Chapter 2

A Portal Blueprint

This chapter provides you with the concepts, definitions, standards, and architecture that form the basis on which all Java-based portal server software is built. Many of these concepts, definitions, standards, and architectures hold equally true for both WebSphere Application Server V5.x and WebSphere Portal server V5.x.

Concepts are basically good computing paradigms. *Definitions* are defined by the IT industry at large. *Standards* are based on widely accepted Java standards, some of which relate to portal servers. *Architecture*, though specific to WebSphere Portal, is really dictated by a combination of the portal server's software, the available hardware, and the goals of an enterprise.

J2EE Architecture in WebSphere Portal

This section introduces J2EE and gives you an overview of the J2EE runtime environment, just to reinforce some of the concepts. Since WebSphere Portal is based on Java, its architecture closely follows the J2EE architecture. As such, the terminology and concepts will seem familiar to those who already know the Java language.

If you are interested in simply using and working with WebSphere Portal, you may choose to skip this section. If, on the other hand, you want a more detailed

description of the Java-based standard, please refer to the information at the following Web sites:

- Java 2 Platform, Enterprise Edition home page, http://java.sun.com/j2ee
- IBM WebSphere Developer Domain, http://www.ibm.com/websphere/developer
- Portlet API JSR #168, http://www.jcp.org/jsr/detail/168.jsp

What Is J2EE?

Java 2 Platform, Enterprise Edition defines the standard for developing multitier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components; by providing a complete set of services to those components; and by handling many details of application behavior automatically, without complex programming.

These modular components are servlets, Java Server Pages (JSPs), and Enterprise Java Beans (EJBs). The set of services is provided by the containers in which these components operate, such as the Web server or the EJB container. As such, the architecture shown in Figure 2.1 lets a client indirectly access back-end enterprise applications while protecting enterprise-sensitive data.

J2EE comes with a standard application model and platform. It not only comes with a reference implementation, but also a compatibility test suite for products wanting to achieve J2EE certification.

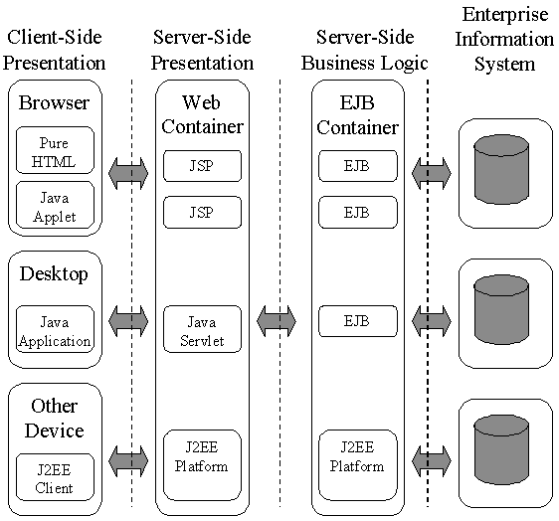

Figure 2.1: General J2EE architecture.

Although the J2EE architecture in Figure 2.1 does not show it, we recommend using the Model-View-Container (MVC) paradigm with respect to Portal. In that regard, the Web Server really is the Web Container. This is explained in a later section in this chapter.

J2EE Application Model

As mentioned, the J2EE programming model has four types of application components:

- Java application clients
- Applets
- Servlets and Java Server Pages (JSPs)
- Enterprise Java Beans (EJBs)

Each application component executes in a well-defined container, which is an execution environment within whose scope components run. Containers are built on the Java 2 Platform, Standard Edition (J2SE), and provide the runtime support. By having a container between the application component and the set of services, J2EE can provide a federated view of the APIs for the application components.

The primary component in a portal server application is the portlet. A portlet is a reusable component that provides access to Web-based content, applications, and other resources. Since a portlet is really an extension of a servlet, portlets run in the Web container, along with servlets and JSPs.

Table 2.1 lists the application components and the containers in which they run. The table also refers to JAR, WAR, and EAR files. These file types stand for "Java Archive," "Web Archive," and "Enterprise Archive," respectively. They are covered in detail later in this chapter.

Table 2.1: Application Components and Containers

Application Components	J2EE Containers	Container Notes
Application client	Application Client container	Packed as JAR files. Not required to manage transactions.
Applets	Applet container	Communicate over HTTP in a browser; can also use serialized objects.

Table 2.1: Application Components and Containers (continued)

Application Components	J2EE Containers	Container Notes
Portlets,servlets,JSPs	Web container	Create portlet/servlet instances, load and unload portlet/servlet, create and manage request and response objects. Packaged as WAR files.
EJBs	EJB container	Packaged as EAR files. Provide threading, transaction support, and data storage management.

Figure 2.2 shows the J2EE containers and the data flow from a portal server perspective. It shows the four kinds of containers that provide all the required services for the components they support, namely the Applet container, Application Client container, Web container, and EJB container. The components and containers are the same, with the additional component, a portlet, shown residing in the Web container.

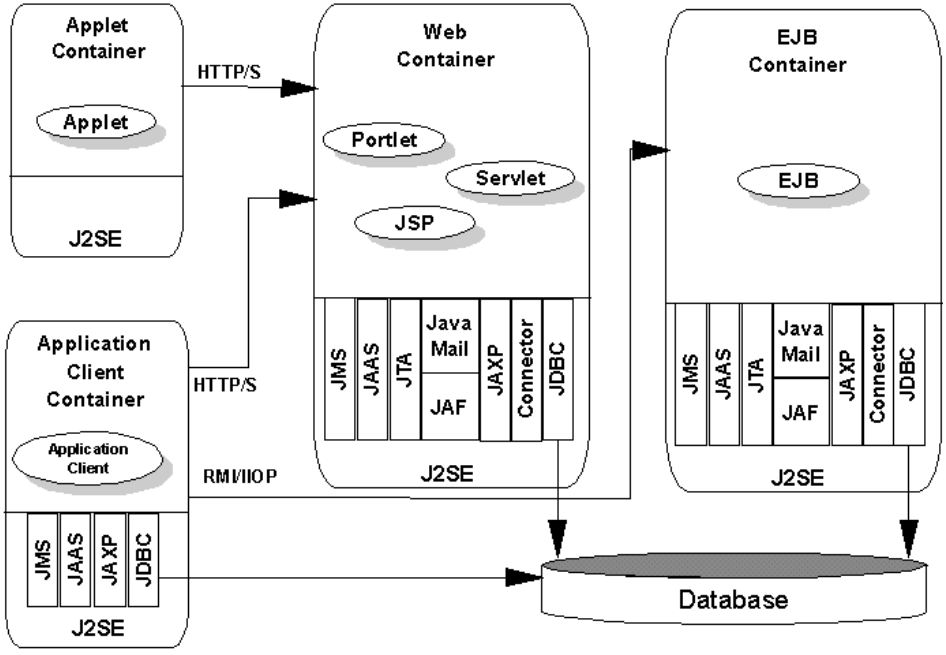

Figure 2.2: J2EE application model.

In addition to providing support for EJBs, servlets, and JSPs, the J2EE specification defines a number of standard services for use by J2EE components. Services provide integration with existing systems, including JDBC for database connectivity, Java Messaging Service (JMS), JavaMail, Java Authentication and Authorization Service (JAAS), Java API for XML Processing (JAXP), Java IDL, and Java Transaction Architecture (JTA). These services not only help Java and Web clients communicate with back-end legacy systems, they also can be configured to handle reliable and secure business transactions.

Figure 2.3 takes a step back, to look at the entire Java platform. You can see how the various components and services fit along with the Java 2 SDK (Software Development Kit).

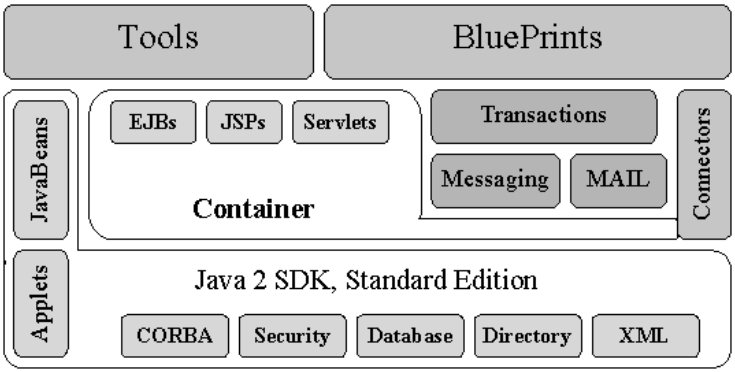

Figure 2.3: J2SE component and service model.

You get a simplified architecture, based on standard components, services, and clients, that takes advantage of Java's write-once-run-anywhere technology. This architecture offers scalability to meet the computing demands of even the largest configuration. It also offers a unified, flexible security model, which is critical to many enterprises.

This passing reference to Java 2 SDK does not do justice to the revolutionary Java platform introduced by Sun, Inc. Suffice to say that it is a stable, secure, and feature-complete development and deployment environment that provides software developers a cross-platform compatible and rapid application development platform. More information on J2SE can be obtained from http://java.sun.com/j2se.

J2EE Platform Roles

The J2EE platform defines six distinct roles for use during the application development lifecycle:

- Product provider
- Tool provider
- Application component provider
- Application assembler
- Application deployer
- System administrator

Product providers and tool providers have a product focus. Application component providers and application assemblers focus on the application. Application deployers and system administrators focus on the runtime.

If you mapped the J2EE roles to the tasks in the portal world, you would have the following categories:

- Product provider
- Tool provider
- Portlet application assembler
- Portlet deployer
- Portal Administrator

The roles are similar, but not identical, because in the portal world, they help identify the tasks that need to be performed by the people working on a Java application, namely the portal. There is no role corresponding to Application component provider because portals integrate existing applications. Roles are analogous to privileges within the portal. Users are grouped together. Then, individual users or groups are assigned certain privileges via Access Control Lists (ACLs).

J2EE Compliance

WebSphere Application Server V5.x is fully J2EE certified. Since WebSphere Application Server serves as the engine or the platform for WebSphere Portal server, which is an application within the application server, all the benefits of J2EE compliance are available for Portal Server. WebSphere V5.0 also provides Web Services support, Connector Architecture, and JMS/XA interface to IBM MQ

Series. The Connector Architecture and Java Messaging Support (JMS) are useful in middleware space. Web Services helps with creating and accessing portlets on remote systems and communicating with components running on non-Java systems like .Net.

Table 2.2 lists the various API levels supported by the base version of WebSphere Application Server V5.x. The basic tenet of the Portlet API is that it will be based on the Servlet specification. It is also envisioned that the Developer API will be similar to the Servlet API. In reality, the Portlet API will be something of a subset of the Servlet API.

Table 2.2: WebSphere Portal and J2EE Compliance

J2EE Items	APIs	WebSphere Portal V5.1
Components	Portlet*	JSR168
	Servlet	2.2
	JSP	1.1
	EJB	1.1
Services	JDBC	2.0
	JTA/JTS	1.1
	JNDI	1.2.1
	JAF	1.0
	XML4J	3.1.1
	XSL	2.0
Communication	RMI/IIOP	1.0
	JMS	1.0.1
	JavaMail	1.1

**The Portlet API submitted as Java Specification Request (JSR) 168 has been accepted as a standard. More information can be obtained at* http://www.jcp.org/jsr/detail/168.jsp.

MVC Paradigm and Architecture

The programming model for portlets, servlets, and JSPs is based on the model-view-controller (MVC) model. In the MVC model, the data itself (the model), the presentation of the data (the view), and the logic manipulating the data (the controller) are designed to be independent. If the view needs to change, the business logic and the data are not affected. If the data interface changes, the controller can be updated without affecting the view.

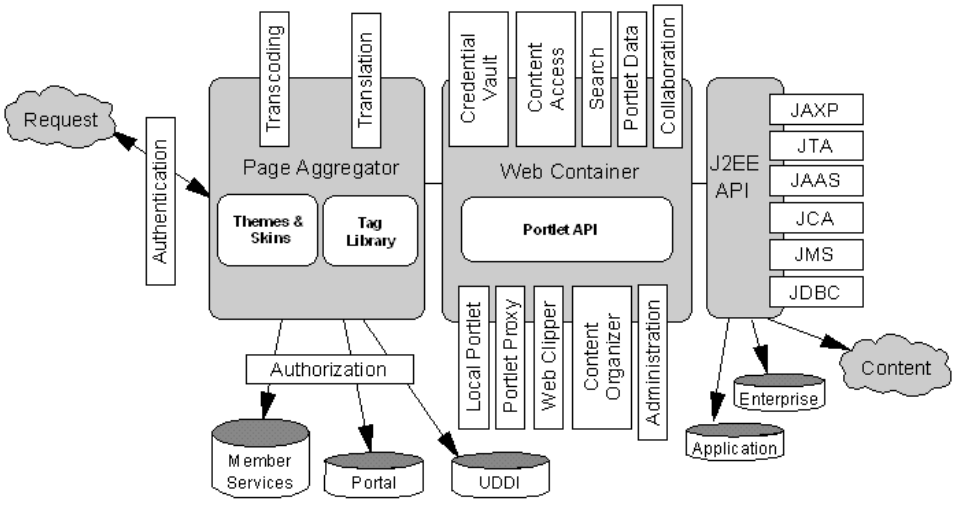

Figure 2.4: Overall portal architecture.

The overall portal architecture is shown in Figure 2.4. In keeping with the J2EE architecture, separation is present between the presentation or view layer (the Page Aggregator), the controller layer (the Web container, where the portlets reside), and the model layer (the J2EE API, which really deals with integrating with the back-end systems).

In the MVC model, the portlet receives a request from a Web client, accesses the data through a set of reusable components (beans or EJBs), and invokes a JSP component to display the results of the request. The sequence of events and data flow is shown in Figure 2.5.

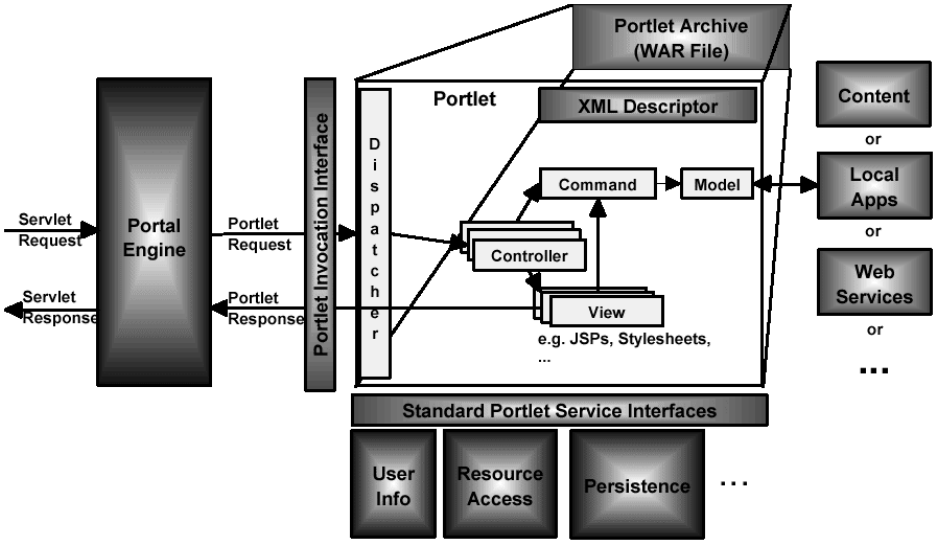

Figure 2.5: Portlet execution cycle.

Packaging J2EE Applications

Earlier in this chapter, we mentioned Java Archive (JAR) files. J2EE components, like servlets and portlets, are packaged into modules. Modules are then packaged into applications, and applications are then deployed. Each module and application contains a J2EE Deployment Descriptor (DD), and is packaged up as an archive file. A DD basically lists the contents of an archive file and contains the entire file structure. Software development and assembly tools use Deployment Descriptors to validate a package.

Figure 2.6 shows the file structure of a J2EE enterprise application. It is commonly known as an Enterprise Archive (EAR) file. The arrows in Figure 2.6 show what is contained at each level. An EAR file can contain one or more JAR files, and/or one or more Web applications. Web applications are packaged as Web Archive (WAR) files. These files show up in the UNIX and Windows file systems with the file extensions *.ear*, *.jar*, and *.war*.

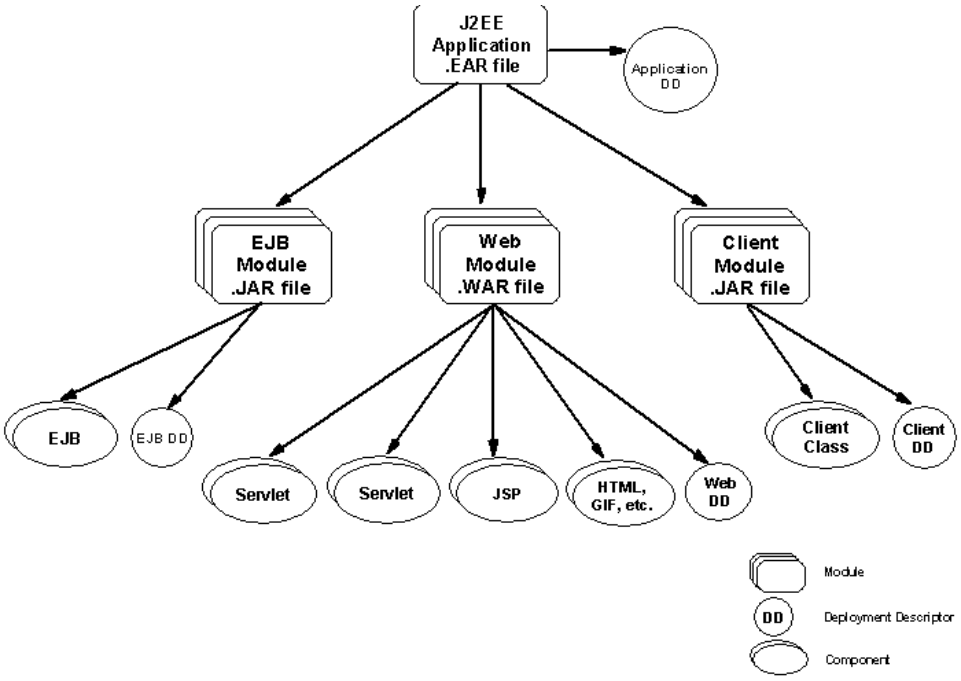

Figure 2.6: J2EE Application Archive file structure.

In the case of a portal application, one or more portlets (possibly JSPs), and/or servlets and accompanying image files make up the application. These portal applications or portlets are packaged up as a WAR file. The Deployment Descriptor for the portlet application is in a file called *web.xml*. Table 2.3 lists the contents of a sample WAR file.

Table 2.3: Sample Portlet WAR File Contents

File	Description
/PORTLET.war/META-INF/MANIFEST.MF	Standard JAR file manifest
/PORTLET.war/WEB-INF/web.xml	Web application descriptor is a mandatory item in a J2EE Web archive. It provides the application server with information about the Web resources in the application.

Table 2.3: Sample Portlet WAR File Contents (continued)	
/PORTLET.war/WEB-INF/portlet.xml	The portlet deployment descriptor provides the portal server with information about the portlet resources in the application, including configuration, support characteristics, and localized titles.
/PORTLET.war/WEB-INF/lib	Directory containing required JAR files
/PORTLET.war/WEB-INF/classes/MYPORTLET.class	Portlet class file
/PORTLET.war/PORTLET/MYJSP.jsp	JSP file or files
/PORTLET.war/images/MYIMAGE.gif	Image file

The Portlet Application Programming Interface (API) is explained in detail in Chapter 5. That chapter shows how to code portlets, build portlet applications, construct Deployment Descriptors, and package it all.

Topologies

"Topology," in the context of this chapter, refers to the mapping of software components to available computer hardware. Other than cost, the primary factors that affect topology are:

- Performance
- Availability
- Security
- Maintainability

WebSphere Portal is another application running within WebSphere Application Server, so all the things to watch out for and the numerous benefits about the various WebSphere Application Server topologies apply. The topologies presented in this section should look very similar to the WebSphere Application Server V5.1 topologies. WebSphere Portal topologies typically involve tier 2. These come in one-node, two-node, and three-node configurations. Tier 2 is where the Web container, with the portlets, logically resides.

Note: WebSphere Portal V5.1 needs WebSphere Application Server V5.1 to run on the same machine.

The following abbreviations are used in the topology diagrams:

- WAS – WebSphere Application Server
- PZN – Personalization Server
- WPS – WebSphere Portal server
- BPC – Business Process Container
- LDAP – Lightweight Directory Access Protocol
- JSP – Java Server Page
- EJB – Enterprise Java Bean
- DB – Database
- LIMWC – Lotus Instant Messaging and Web Conferencing (formerly Sametime)
- LWWCM – Lotus Workplace Web Content Management
- LTW – Lotus Team Workplace (formerly QuickPlace)

The topology diagrams depict the middle tier hosting the application server with a "shadow" box. This suggests that more than one application server could be running on the same machine. In fact, the additional server or servers might be identical copies of the original server, known as *clones*.

One-Node Configuration

Figure 2.7 shows the simplest configuration. All software components are installed on a very powerful multi-CPU machine, with at least 2 GB of memory and a minimum of 6 GB of storage. Please refer to the WebSphere Portal server performance tuning guide from IBM for not only memory and disk requirements but for key tuning parameters.

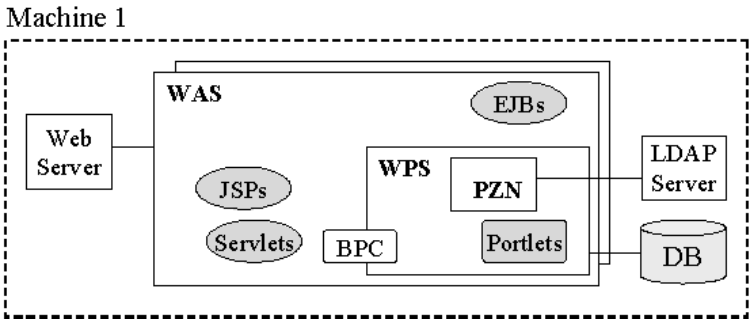

Figure 2.7: WebSphere Portal configuration on one node.

This one-node configuration works well as a development or test environment, it would not be robust enough for a mission-critical production environment. This hardware setup is easy to configure, and it does provide for software process isolation.

Although you get full utilization of the machine's computing power in this configuration, the application server, database server, and directory server will be competing for CPU and memory. Therefore, in a production environment, we highly recommended that you separate the middle tier (the application server) from the backend servers. This helps in system scaling and management and can be used to set up a failover configuration.

Most enterprises will already have a database server and/or a LDAP directory server. A portal server would be an addition to this environment. That would suggest having the database server separate from the application and portal servers. Putting not only the LDAP server on a separate node but even running the database on a separate server is highly recommended.

Note: If your configuration uses Domino Server, we suggest that you install the Domino components on a separate machine. It is a good idea to separate the old Sametime (LIMWC) and QuickPlace (LTW) servers.

Figure 2.8 shows a variation of the basic "everything-on-one-node" configuration. In this example, all of the Domino components are on one or more physical machines, separate from the other servers. Isolating the Domino components provides the following benefits:

- The components are more easily managed.
- Provides for independent maintenance cycles and OS patch levels.
- Components like Lotus Team Workplace (LTW), formerly known as QuickPlace; Lotus Instant Messaging and Web Conferencing (LIMWC), formerly known as Sametime; and Lotus Workplace Web Content Management (LWWCM), which replaces the old Web Content Publisher, do not have to compete for the same resources as the core portal processes.

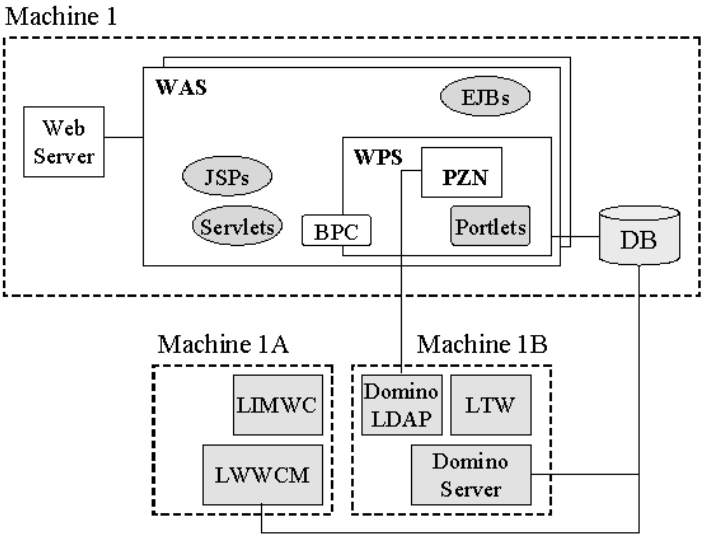

Figure 2.8: WebSphere Portal configuration on a one-node variant.

This is the "one-node variant" configuration, which is not to be mistaken for highly available or high performance configuration. The machine (or machines) hosting the Domino components simply denote parts of the portal topology that reside on a different machine. The database, denoted as DB, comprises the WebSphere Portal server database (WPS DB), the WebSphere Member Management database (WMMDB), Personalization (PZN)-related databases (FDBKDB and LMDB), and the Document Manager DB (JCRDB).

Two-Node Configuration

When properly designed, the two-node configuration really means separating out the back-end database server (tier 3), to run on a physically separate machine. Since Portal Server has to use an LDAP server for security purposes, the database

server could co-host the LDAP server too, as shown in Figure 2.9. In large production environments, as stated before, we recommend running the LDAP Server on its own separate node.

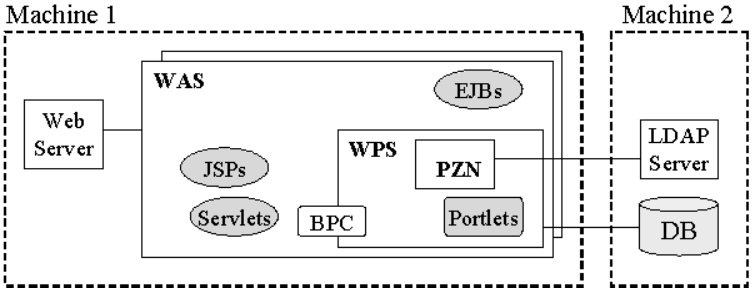

Figure 2.9: WebSphere Portal configuration on two nodes.

This is a better configuration because all database-related activity is confined to one or more machines in tier 3. Tiers 1 and 2, made up of the Web server, the application server, and the portal server, all exist on one machine. When any LDAP access is needed, or when back-end database access is required, the network hops from node 1 to node 2. Otherwise, simple servlet processing and the presentation of the JSPs are all taken care of on the "local" node.

Separating the database server from the application and portal servers represents a recommended best practice for the following reasons:

- Separating WebSphere Application Server and WebSphere Portal Server from the database server helps performance under heavy loads. Otherwise, full utilization of computing resources could become an issue; as resources get scarce, a degradation of performance might occur.

- With the database server separated, those components of tier 2 that are related mainly to the application server can use their cloning facility. WebSphere Application Server could be vertically or horizontally cloned without affecting the database server. This helps improve performance and assists with failover.

- Usually, database servers are already configured in a highly available (HA) manner, so putting Portal Server on the same machine would represent a single point of failure. Thus, separating the application server and the portal server from the database server maintains the HA architecture.

- High availability of the database server goes hand in hand with backup and restore. Enterprises usually have good backup and restore procedures in place for their back-end data. Housing the portal server's application data on the database server ensures that the portal data is always backed up.

- Database servers are generally optimized and well tuned. Again, housing WebSphere Portal server's data on the database server helps portal performance. Additionally, if a site makes extensive use of Personalization, that would require the use of a shared database.

We are suggesting a "two-node variant" configuration, separating the Domino components onto one or more machines, as shown in Figure 2.10. Separating the components to run on different machines also makes performance tuning a lot easier. If the enterprise site makes heavy use of the Web Content Management component, we suggest running LWWCM with Domino Server and Domino LDAP server on one machine and running the Lotus Team Workplace and Lotus Instant Messaging and Web Conferencing servers on different machines.

Note: In a production scenario, it is recommended that the two products—Lotus Team Workplace (LTW) and Lotus Instant Messaging and Web Conferencing (LIMWC)—not be co-located on the same physical server because of resource contention issues.

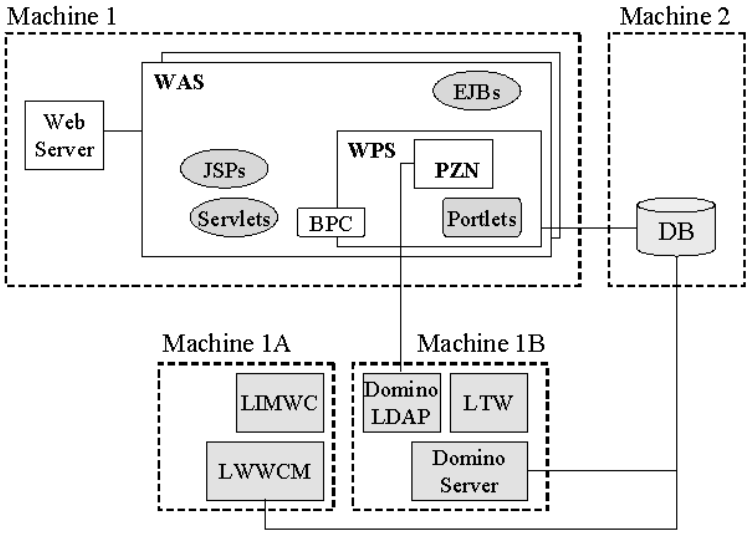

Figure 2.10: WebSphere Portal configuration on a two-node variant.

Tip: Although we do not discuss performance and capacity planning in this book, the database server and LDAP server need their own capacity planning, based on projected loads throughout the cluster and from other applications using the database and LDAP as well.

Three-Node Configuration

Three-node configuration is the classic three-tier setup that was introduced with the advent of application servers. In this scenario, tier 1 (the presentation tier), tier 2 (the business and application logic tier), and tier 3 (the back-end data store) are all running on separate machines, as shown in Figure 2.11.

This separation is possible because of the WebSphere V5.x HTTP plug-in. The HTTP plug-in behaves very much like a reverse proxy, but it is really using the HTTP transport to communicate between the Web server and the application server. The plug-in may also perform static content caching for certain HTTP servers, such as IBM HTTP Server, on certain platforms. The HTTP plug-in supports the clustering of and workload management (WLM) on application servers. During heavy loads, the Web server can send requests to multiple application-server machines. The HTTP plug-in provides for both vertical and horizontal scaling of the WebSphere environment.

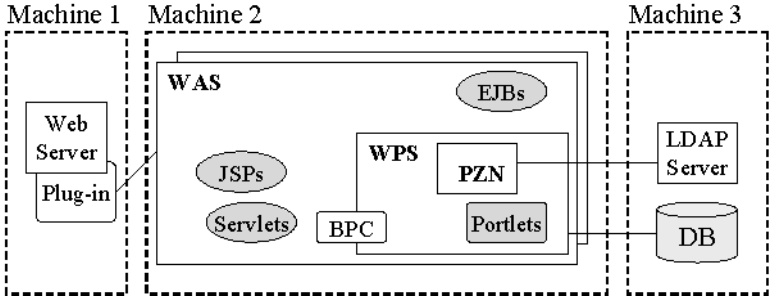

Figure 2.11: WebSphere Portal configuration on three nodes.

The HTTP plug-in also supports data encryption between the HTTP server and the application server, using HTTPS or HTTP over SSL. This makes the HTTP plug-in suitable for environments with two firewalls, such as a demilitarized zone (DMZ), where all network communication must be encrypted. Splitting the HTTP server is useful for sites serving static content from the HTTP server, as opposed to the

WebSphere Portal server doing that. The separation is also useful for sites performing Secure Sockets Layer (SSL) encryption/decryption at the HTTP server, because any form of encryption adds to the burden of the Central Processing Unit (CPU).

If you use the Domino Server, we suggest a "three-node variant" configuration, separating the Domino components as shown in Figure 2.12.

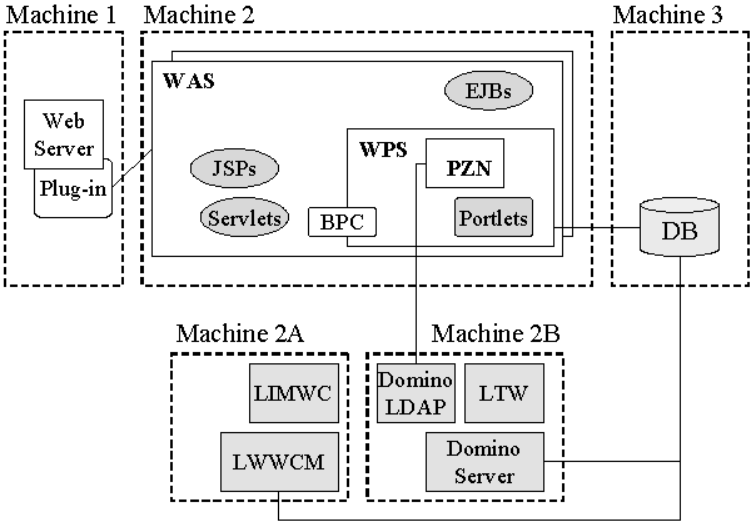

Figure 2.12: WebSphere Portal configuration on a three-node variant.

Notes about the Configurations

Thus far, the topologies in Figures 2.7 through 2.12 show the middle tier hosting the application server. The "shadow" box suggests that more than one application server could be running on the same machine. In fact, the additional servers are identical copies, or clones, of the original server. At this point, we have to introduce some WebSphere Application Server V5 concepts involving the WebSphere Application Server Network Deployment (ND) product. With ND, you can run multiple WAS instances and manage them together as a single *cell*.

One *deployment manager* process runs for the cell, which provides a central point of administrative control for all WAS instances. A cell's deployment manager

communicates with all related node agents to propagate and synchronize configuration information across the cell.

A *cluster* is a set of WAS instances within a cell that has the same applications deployed on them. A WAS instance that is a member of a cluster is sometimes referred to as a *clone.* When creating a cluster, one of the options is to create a *Replication Domain*. This option is used for memory-to-memory replication for sharing persistent session data across servers and for enabling dynamic caching of servlets and JSPs. This method of providing session data failover is new in WebSphere Application Server V5.

A *replicator* is a WebSphere Application Server run-time component that handles the transfer of internal WebSphere Application Server data. Replicators operate within a running application server process. You must define replicators, as needed, as part of the cluster management.

WebSphere Portal Implications

WebSphere Portal uses replicators for dynamic caching and memory-to-memory session replication. Enabling replication for dynamic caching in a WebSphere Portal cluster environment is absolutely necessary to maintain data integrity between various WebSphere Portal nodes in the cluster. Replication also helps improve performance by generating data once and then replicating it to other servers in the cluster. Therefore, a replication domain with at least one replicator entry needs to exist for WebSphere Portal.

Vertical cloning refers to situations in which the clones all run on the same machine. *Horizontal cloning*, on the other hand, refers to situations in which clones are created to run on another machine. WebSphere Application Server supports both forms of cloning. It follows, therefore, that Portal Server also can be cloned.

Cloning works well with the three-node configuration (when tier 1, tier 2, and tier 3 are all running on separate machines). For example, if you were to introduce one vertical clone of the application server into Figure 2.12, the new layout would look similar to that in Figure 2.13.

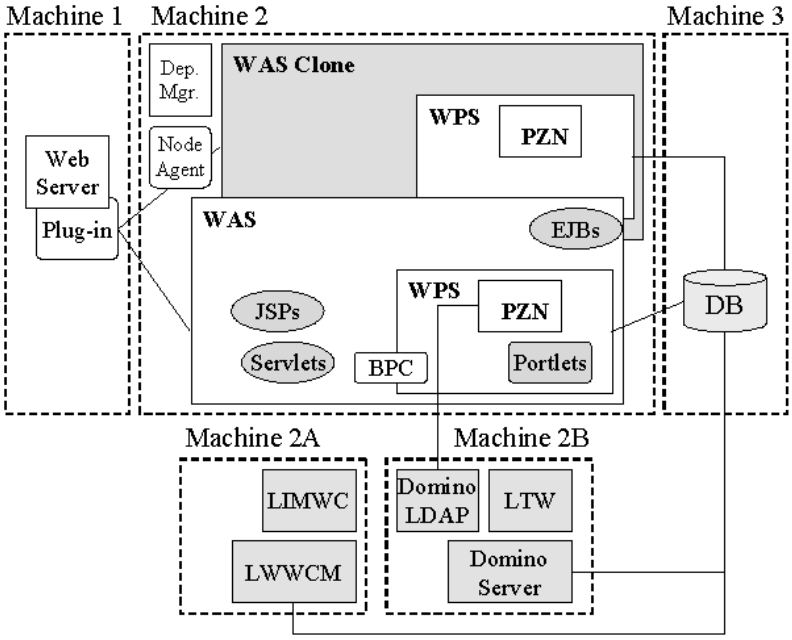

Figure 2.13: WebSphere Portal configuration on a three-node variant, with vertical cloning.

Note: In production environments, WebSphere Portal server requires global security to be set in WebSphere Application Server. That forces you to place common JAR files in a shared directory when you implement cloning.

Vertical cloning provides application failover within the same physical machine. If you have a very powerful multiway processor, vertical cloning allows you to make maximum use of the computing resources of the machine. Also, if you are memory constrained, it allows you to open multiple JVMs to take advantage of system memory. However, use this approach only if you have the CPU capacity to support multiple clones.

Horizontal cloning, that is, running the application server clone on a separate machine, would make the layout look something like that in Figure 2.14. Cloning, as such, mainly involves components in tier 2—the application server layer. (Figure 2.13 shows Domino Server, whereas Figure 2.14 shows LDAP Server as it relates to any of the common ones like Tivoli Directory Server or SunOne

Directory Server or Novell eDirectory Server. It is not our intent to confuse the reader, but to show you configurations using different software components.)

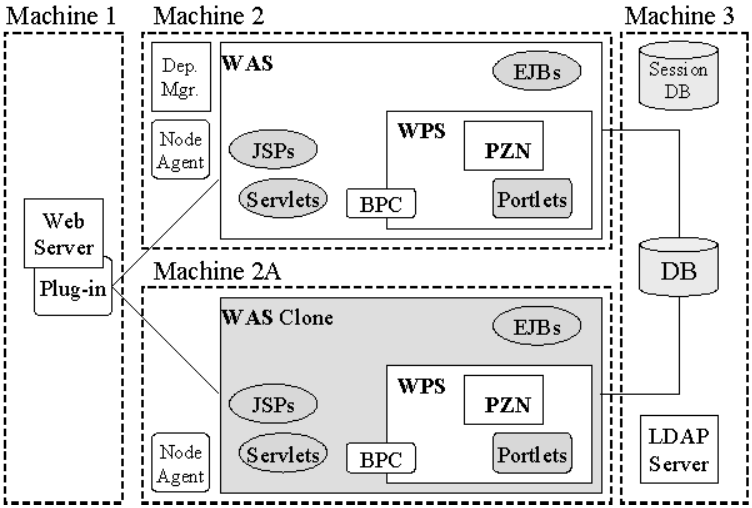

Figure 2.14: WebSphere Portal configuration with horizontal cloning.

Horizontal cloning provides not only application failover, but also hardware failover. The only requirement is that the application (in this case, Portal Server) must be distributed across multiple machines.

Tip: Vertical and horizontal cloning within the same configuration is also possible. However, it is beyond the scope of this book.

Figures 2.13 and 2.14 show Personalization components in both clones of a clustered environment. We want to point out that WebSphere Portal provides two property files that you can modify to customize the Personalization feature. These files are not managed by the WebSphere Application Server deployment manager. This means that if you make any changes to these files on a node in the cluster, those changes are not transferred to other nodes when you perform a synchronization of the cluster members. Instead you must manually copy the following properties files to each node in the cluster:

- <WPS_HOME>/shared/app/config/services/PersonalizationService.properties
- <WPS_HOME>/shared/app/config/services/FeedbackService.properties

On the topic of clustering, please refer to the excellent documentation that is available in the WebSphere Portal online help facility.

Other Configurations

Later in this book, you will learn about components that are part of WebSphere Portal Extend edition, like Search and Site Analyzer. If an enterprise is running these and other WebSphere Portal software components, what is an appropriate configuration? In other words, which software components can and should be co-located?

Because WebSphere Portal server is an application in WebSphere Application Server, the technical nuances that affect a complex WebSphere Application Server configuration are relevant even for WebSphere Portal server.

In a WebSphere Portal Extend configuration that includes Extended Search Server and the Site Analyzer there will be at least four extra servers. Our intent is not to sell you more hardware by having Lotus Team Workplace (LTW), Lotus Instant Messaging and Web Conferencing (LIMWC), and Lotus Workplace Web Content Management (LWWCM) on three different machines with Domino servers. It is to drive home the point that most of these software packages can be isolated and run as "independent" components. Some software components like Lotus Team Workplace and Lotus Instant Messaging and Web Conferencing, *must* be on different machines because those products compete for similar resources and resource conflicts could occur. You also have the option of co-hosting software packages and even running them on different operating systems (OSs) within the same configuration.

Installing and configuring a complex configuration that has multiple WebSphere Portal clones on multiple nodes along with Web Content Manager and all the Lotus components plus Site Analyzer, requires a lot of planning and takes time to set up. More important, it also requires careful management from the perspective of keeping things synchronized and within the confines of enterprise security.

Some companies prefer to separate intranet users from internet users by having dedicated servers that serve up totally independent content. Other companies have common content, and they maintain the separation via users, groups, and virtual hosts. The latter scenario is becoming more common with enterprises, especially given the ability to create virtual portals, which is covered in one of the later chapters.

Process Portals

All the topology diagrams thus far depicted a box named BPC that seems to span both WebSphere Application Server and WebSphere Portal Server. This is the Business Process Choreographer component that is part of WebSphere Business Integration Server Foundation (WBISF) V5.1. WBISF is one of the new components in WebSphere Portal V5.1. The intent was to indicate that the Business Process Container (BPC) can be configured either on *WebSphere_Portal* server or on the default *server1* of WebSphere Application Server, which would be our recommendation.

No drastic changes are necessary in the topologies, because WBISF is an integral part of WebSphere Application Server and requires it to run. The recommendation is to configure the Business Process Container on a server other than on *WebSphere_Portal* so that the portal server is not overloaded. With that said, Figure 2.15 shows how the one-node configuration would look. It is important to note that Personalization (PZN) and WebSphere Global Security play an important role in Process Portals.

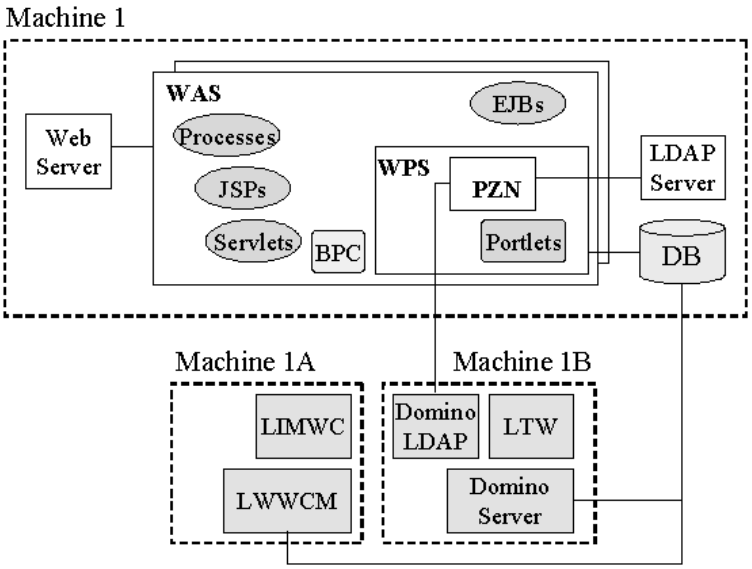

Figure 2.15: A Process Portal configuration in a one-node variant topology.

So what is a process portal? It is a corporate portal that is designed to present the right tasks to the right people at the appropriate time through a consistent and easy-to-use personalized interface. It benefits enterprises that have business processes.

Business processes can be noninterruptible or interruptible. Typically, *interruptible processes* are long processes that might require human intervention, such as a loan approval process or a parts ordering process or vacation approval. Processes like these that have human tasks can be presented via a portal. Thus, a process portal makes use of the underlying process workflow, along with the personalization engine and overall security infrastructure, to present the appropriate task to the user when he is signed into the portal. Process portals are discussed in Chapter 8.

In designing a topology for a process portal, the architect might like to maintain a separation of the different layers, especially the presentation layer and the process layer. An example topology is shown in Figure 2.16. Such a design also allows you to tune the portal machine (Machine 2) in a way different from the process machine (Machine 3).

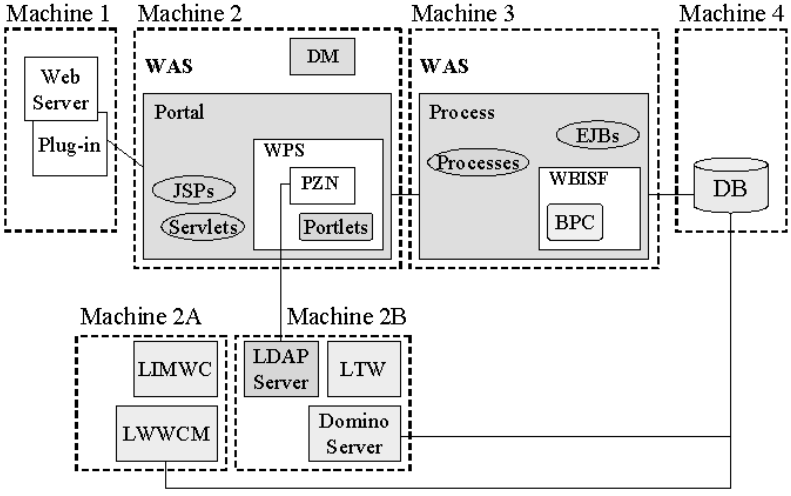

Figure 2.16: A process portal configuration that maintains separation of layers.

Another subtle change appears in Figure 2.16, wherein we replaced Domino LDAP with a generic LDAP Server. That was to indicate the fact that Domino Server can work with other LDAP Servers like IBM Tivoli Directory Server (TDS).

Separating the middle tier in such a way would also help in product upgrades. It is conceivable that the business processes are meant only for employees—that is, the process portal might be surfaced in an intranet portal, whereas the portal machine could be serving up content meant for both, intranet and Internet portals. As enterprises become more knowledgeable and familiar with process portals, we are quite sure that architects will find better ways to design them.

Door Closings

No matter which topology you choose, avoid last-minute surprises by configuring your test or staging system to look exactly like your proposed production system. For example, even if everything works fine on a single-node configuration, when some of the software components are configured to run on different machines, you have to make sure things are deployed properly and that inter-machine communications are not a bottleneck.

With the newest kind of portal, Process Portals, a few more components are introduced into the configuration, but they still follow the MVC paradigm, and the topologies do not drastically change.