being used to create applications that provide an optimal experience for wireless subscribers, developers must divide the presentation layer into three areas: navigation control, presentation units, and presentation layout. HTML is not well suited for this new paradigm because HTML layout attributes are mixed with the tags that delineate the blocks of presentation content. In addition, HTML provides no way to include logic for associating links with power keys.

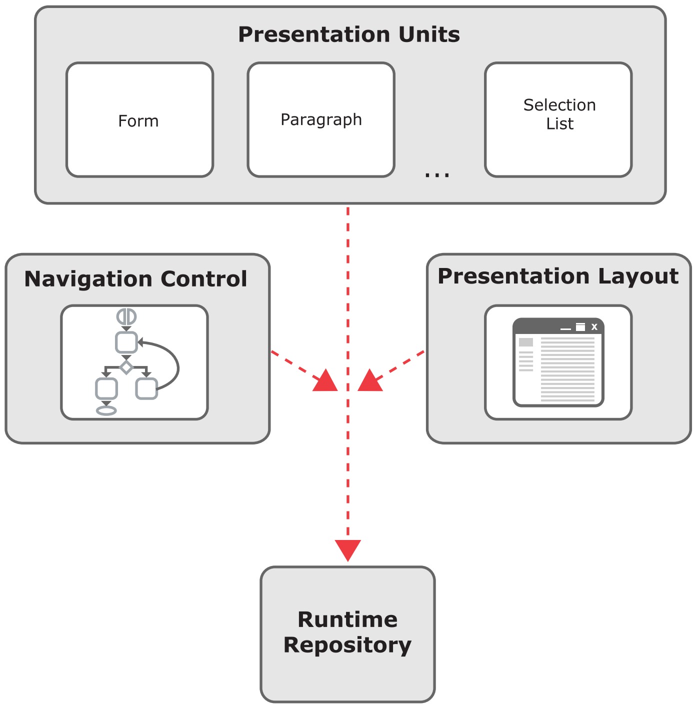

Figure 2: New Development Paradigm.

Navigation Control

Navigational links are used to move from one page to another within the same application, and are usually coded directly into a Web page as selectable URL links. Using the new paradigm, an application developer culls out these links at design time (that is, links should not be coded directly into an application), and uses a navigation flow definition mechanism to define these links. The transformation engine will put the links into the final application at run time.

The reason for separating navigation control from the rest of the application is to allow for device-specific assignment of navigation links at run time; assign navigation links to a cellphone's power and soft keys, to the buttons on a PDA, or to links on a desktop browser page. The new archetype also allows for various navigation flows for different classes of devices.

An application developer should create different high-level navigation models for each class of device. For example, consider the following navigation model for presenting an e-commerce site on a desktop browser:

The e-commerce site home page presents many links to different sections of the e-store, a search form, and several advertisements for current specials. After selecting a part of the store, a product category, and a specific product, the consumer is presented with a full description of the product and an option to add the product to the shopping cart. After selecting the "add" option, the consumer is presented with options to continue shopping or to proceed to checkout.

This navigation model encourages consumers to browse through more of the e-store and potentially make more purchases than originally intended. With a desktop browser and a mouse, this model works very well. However, since mobile devices are simply not well-suited for browsing, using this navigation model to present the e-commerce site on a mobile device may cause a consumer to leave the e-store in frustration. As an alternative, consider the following navigation model for presenting the same e-commerce site on a cellphone:

The home screen allows the consumer to enter a UPC bar code or some other unique identification for the product. The follow-up screen contains a link to add the item to the cart, proceed to checkout, or get a full description of the product. A link to the "general e-store" appears on each page so consumers who want to browse can do so, which allows a consumer to easily do comparison-shopping while simultaneously walking through a brick-and-mortar store.

Note that this navigation model is the inversion of the desktop-browser navigation model. The key to creating a navigation model for a mobile device is to limit the amount of user interaction with the device's input and selection mechanisms.

Presentation Units

A presentation unit is any sub-component of an application page that can be used as an autonomous unit. Types of presentation units include forms, selection lists, paragraphs (one or many), and radio button sets. Presentation units generally do not depend on other presentation units or the order in which they appear in the application.

A presentation unit should be a small and discrete entity. Large presentation units, especially forms and selection lists, should be broken into smaller units, where one unit is the required portion of the larger unit, and the other units are optional portions.

Presentation units reference the data source from which they are populated. They do not actually hold any data, including static data, within their definition. A presentation unit may contain simple logic to allow alternate data references based on an attribute of the device. For example, if a device has a horizontal form factor with a wider screen than other devices in its class, a longer alternate title may be referenced for the device. The logic may also provide control over the number of items presented in lists, such as news headlines, based on a device attribute.

Using a mechanism to define layouts (described later), an application developer chooses a subset of presentation units for each page. Presentation units can be split onto separate pages, or not used at all, depending on the device. Their order may be different from device to device.

Presentation Layout

Each class of device has a different layout. A desktop browser application layout is different from a PDA layout, which is different again from a cellphone layout. In each case, a diverse subset of presentation units is used on the different pages of the application for each class of device.

Within a class of devices, individual devices may require a separate layout from the general class layout because of the form factor of the device. For example, the Ericsson R380 smart phone has a horizontal form factor that is quite different from other devices in its class. For these phones, presenting tables in a horizontal layout is in all probability more desirable than presenting tables in a vertical layout.

Having a different layout for each class of device is very different from current application design for which one layout fits all. Application developers must now consider layouts for many different types of devices.

Application Development Platform

Clearly, without additional support from the application development platform, an application developer faces a daunting task when creating an application for the desktop browser-based environment and the many environments of mobile devices. Each new device presents its own set of quirks and differences. In fact, the task becomes even more daunting when you consider the other data-enabled appliances connecting to the Internet daily.

Note:

One such data-enabled appliance is interactive television. While the markup language is HTML and the screen is similar to that of the typical wireline computer screen, the user is situated several feet away from the screen, making it necessary to increase the font size and perhaps reduce the amount of content on each page. Additionally, the user may be using a remote control without a keyboard to interact with the application.

Without application development platform support, an application developer must do one of two things: either write a separate client layer for each device or write an application to the least-common denominator for the devices supported by the application. The first case creates a content management problem. When the application changes, the developer will have to modify every device layer to conform to the changes. The second case creates an application that is overly simple on most devices. Users will perceive the application as dull and go elsewhere.

A development platform must assist the developer in creating an application with as much reuse of code as possible, while providing a satisfying experience for users accessing the application on the different devices. The platform must allow the developer to access each channel of the device as well as each special feature without significantly compromising the ability to reuse as much code as possible. To this end, the platform should include the following required and optional features.

Required Features

Device Profile Database

The device profile database encapsulates required information about each device so that the platform can simulate how an application will look and behave during development, and applications can make intelligent decisions during run time, based on the device. Attributes of the device, such as the number of power keys, the microbrowser, the screen size, the graphical capabilities, and messaging protocol, are but a few of the pieces of information in the database. Within a few years, most of this information will be available in the HTTP headers of the incoming request messages. However, for non-IP channels like messaging, much of the information will still need to be available in a local database.

The devices profile database must be easy to update at run time without shutting down applications. Automatic updating of the database is highly desirable.

Presentation Meta-Language

The presentation meta-language, which serves to demarcate the individual presentation units described previously, must be flexible enough to support all of the markup languages (WML, HDML, cHTML, HTML), targeted from it. A runtime transformation engine will translate the meta-language into the target language from the presentation units for a specific device and add the navigation links to the page or power keys.

The W3C (World Wide Web Consortium) organization, is currently working to define the XHTML (Basic) module for mobile devices. Whether XHTML (Basic) will fulfill all the presentation meta-language requirements will not be known until the specification is complete.

Presentation Layout Mechanism

The presentation layout mechanism allows the developer to create a layout specific for each device. All, or a subset of, the presentation units can be selected and positioned in any order desired. Every object manipulated within the mechanism must be represented by a presentation unit. No additional static text is added using the presentation layout mechanism, which ensures complete separation of the layout from the presentation units and allows all presentation units to be reused in any layout. Any device or device-class specific presentation units, including static text labels, should be specified within the presentation meta-language.

The presentation layout mechanism uses the device profile database along with optional skins to create an accurate representation of the device on which the layout is performed. Layouts should be hierarchical with a default layout for every page.

The second level in the hierarchy should be a level representing the major classes of devices: cellphone, pager, PDA, and so on. Any modifications to the default layout for a class of device should be represented in this layer. One or more layers may exist beneath the device class.

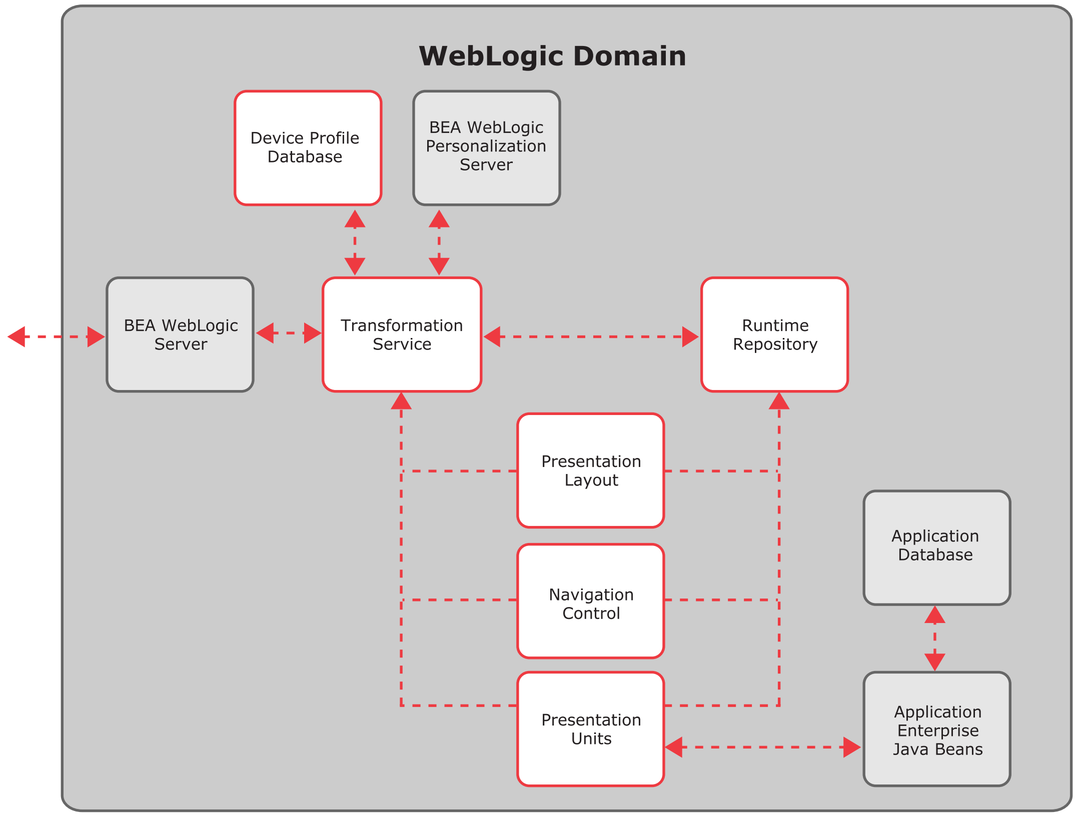

Figure 3: Runtime Data Flow – Web Channel.

Depending on the application, the next layer could be based on the general model or manufacturer of a device. Alternatively, the following layer could be based on the microbrowser, with the subsequent layers representing the models and sub-models of devices. Possibly no device-specific layer exists if the desired behavior is covered in previous layers of the hierarchy.

With a default representation of each page, new devices should be able to access a site before the device profile database is updated to include the new device and before any necessary new layouts are created; however, the site would not be optimized for the device. Plugging in new device skins as they become available should be easy.

Like the device profile database, incorporating new device layouts in a running application should also be easy, without needing to bring the application down to install them. If the application is well designed, the layout and the navigation flows should be the only aspects of an application that may have to be written differently for each device or class of device.

If a site contains many similar pages having the same layout but different data, application developers should consider creating templates to maximize code reuse. Given the extremely small size of the cellphone screen and the often-limited page size that cellphones can handle, a few well-designed templates should satisfy a high percentage of uses for this class of device.

Navigation Flow Mechanism

The navigation flow mechanism allows application developers to create nodes that map to pages of an application. Navigation flows are generally applied to the device class and not to the individual devices. Links are created between nodes and given key mapping semantics.

For example, consider the semantics for a link from one node to the next logical node in the following cellphone class navigation flow:

> *Map the link to a power key labeled <NEXT>, where this key has the highest precedence if more than one link with power key mappings is available. If a power key is available, it will be labeled <NEXT>, and it will take the user to the next logical page. If no power key is available, the directive is to create a link at the bottom of the page.*

Messaging API

For asynchronous messaging, the application development platform must provide a single canonical or unified API covering the many different types of messaging, including SMS, WAP Push, AOL-IM, Paging, and Email. The mechanism underneath the API should be a plug-in architecture enabling new protocols to be added easily as they are defined.

No assumption should be made as to where the messaging gateway is hosted. It could be hosted by the enterprise or by a third-party service provider. The API should be flexible enough to allow a message to be sent to an alias that may translate to a single user or a group of users.

Location API

The location of a device is determined by the telecommunications carrier, which uses a mechanism that triangulates from cell towers within reach of the device and returns a location that can be as accurate as a city block or as inaccurate as a few thousand meters. Each carrier establishes its own representation of location data because no standard representation yet exists.

The location of a device may also be determined using a device-assisted mechanism, such as global positioning system (GPS). The accuracy of device-assisted GPS is usually a few meters.

The application development platform must provide a single canonical or unified API covering all location mechanisms. The API should be a plug-in architecture so that new location mechanisms and representations can be easily added as they become available. Some notion of the granularity of the location data should be considered. No assumption should be made as to the type of mechanism from which the information was gathered.

Personalization

In addition to the standard personalization information, such as preferred language or currency, personalization of an application should also be available based on the device that a user is using. The information that a user is interested in on one device (say, a desktop browser) may be very different from the information that the user is interested in on another device (say, a cellphone). The application development platform should permit the user to create several portals based on device type.

Additionally, a user should be able to personalize an application based on the location of the device. For example, consider the following location-based personalization:

> *When a device is more than twenty miles from home, local dry cleaning establishments appear on the home page. When the device is within twenty miles from home, no dry cleaning information appears.*

A user may want a certain portal when visiting on the East Coast and a different portal when visiting on the West Coast, and so on.

The personalization mechanism should also allow for the sending of alerts to different devices based on the time of day or the day of the week. For example, if a user does not carry his or her pager on the weekends, a voice alert could be generated and left on an answering machine.

Optional Features

Transcoding Mechanism

An extremely fast way to enable an existing Web application on devices other than through a wireline desktop browser is to use a technique called transcoding. Transcoding is the process of selecting presentation elements from an existing Web site and mapping them into new data flows and layouts. Transcoded applications cannot take advantage of the special features of a device, such as messaging channels or the location of the device, because these features have no meaning to the original desktop application from which the transcoded application was formed. Transcoded applications also suffer performance degradation because the entire desktop source page must be generated for transcoding to work, regardless of whether an entire desktop page or just a few lines of text are used from the source on the target device. The same transformation engine used to transform the presentation meta-language into the appropriate markup language for the device is used to transform the transcoded presentation units into the appropriate markup language.

The transcoding method just described is developer-assisted transcoding, where a developer must physically select presentation elements to be used for the target device. Another transcoding method called mechanical transcoding requires no developer interaction at all. Mechanical transcoding usually produces useless results, and is not generally considered a viable method for enabling non-desktop devices.

Provisioning API

Over-the-air (OTA) provisioning allows an application or an administrator to modify the physical settings on a wireless device remotely. Since each wireless gateway has a different API for modifying the physical settings, the application development platform should provide a single canonical or unified API covering all of these API. The mechanism underneath the API should be a plug-in architecture so that new OTA provisioning mechanisms can be easily added as they become available.

Application Runtime Platform

Once the client portion has been developed for the desktop browser and the many other data-enabled devices, BEA WebLogic's application server runtime platform will provide the optimal execution environment for the application. The platform provides all the features such as scalability, reliability, and portability that applications have grown to rely on from BEA WebLogic. It caches assembled pages for a specified period, which yields better performance than dynamically creating pages upon each request.

BEA WebLogic is a complete environment for serving multiple devices. Some of the additional services described in the follow list may assist the application in its execution.

Transformation Service

A transformation service deployed in a WebLogic cluster will enable an existing or new application Built on BEA™ to be extended to any data-enabled device. The transformation engine determines the specific device making a request, selects the appropriate navigation model, presentation layout, and required presentation units, and then combines all of these constituents into a page written in the correct markup language for the device. The transformation engine handles both meta-language source code and transcoded source code.

The transformation engine will also use information from the device profile database to determine where pages must be split due to page data size limitations. It will break the pages and add appropriate forward and backward links, as well as retain context if necessary.

Personalization Server

An application using BEA WebLogic Personalization Server™ will be further enabled to provide content tailored to user needs and adapt content based on the type and location of a device. A user having several devices may choose to construct several personalized portals. For example, a user having a cellphone, a PDA device, and a desktop computer, may choose to create personalized portals for each class of device, such as a minimal-content portal for the cellphone, a medium-content portal for the PDA, and a full-content portal for the desktop computer. A user may also choose to create personalized portals based on location, such as a portal for New York, a portal for San Francisco, and a portal for Osaka.

In addition, an application using BEA WebLogic Personalization Server will be enabled to provide personalized navigation of a Web site based on previous visits. By predicting the most likely purchases or navigational choices a user is expected to make, the interaction with the device can be reduced, thus improving the user's experience on the site

Messaging Gateway

A messaging gateway handles protocol conversions between the wireless carrier's server (for example, the SMSC) and the application. Some messaging protocols are IP-based (for example, WAP Push, AOL-IM, and SMTP) and some are not (for example, SMPP, CMID2, UCP/EMI, and SNPP).

Mobilized enterprises use either a private wireless network or their own messaging gateway to handle two-way messaging. In either case, a messaging gateway must be able to handle all the messaging protocols used by the enterprise-supported mobile devices.

For messaging protocols not carried over the IP network, an enterprise must individually contract with the telecommunications carriers worldwide already providing service to any of the enterprise users, to send and receive these messages through their messaging networks. Fortunately, companies providing hosted services for two-way messaging have contracts with each of these carriers, so an enterprise needs only one contract with one of these companies to obtain global coverage. If only IP-based messaging protocols need to be supported, these special carrier relationships are usually not necessary.

Wireless Gateway

A wireless gateway, sometimes called a wireless protocol gateway or wireless server, converts the over-the-air protocol (such as WML/WAP) to the application's protocol (WML/IP). To provide service to mobile devices, a telecommunications carrier hosts one or more wireless gateways.

For a secure connection employing the WAP protocol, a very small security risk exists at the WAP gateway during the switching of WTLS (WAP side) to SSL (IP side) and SSL to WTLS. Since the WAP protocol allows a session to be redirected from the carrier's gateway to the enterprise's gateway, an enterprise may want to control this minimal risk by including a WAP gateway behind its firewall. The enterprise secures the server running the WAP gateway in a controlled environment to reduce exposure to the security risk.

Since the carrier is a trusted entity and is continuously responsible for protecting voice, fax, computer and other types of data, enterprises most likely do not need to host their own gateway.

Voice Services

Supporting a voice portal requires software services to perform voice recognition as well as services to convert text to speech. Unless VoIP is available, the enterprise must also procure telephony hardware to convert analog voice input to digital VoIP.

Acquiring and maintaining the equipment needed to support a large scalable voice portal can be very costly. However, companies are available that provide hosted services with this hardware. They arrange for a toll-free number into their server pool, convert the analog data to and from VoIP, and pipe the VoIP data to and from the enterprise application.

The severing of the wire tether to the Internet has opened up a whole new world of opportunity for telecommunications carriers, wireless service providers, handset manufacturers, and application developers. But not just members of the wireless community are producing data-enabled devices: members of the wireline community are busy developing their own data-enabled devices such as interactive television. Common to both communities are the application developers, who must now look at developing Web applications in a completely new way. There to help them is the BEA WebLogic application server runtime platform, which provides a complete environment for serving the traditional desktop browser and the many data-enabled devices available on the market today. Application developers need only write a new multi-device client and add a transformation service to a BEA WebLogic Server™ cluster to make an application available for any Internet-enabled wireless or wireline device. Alternatively, customers may choose to work with BEA Wireless Star Solution partners. The complete listing of these partners can be found at: **http://bea.com/partners/wireless_star_solution.shtml**.

About BEA

BEA Systems, Inc. (Nasdaq: BEAS) is one of the world's leading e-business infrastructure software companies, with over 11,000 customers around the world, including the majority of the *Fortune* Global 500. BEA and its WebLogic® brand are among the most trusted names in e-business. Businesses built on the award-winning BEA WebLogic E-Business Platform™ are reliable, highly scalable, and poised to bring new services to market quickly. The BEA e-business platform is the *de facto* standard for over 2,100 systems integrators, independent software vendors (ISVs) and application service providers (ASPs) to provide complete solutions that fast-track and future-proof e-businesses for high growth and profitability. Headquartered in San Jose, Calif., BEA has 93 offices in 34 countries and is on the Web at www.bea.com.

Glossary

3G	— Third generation
cHTML	— Compact HTML
GPS	— Global Positioning System
HDML	— Handheld Device Markup Language
HTML	— Hypertext Markup Language
HTTP	— Hypertext Transport Protocol
i-Mode	— NTT DoCoMo's OTA protocol
IVR	— Interactive Voice Response
OTA	— Over-the-Air
PDA	— Personal Digital Assistant
PKI	— Public Key Infrastructure
SMPP	— Short Message Peer to Peer
SMS	— Short Message Service
SNPP	— Simple Network Paging Protocol
WAP	— Wireless Application Protocol
WML	— Wireless Markup Language

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131 U.S.A.
Telephone: +1.408.570.8000
Facsimile: +1.408.570.8901
www.bea.com

CWP0297E0801-1A