WebSphere Application Server: A foundation for on demand computing

by E. N. Herness
R. H. High, Jr.
J. R. McGee

WebSphere® Application Server is the foundation for IBM's middleware software portfolio. It has evolved rapidly from a simple extension for Web servers and a server runtime for business objects to the IBM distributed operating system for mission-critical computing and the leading application server in the industry. WebSphere Application Server plays a central role in the transformation from a distributed operating system to a distributed on demand operating system. This transformation is achieved by forging extensions to the WebSphere Application Server foundation for the grid-computing infrastructure, rich Web-based interaction models, service-oriented architecture, autonomics, business process management, and dynamic provisioning and utility management. This paper describes elements of the WebSphere Application Server architecture and how this architecture provides a foundation for the on demand computing infrastructure and application environment.

There is little doubt about the importance of e-business on demand*.[1] The information technology industry has come to a juncture with the next major step being one toward greater productivity— in terms of improving both the productivity of information systems specialists and, more importantly, the productivity of businesses that depend on information technology for conducting their business. This next step forward will be enabled by e-business on demand* through more efficient utilization of computing facilities by sharing resources among many lines of business, and by interconnecting those facilities into a computing grid that will enable access to more computing capacity on demand. This potential is then extended through e-business on demand by allowing lines of business to be interconnected as a seamless flow of information and business processing and by providing concrete definitions of customer business processes so that customers can adjust those processes rapidly as business conditions change.

The role of WebSphere* Application Server[2–6] in enabling on demand computing is significant in two ways. First, it is a container for application components whose very programming model design enables a high degree of virtualization. This is achieved in the underlying information system by separating the presentation and business logic of the application from the infrastructure hosting that logic.[7] Second, the application server is a resource manager—it manages application components as resources, and manages them in the context of the computing and information resources they depend on, including the execution environment, data systems, connections, transactions, security contexts, RAS (reliability, availability, and serviceability), messaging systems, and other application components. Both of these properties form a critical backdrop to enabling on demand computing.

WebSphere Application Server supports four major models of application design: multitiered distributed business computing, Web-based computing, integrated enterprise computing, and service-oriented computing. All of these design models focus on separating the application logic from the underlying infrastructure; that is, the physical topology and explicit access to the information system is distinct from the programming model for the application. Use of underlying resources within the information system is abstracted in the programming model by high-level interfaces and logical resource references and by encouraging service processing through declarative policies in the components. The appearance of control is given but in a way that can be mapped to physical resources by the application containers in WebSphere Application Server based on its management algorithms. Exploiting the component models defined in the WebSphere Application Server programming model makes programmers more productive, but it also enables the application to be managed by WebSphere Application Server. The application components can be located within the topology as needed on the basis of the resources required by the application, as measured by the availability and capacity of the underlying computing facilities, and on the basis of the relative requirements of the application as compared to other applications in use by the enterprise.

WebSphere Application Server provides support for deploying the application, managing the resource requirements for the application, ensuring the availability, isolation, and protection of the application from other applications and their resource requirements, and monitoring and securing the application. In the following sections we will survey various aspects of WebSphere Application Server and how it enables computing for e-business on demand. Included is a discussion of how applications are managed and deployed, how the application server is monitored to determine how efficiently it is using resources and how this affects workload management, how applications can be profiled to enable the application server to serve its resource dependencies more efficiently, the infrastructure technology the application server uses to ensure high availability and failover, how this capability extends out to the edge of the network, and finally the role the application server plays in the area of grid computing.

Other aspects of WebSphere Application Server and much of the WebSphere platform, including approaches to obtaining maximum scalability from WebSphere Application Server,[8] how to improve performance by caching,[9] information integration,[10] portals,[11] business process choreography,[12] disconnected and rich clients,[13] and connectors and adapters[14] are discussed at length throughout this issue of the *IBM Systems Journal*.

Application models for e-business

One major characteristic of an on demand e-business is that it is dynamic. It changes at the rate and pace of demand—demand for business services, demand for information, and demand for computing capacity. An application will survive in this sort of environment only if it is designed for change. There are many application design patterns,[15] including the principles of structured and object-oriented programming, that describe techniques for achieving high degrees of reuse and component sharing.

However, going from reuse to the kinds of dynamic rehosting that can occur as resources are shifted across a data center, or to handle the distribution of workload across application partitions, or to handle rapid changes in business processes that can occur in an on demand computing environment requires strict adherence to the best design disciplines. In an on demand computing environment, databases and legacy data systems in the Enterprise Information System (EIS) can be moved frequently to accommodate surges or drops in demand for one system or another. It may become necessary to partition workloads and route different transactions to different computers. There may be many instances of the same application in the same computer complex to serve different customers. The steps and activities that are performed for a given business situation can vary from day to day, or even for different business customers on an individual basis. Under other circumstances it would be very difficult to program an application to tolerate this kind of variability.

Letting programmers add value to their businesses by enabling them to focus their attention more on business domain concerns and less on the underlying computing infrastructure is one of the fundamental tenets of WebSphere Application Server. In addition WebSphere Application Server is a J2EE** (Java 2 Platform, Enterprise Edition)-compliant application server supporting the entire breadth of the Java** language and J2EE specification, including support for a Web-services-based service-oriented architecture. The J2EE programming model supported by WebSphere Application Server makes it much easier to build applications for on demand computing specifically because it separates the details from the underlying infrastructure. It does this with the component- and service-oriented programming model offered by J2EE. By leveraging that separation, WebSphere Application Server is able to offer a broad range of scaling in its application server implementation—from a single server installation all the way up to multiprocessor, multihost, and multicluster installations.

To obtain maximum advantage as an on demand business an application should follow the best-practice patterns that have been espoused for e-business computing[16]—using the J2EE and WebSphere Application Server programming model for component-based, service-oriented, distributed, and message-driven computing, and using business process choreography for composition. The patterns for e-business computing with this approach include:

- Multitiered distributed business computing
- Web-based computing
- Integrated enterprise computing
- Service-oriented computing

Multitiered distributed business computing. The value of multitiered distributed computing comes from first structuring the application with a clean separation between the logic elements (presentation, business, and data) and then leveraging the boundaries between these elements as potential distribution points in the application. This is enabled with a formal component model for business logic: Enterprise JavaBeans** (EJB**). This component model has several key benefits for application development. Foremost, the component model provides a contract between the business logic and the underlying runtime. The runtime is able to manage the component.

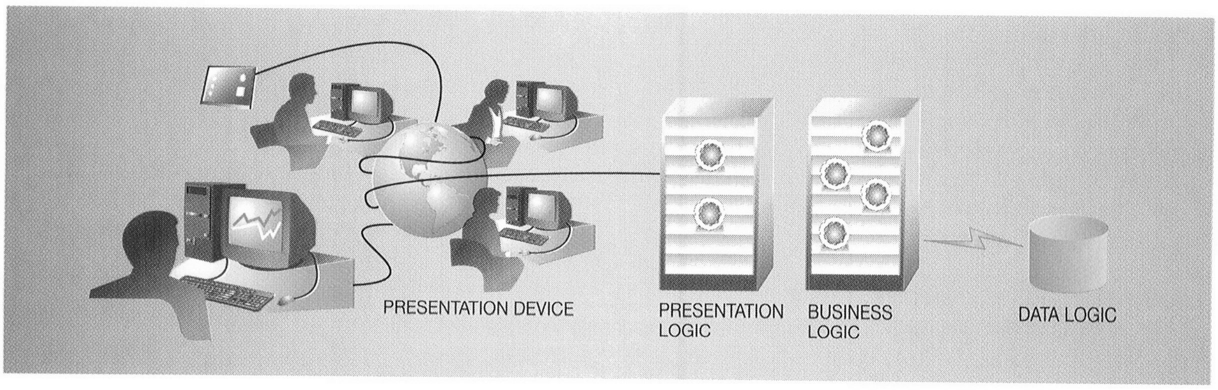
The component runtime (also called the component container) ensures the optimal organization of component instances in memory, controlling the life cycle and caching of state to achieve the highest levels of efficiency and integrity, protecting access to the components, and handling the complexities of communication, distribution, and addressing. This same principle of management applies to object identity, transaction and session management, security, versioning, clustering, workload balancing and failover, and so on. Part of the EJB component model includes the idea of a single-level-store programming model, whereby the issues of when and how to load the persistent state of a component are removed from the client.[17] In many cases, the runtime has a much better understanding of what is going on in the shared system than any one application can ever have and thus can do a better job of managing the component and obtaining high performance and throughput in the information system. The runtime does this for the component developer; the programmer does not have to worry about these details.

Web-based computing. Web-based computing is, in some sense, a natural extension of the multitiered distributed computing model, whereby the presentation logic has been relocated in the middle tier of the distributed computing topology and drives the interaction with the end user through fixed-function devices in the user's presence. We refer to an in-presence, fixed-function device as a Tier-0 device in the multitiered structure of the application. The most common form of a Tier-0 device is the Web browser on a client desktop. Pervasive computing devices are emerging in the market in other forms as well, from personal data assistants (PDAs) and mobile phones, to intelligent refrigerators and cars (see Figure 1).

Web applications exploit the natural benefits of component-based programming to enable the construction of Web presentation logic that can be hosted on a server platform and to achieve richer styles of interaction than can be achieved with simple static content servers.[18] The Web application server was originally conceived to extend traditional Web servers with dynamic content; that is, page content that is derived dynamically by interacting with business logic and back-end data systems. However, in the course of developing these Web application servers, we realized that the issues of serving presentation logic are essentially the same as the issues of serving business logic. Through the use of servlets,[19] portlets,[20] and JavaServer Pages** (JSPs**),[21] we see this model supporting both a presentation and a business logic tier in the application server layer. As with business logic, the purpose of the presentation logic server is to accommodate many clients (in this case, Tier-0 clients) sharing a common implementation.

Integrated enterprise computing. Integrated enterprise computing is critical to retaining customer investments in past technologies. Few new applications can be introduced into an established enterprise without some regard as to how they will fit with existing applications[22] and, by extension, the technology and platform assumptions on which those ap-

Figure 1 Web computing model



plications have been built. A close examination of any enterprise will reveal a variety of applications built on a variety of underlying technology assumptions.

As usual, the issues of cross-technology integration are complex. In mission-critical environments, we must address concerns about data integrity, security, traceability, configuration, and a host of other administrative issues for deployment and management. However, to support the productivity requirements of developers, these complexities should be hidden. The key programming model elements provided by WebSphere Application Server for enterprise integration are offered in the form of Java 2 Connector Architecture (J2CA)[23] and the Java Messaging Service (JMS),[24] both of which are part of the J2EE specification. Figure 2 depicts this integration.

Another major advance in application integration can be realized with the use of business process choreography. Choreography is accomplished through a scripting language (the Business Process Execution Language for Web Services, or BPEL4WS[25]) that describes how to sequence a number of service activities to form a workflow definition as depicted in Figure 3. Activities can be serialized or executed in parallel. A business process instance is instantiated and forms its own state as it executes; that state can be passed between activities in the flow. A business process can be started, interrupted, resumed, and terminated.

In looking at the problem of application integration (or perhaps, more appropriately, lines-of-business integration) from the top, it is important to model business process flows in a way that allows them to be rapidly adapted to new procedures and opportunities; for example, to be able to model and then rapidly modify the order entry process to perform credit checks and to process partial orders without having to rewrite either the order entry or inventory systems.[26] Choreography lets us script the business process, which makes it very easy to modify and immediately execute changes in the implementation of the process definition. Business rules, when combined with choreography, can allow even more flexible processes to be defined.

Service-oriented computing. The service-oriented architecture (SOA) model suggests a type of application where "business services" are exposed for use both within and outside of an organization.[27] Service-oriented architectures leverage the relative cohesiveness of a given business service as the primary point of interchange between parties in the network.[28] Services can be associated with service-level policies so they can be secured, metered, monitored, and tracked. We can combine the concepts of Web services with J2EE to provide a managed component-hosting environment for these kinds of applications.

Advances in the field of service-oriented computing are heavily focused on Web services—especially on Web Services Definition Language (WSDL), on the Simple Object Access Protocol (SOAP), and Universal Description, Discovery and Integration (UDDI). These technologies combine to introduce business services that can be easily composed with other business services to form new business applications. At some level, Web services are just a new generation of technology for achieving distributed computing,

Figure 2 Integrated enterprise computing

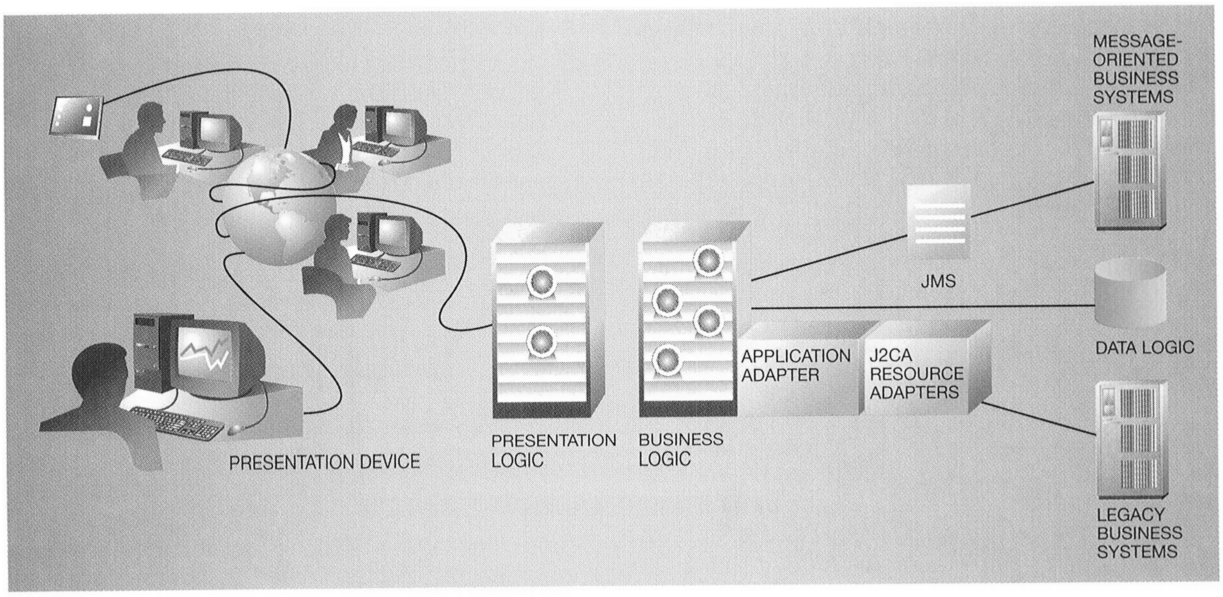
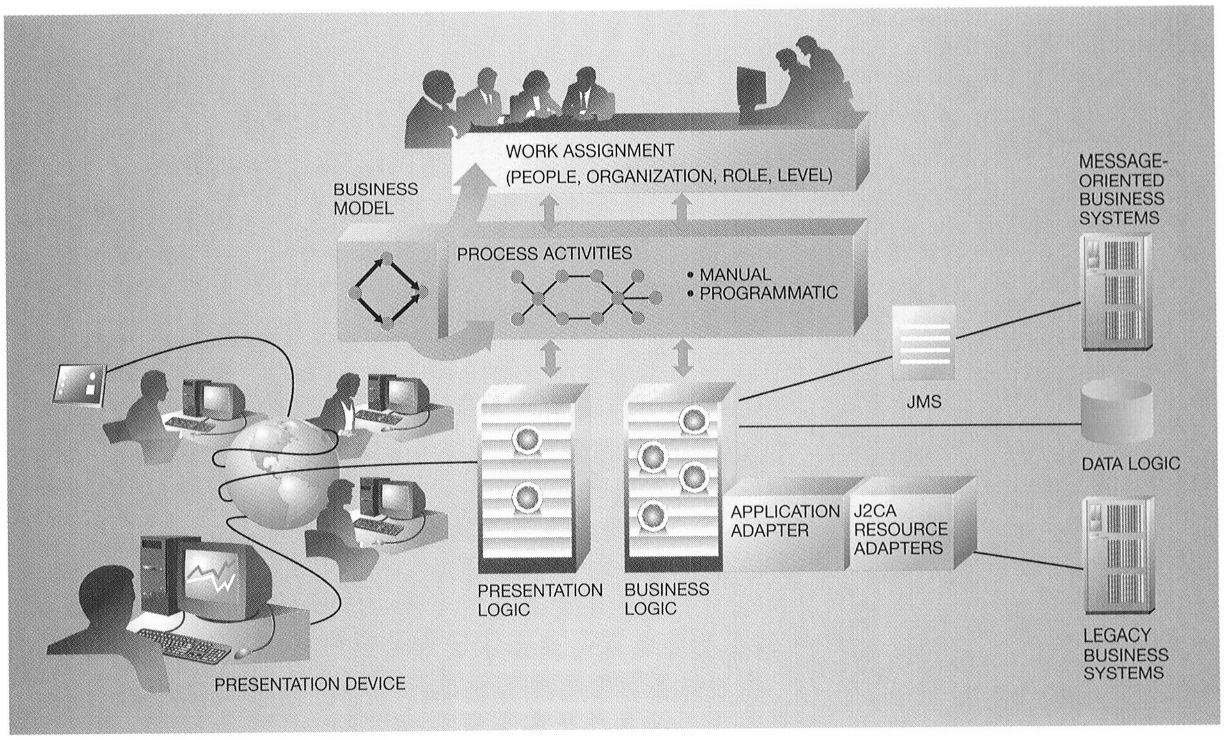


Figure 3 Business process choreography



and in that sense, they have much in common with many of the other distributed computing technologies, such as Open Software Foundation/Distributed Computing Environment (OSF/DCE**), Common Object Request Broker Architecture (CORBA**), and the J2EE Remote Method Invocation/Internet Inter-Orb Protocol (RMI/IIOP**), that went before them.[29] However, Web services differ from their predecessors in the degree to which they deliberately address a "loose coupling" model.[30]

We can measure the coupling of distributed application components in at least three ways:

- Temporal affinity
- Organizational affinity
- Technology affinity

Temporal affinity. Temporal affinity is a measure of how the information system is affected by time constraints in the relationship between its components. If an application holds a lock on data for the duration of a request to another business service, there are expectations that the requested operation will be completed in a certain amount of time—data locks and other similar semaphores tend to prevent other work from executing concurrently. Tightly coupled systems tend to have a low tolerance for latency. In contrast, loosely coupled systems are designed to avoid temporal constraints—the application and the underlying runtime are able to execute correctly and without creating unreasonable contention for resources even if the service requests take a long time to be completed.

Organizational affinity. Organizational affinity pertains to how changes in one part of the system affect other parts of the system. A classic example is in the versioning of interdependent components. If the interface of a component changes, it cannot be used until the dependent components are changed to use that new interface. In tightly coupled systems, the change has to be coordinated between the organization introducing the new interface and the organization responsible for using that interface. The coordination often requires direct and detailed communication between the organizations. In contrast, there is a high degree of tolerance for mismatches between components in loosely coupled systems.

Another dimension of organizational affinity is the degree to which the system has to be managed by a single set of administrative policies. Tightly coupled systems tend to require a common set of administrative policies, most commonly handled with a centralized administration facility to ensure consistency of policy. The administration of loosely coupled systems tends to be highly federated, allowing each party to apply its own administrative policies and expressing the effects of those policies only as "qualities of service" at the boundaries between the organizations. Generally, the invoker of a service in a loosely coupled system can make choices based on the trade-offs of costs, benefits, and risks in using an available service. Different providers with different quality-of-service characteristics can supply the same service, thus enabling commercial marketplace economics to drive a services community.

Technology affinity. Technology affinity addresses the degree to which both parties have to agree to a common technology base to enable integration between them. Tightly coupled systems have a higher dependence on a broad technology stack. Conversely, loosely coupled systems make relatively few assumptions about the underlying technology needed to enable integration.

All of these forms of affinity represent potential barriers to the dynamism that occurs in on demand computing systems. Avoiding them by adopting the principles of loose coupling promoted by SOA is critical in overcoming these barriers.

Programming for e-business on demand. There is often a temptation to take short cuts when building applications to respond rapidly to an immediate opportunity or to disregard the more powerful elements of the J2EE programming model. In particular, we see many cases where servlet programmers directly invoke system services or make direct database calls. That method may save developers some time initially, especially if programmers are familiar with these services and have intimate knowledge of their database. However, doing so also weds their application to that specific set of service technologies and database schema. These applications will be more brittle and will not be able to fully benefit from the capabilities that WebSphere Application Server introduces for on demand computing. To benefit from on demand computing, the application must be structured to avoid directly connecting to the resources it uses.

Conversely, we recommend that application abstraction layers which attempt to hide the J2EE and Web services programming model be very carefully designed and generally avoided. Although better abstractions might offer some simplification to programmers, they also tend to inhibit the application from taking advantage of some of the underlying flexibility that is being built into the application server. Specifically, it can be difficult to leverage quality-of-service semantics that enable WebSphere Application Server to manage components optimally when those underlying component models are obscured by higher-level abstraction layers. Often, we see framework designers having to duplicate the very same quality-of-service declarations and container management that are inherent in the application server itself.

Caches are another good example. A number of new capabilities related to workload management and virtualization will be driven by e-business on demand. Applications that build local caches and alter normal affinity rules will not be able to efficiently participate in environments where work is dynamically moved around a distributed system based on load statistics. WebSphere Application Server has built-in cache support and leverages and manages those caches to ensure optimal throughput and resource utilization while maintaining a very high level of data integrity. These properties are difficult to duplicate in the application layers. Application and framework developers often "get it wrong." Moreover, even when they "get it right," they may be contravening the inherent capabilities of the application server.

When WebSphere Application Server knows about all of the resources being leveraged, such as connectors, databases, queues, JMS topics, memory, and disk storage, the inherent virtualization capabilities in the server can more readily balance workloads across those available resources, move work around the system, and provision additional resources on machines in the network. If the server does not know about the resources, it is not possible to provision new capacity without operator, administrator, or sometimes even programmer intervention.

Finally, the description of applications and the application architecture has been centered largely around the current J2EE standards and Web services standards. As more advanced applications are constructed to solve business problems, they will make increasing use of additional WebSphere programming model extensions[31] and other platform capabilities.

Developing and deploying an application to WebSphere Application Server

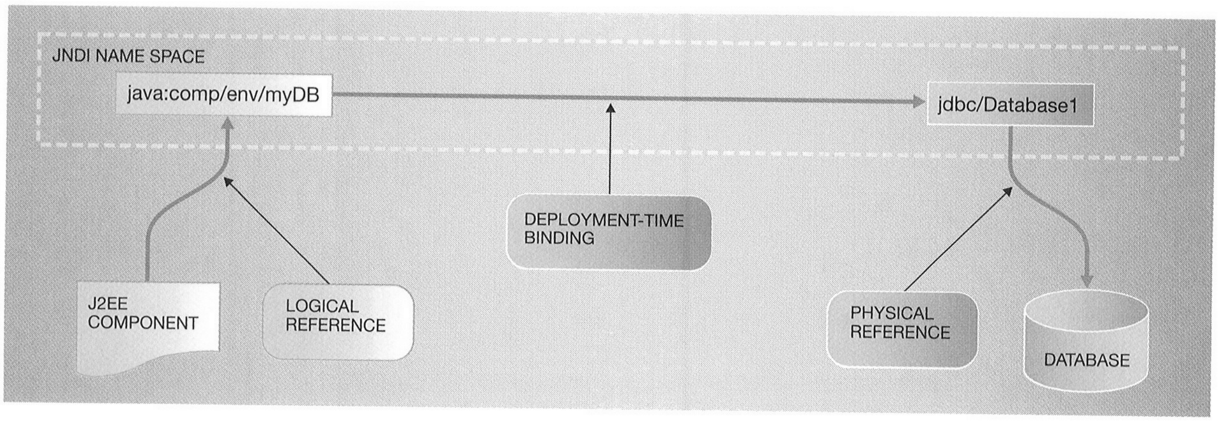
The principles of virtualization can be defeated if the processes for constructing a virtualized application component are too difficult. In fact, we want to make the process for creating a virtualized application easier than creating an application that makes hard-coded assumptions about the resources and topology in which it will execute. The goal of WebSphere Application Server is to simplify the development and deployment process to enable more rapid application construction and improve the efficiency of the development and debugging process, specifically to encourage the construction of applications that are enabled for on demand computing.

We begin by recapitulating the development and deployment process for component-based applications. The development process is comprised of creating two primary types of artifacts: *logic elements* and *declarative meta-data*. Logic elements are the code portion of the application, captured in components such as servlets, EJBs, Java classes, and the like. Declarative meta-data is the information usually provided in the form of an eXtensible Markup Language (XML) document that controls the deployment and execution behavior of the application. By specifying information about how the logic elements should be executed and bound to the runtime environment outside of the code itself, an application gains greater flexibility and adaptability to new problems and new environments.

One of the most important forms of declarative meta-data introduced by J2EE is the notion of resource or component references. In J2EE, references allow a level of indirection in the name of some resource that a logic element needs to use.[32] For example, if a servlet needs access to a data source, the servlet can refer to that data source using a logical name of its choosing. Later, when the application containing the servlet is installed onto the application server, the administrator is able to choose the physical database that will be accessed by that servlet (see Figure 4) and map that database to the logical name selected by the programmer within his or her code. At any time, the binding between the logical and physical data source names can be modified by the administrator without any change to the logic of the servlet.

By providing indirection, the application logic is prevented from being too tightly bound to a particular

Figure 4 J2EE reference binding



deployment environment, and the developer of that application does not have to manually build an indirection mechanism. J2EE provides this indirection mechanism for database connections, Web services, J2CA resources, uniform resource locators (URLs), EJBs, JMS topics and queues, and generically for any object through the Java Naming and Directory Interface (JNDI) service.

Other forms of declarative meta-data include caching policy, transaction policy, security policy (policy to be applied both to users of the application and to the application logic itself), performance settings, persistent field information, and application profile data (indicating how the application is expected to be used by particular clients). Declarative meta-data is also used to inform the system of the elements of the application, such as which EJBs are present, which servlets are present, which URL should be used as the default page for an application, and which modules make up the application. Because this information is provided outside of the application logic, the behavior of the application can more easily be modified, and the logic elements can be reused in different applications with different functional and nonfunctional requirements.

Deployment of an application is the process of installing an application into the application server and making that application available for execution. Deployment is a multistep process, including:

1. Presenting the application for deployment and inspection so that its contents can be understood by the runtime.
2. Generating the additional logic elements required to execute the components of the application in the system. This process uses the declarative meta-data to decide which behaviors must be applied to the component. For example, the meta-data for container-managed entity EJBs indicates which fields should be persistent, and the code generation process produces the actual logic to load and store the element in the database.
3. Binding the application to the environment after code generation. Binding is the process of resolving any logical references in the application to physical resources available in the environment (for example, picking the exact database that should back a particular data source).
4. Deciding on which servers or clusters the different components of the application should execute, which is also done by binding. See "Management and Provisioning" for more information about WebSphere clustering topology.
5. Specifying configuration and tuning parameters for the application that are specific to this installation of the application (such as bean pool sizes) by an application deployer in addition to the binding information. This additional deployment information allows the server to be tuned for the particular application in a per-application manner.
6. Distributing the application to all of the machines that will host the application and then starting the application (the final step in deployment). WebSphere Application Server does this automatically from the deployment manager through a configuration synchronization protocol used between the deployment manager and node agents. The administrator can configure the application serv-

ers and nodes in their cell (a collection of machines enabled for WebSphere Application Server) through the administrative console attached to the central deployment manager. The relevant configuration information, including the EAR (Enterprise Application Archive) and JAR (Java archive) files for the application itself are then synchronized with each node in the cell.

At this point, the application is running and available for service. It is important to note that many of the decisions made during deployment can be modified at any time to adjust the system. For example, bindings can be changed, and application modules can be moved to different servers or clusters. (Clusters are discussed in the section "Workload Management.") Most of these changes can be made without restarting the server by simply restarting the application that was affected.

The deployment flexibility in both J2EE and Web services provides the foundation for on demand enablement of applications. In on demand computing environments, the system will make decisions about the optimal configuration to meet the goals defined by the administrator. Optimization includes both adjusting tuning parameters for a given application or server and adjusting the assignment of applications to servers and resources available in the cell. By using the indirection mechanisms described above, the on demand system can adjust numerous parameters of the application, such as the data source settings for a particular database, the server or cluster on which to execute the application, and the tuning parameters for object pools and caches, without the application itself being aware of the changes, and without requiring the application to make any modifications.

Let us discuss a particular example. In an on demand system, the demand manager monitors the current loading on the system. On the basis of the performance data that are being collected, the system is able to detect that a particular server (say, server A) is underutilized. At the same time it might detect that server B is overloaded, handling the requests of multiple applications. On the basis of the described deployment flexibility, the system could change the server assignment binding of a particular application, moving the application from server B to server A, to better balance the load. When this happens, the system will automatically move the binaries for the application to the new machine, remove them from the old machine, start the application on the new server, and stop it on the old server. All of these changes are transparent to the application logic itself and to the users of the application.

WebSphere rapid deployment. The deployment and configuration capabilities of WebSphere Application Server and J2EE are powerful, but the addition of new concepts and artifacts to the development and deployment process increase the complexity of application construction for individuals who are not familiar with J2EE. Therefore, some may be dissuaded from using the full power of J2EE if they do not understand how to bring all the pieces together. To help facilitate the process, WebSphere Application Server Version 6.0 will add a number of features to simplify the development and deployment experience for developers—enabling the "pay-as-you-go" notion of on demand computing. Collectively, these features are referred to as WebSphere Rapid Deployment (WRD). WRD introduces two key concepts: *annotation-based programming* and *deployment automation*.

Annotation-based programming. Annotation-based programming is the notion of adding meta-data directly to the source code of a program and using that meta-data to generate the additional artifacts of the application. This approach simplifies development by greatly reducing the number of artifacts that must be created and maintained directly by the developer and by reducing the amount of redundant information between multiple artifacts.

As an example, consider the remote interface class of an EJB. This class contains the method signatures of the methods that should be exposed to remote users of the EJB. The method signatures in the remote interface are essentially the same as the method signatures in the EJB implementation class. If the developer changes the method signature of some method in the implementation class of the EJB, the same redundant change has to be made to the remote interface class. This extra step adds complexity and the opportunity to make mistakes in the development process.

With annotation-based programming, the developer instead adds special tags to the EJB implementation class to mark which methods should be made available on the remote interface. The remote interface can then, as part of the deployment process, be automatically generated from the implementation source code. This generation removes the redundancy, reduces the opportunities for errors, and reduces the number of artifacts with which the developer must deal. This is the essence of annotation-based programming. Currently, annotations will be added to Java source code through Javadoc**-style tags in comments within the code. For the above example, the source code of a remote method would look something like the following:

```
/**
* @ejb.interface-method view-type=
  remote
*/
public String hello(String name)
{
   return "Hello: " + name;
}
```

With annotation-based programming, external files would take precedence over the annotations contained in the source code if the developer also provides some of the declarative meta-data through external XML files. As can be seen in this example, if the developer changes the signature of the hello() method, he or she will not have to change any other artifact in the application. The remote interface automatically is updated to reflect the change. Thus, the development process has become simpler to understand and less error prone, and the developer has had to acquire less knowledge about J2EE and the full complexity of artifacts involved. This simplification is an important aspect of on demand computing.

Deployment automation. Deployment automation is the notion of automatically handling the deployment process for the user, including code generation, compilation, and installation tasks. The key feature of deployment automation is the notion of an actively monitored directory where any changes made to the contents of the directory are detected by the system and appropriate actions are taken on behalf of the user to reflect the changes made. This capability enables a simple drag-and-drop or edit-in-place deployment model, greatly simplifying the deployment process. As an example, a developer can install a new J2EE EAR file onto the server simply by copying the EAR file into a monitored directory, and then the rest of the deployment process will proceed automatically.

Of course, to automate some of these processes requires the system to choose defaults for certain settings that normally would be provided by the user. With use of the knowledge acquired for on demand management of applications, defaults can be chosen that are appropriate for most applications. For many applications, deployment follows a simple pattern, and there is no need to ask the user to provide deployment data. For other applications, complex analysis could be performed to infer the correct values. The deployment automation is smart enough to do the most efficient action possible in the face of the changes detected in the monitored directory, thereby improving developer efficiency by reducing the edit-compile-debug cycle time, a key goal of the simplification aspects of on demand computing.

The combination of deployment automation and annotation-based programming enables a powerful model of on demand development. In this model, the developer can edit a small number of logic artifacts (artifacts that are predominantly business logic artifacts) and have a fully compliant J2EE application constantly being constructed and deployed in the background, making his or her latest changes continuously available for debugging without any additional manual steps.

Management and provisioning

The WebSphere Application Server management and provisioning system is one of the most important components in supporting the paradigm of on demand computing. The management system provides the fundamental capabilities to understand and control the information technology (IT) system. The role of the management system is to provide information about the current configuration of the system and to provide a mechanism to change that configuration. In on demand computing, however, there are additional important requirements. First, the management of the system must be accessible in a standard way to enable multiple systems to be controlled together. Second, the system must be capable of being changed dynamically and of adjusting to those changes transparently. Capabilities are being added that leverage the standard interfaces and configuration flexibility that already exist in WebSphere Application Server to enable it to participate further in the on demand provisioning system.

After the application itself has been developed and deployed, the ongoing cost and success of an application in production depends largely on the ease and ability of managing that application in the production environment. The WebSphere Application Server management system provides a unified management view, called a *single-system image*, across a multiprocess, multimachine, heterogeneous deployment environment.[33] It allows the system to be configured, modified, monitored, and controlled through a single point of administration from multiple administrative agents, simultaneously and dynamically.

The core of the WebSphere Application Server management system is a data model that represents the configuration of the cell. Currently, the data model is embodied in a collection of XML documents on the file system of the various machines in the cell. A central management process called the *deployment manager* (dmgr) is provided, which maintains the master configuration of the cell and provides a single point of contact for controlling the environment. Each machine that is controlled by the dmgr contains an agent process called the *node agent*. The dmgr and the node agents collaborate to allow control and monitoring of the system and to perform a configuration synchronization function. Each machine in the environment contains a copy of a subset of the master configuration of the cell appropriate for that machine, and the dmgr and node agents communicate to keep that copy synchronized with the master copy.

For control of the system, WebSphere Application Server exposes all of its management functions via standard Java Management Extensions (JMX**) MBeans.[34] JMX is the Java standard application programming interface (API) for managing a system. The core concept in JMX is the notion of a management bean—an MBean. An MBean is a component that allows access to the management data and operations of some portion of the system. MBeans expose operations, attributes, statistics, and notifications. JMX capabilities allow the server and the cell to be controlled and monitored.

WebSphere Application Server uses the JMX interface internally to implement the management functions of the product, but the interfaces are also exposed externally for use by other management agents, such as the IBM Tivoli* product. Accessing these MBeans over multiple protocols is also supported, currently by SOAP/HTTP(S) (Simple Object Access Protocol/HyperText Transfer Protocol [with or without a secure connection]) and RMI/IIOP (WebSphere Application Server Version 6.0 adds support for SOAP/JMS [Java Message Service]). The protocol choices allow the user to select the quality of service most appropriate for his or her environment. With use of standard management interfaces WebSphere Application Server is enabled to be controlled as part of a larger systems management environment, allowing better control of an entire production IT environment. JMX accessed by means of Web services provides the standard control mechanism required by on demand computing.

For access to the WebSphere management system, three primary clients are provided. For graphical management, WebSphere provides a browser-based interface to the entire management environment. An example of the WebSphere administrative console is depicted in Figure 5. This Web user interface (UI) is implemented as a standard J2EE Web application, using servlets, JSPs, and Struts (an open source framework used in building Java Web applications). It is packaged as an EAR file and installed just as other applications are on the application server.

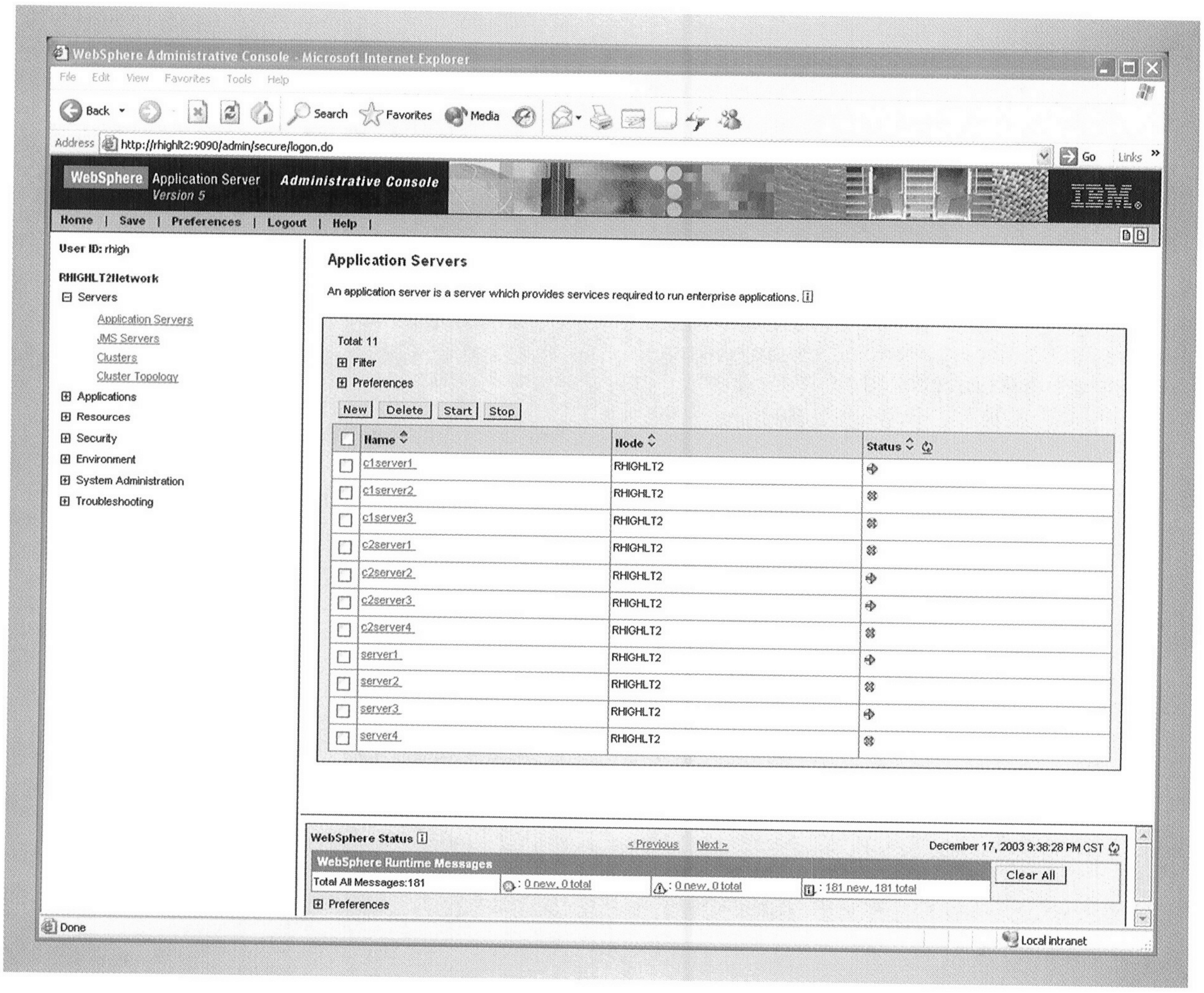
For command-line or scripted access, WebSphere Application Server provides a tool called WsAdmin. WsAdmin has both an interactive shell and the ability to execute scripts. WsAdmin supports the creation of administrative scripts in multiple scripting languages (currently JACL[35] [Java Command Language] and JPython**[36]), allowing customers to choose the scripting language with which they are familiar.

The final client is a programmatic API for accessing the management system. The API client allows custom management applications to be constructed that can monitor and control WebSphere Application Server. The administration API client is perhaps the most important client for on demand computing, allowing external agents to be written that can monitor and control the WebSphere platform.

The WebSphere management system is designed to be very scalable, allowing users to manage a large installation of WebSphere Application Server servers. The system is designed to efficiently distribute information and to minimize the amount of cross-talk and interdependencies between processes. The Web UI is designed to allow viewing and editing of large topologies, providing features such as view sorting, search, and filtering. This scalability is important in allowing a production IT environment to be managed centrally and in allowing partitioning of an environment to be driven more by customer business reasons than product technical constraints. The Web UI is also designed to be extensible, a key capability in on demand computing.

As more of the system is managed automatically, an administrator's role becomes less to manage individual products and more to manage the IT system

Figure 5 WebSphere Application Server Version 5.0 administrative console


WebSphere Administrative Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://rhighlt2:9090/admin/secure/logon.do
Go Links
WebSphere Application Server Version 5 Administrative Console
IBM
Home | Save | Preferences | Logout | Help |
User ID: rhigh
RHIGHLT2Network
Servers
Application Servers
JMS Servers
Clusters
Cluster Topology
Applications
Resources
Security
Environment
System Administration
Troubleshooting
Done
Local intranet

Application Servers

An application server is a server which provides services required to run enterprise applications.

Total: 11

⊞ Filter

⊞ Preferences

New | Delete | Start | Stop

	Name	Node	Status
☐	c1server1	RHIGHLT2	➔
☐	c1server2	RHIGHLT2	✻
☐	c1server3	RHIGHLT2	✻
☐	c2server1	RHIGHLT2	✻
☐	c2server2	RHIGHLT2	➔
☐	c2server3	RHIGHLT2	➔
☐	c2server4	RHIGHLT2	✻
☐	server1	RHIGHLT2	➔
☐	server2	RHIGHLT2	✻
☐	server3	RHIGHLT2	➔
☐	server4	RHIGHLT2	✻

WebSphere Status < Previous Next > December 17, 2003 9:38:28 PM CST

WebSphere Runtime Messages — Clear All

Total All Messages:181 | : 0 new, 0 total | : 0 new, 0 total | : 181 new, 181 total

⊞ Preferences

as a whole. To make this change practical, the administrator needs a common administrative UI that crosses product boundaries and that is tailored to the scenarios that the administrator encounters on a daily basis. The Web UI in WebSphere Application Server Version 5.0 provides part of the foundation for a unified administration model that will be available in the Version 6.0 timeframe, enabling a common and consistent approach to middleware management across the system.

As WebSphere Application Server moves toward on demand enablement, the key aspect of its management system is its support for dynamic adjustment of the configuration of the system. In WebSphere Application Server Version 5.0, the management emphasis was on dynamic workload distribution over a relatively static configuration. In Version 6.0 and in on demand computing, the emphasis shifts to encompass dynamic configuration changes based on dynamic feedback from a policy-driven management agent. Although this shift encompasses some significant new functions, it is built and predicated on what exists today in WebSphere Application Server Version 5.0.

On demand provisioning managers such as the IBM Tivoli Provisioning Manager and IBM Tivoli Intelligent ThinkDynamic* Orchestrator[37] can use JMX facilities to acquire the information they need to make decisions about how to dynamically control the platform. (This monitoring capability is discussed in the following section in more detail.) After an on demand change is decided upon, the same JMX facilities are used to provision that change on the running server. WebSphere Application Server will then adjust to the new environment as the change occurs. For example, if a new server is added to a cluster to increase the performance of some application, WebSphere Application Server will notify the workload management system of the change in the cluster configuration, allowing the new server to absorb its share of the workload.

By exposing JMX and the administration client APIs, WebSphere Application Server enables external control over the configuration of the system in response to utilization and capacity policy assessment. The provisioning manager assesses the utilization of the system against the policies that determine whether a change is needed. If warranted, it can initiate a configuration change immediately, reducing the time for making these sorts of changes down to minutes or hours as opposed to the days or weeks that it might take an administrator to understand and respond to a shift in demand under manual circumstances.

Monitoring and application response measurement

In order to manage an on demand environment dynamically, the on demand system must have accurate and consistent data about the current state and condition of the IT environment. It is not possible to balance an arriving workload across a computing grid without an idea of the current loading of servers within that grid. Therefore, the ability to monitor a collection of systems and to understand their current operation is the critical first requirement of on demand computing. This monitoring environment must be accurate, must provide continuous access to information in the system, and must be lightweight enough to not negatively impact the performance of the systems being monitored.

WebSphere Application Server provides monitoring capabilities as part of its management system, such as monitoring of the current life-cycle status for servers, applications, and their components (e.g., which servers are running), monitoring of problems that are occurring in the cell (e.g., alerts, errors, warnings), and monitoring of performance (both aggregate and per request). As we move forward to Version 6.0, a single infrastructure for capturing IT and business-level events will enable higher-level monitoring to occur as well as correlation across both IT and business-level events.

Java management extensions. Status monitoring is provided through the JMX MBeans that represent the components of the system. The status of any particular component, such as a server or an application, is available either as an attribute of the MBean or indicated by the presence or absence of the MBean itself (the MBean is only registered when the component is active). Status information can be polled by querying the appropriate MBean, or a client can register for notification when the status of a component changes state. Status monitoring is critical for understanding the current state of the environment and is used to control critical functions such as workload management. The ability to understand status is also critically important in enabling WebSphere Application Server for on demand computing, allowing external agents to understand which resources are available and make decisions on what to do next.

Monitoring of problems is provided primarily through two mechanisms. The first, and most obvious, uses log files. Each server in the cell maintains a log of events and errors occurring in the server. An activity log of critical events happening in all servers on a particular machine is also maintained. The contents of both of these logs are available on the local machine and remotely via JMX. These logs allow a detailed understanding to be gained of problems that are occurring in the system.

Additionally, critical events that occur in the system are also exposed via JMX notifications, which proactively notify administrators of problems as they occur. Furthermore, external agents can register for notification when critical problems occur. This notification capability is important in enabling on demand management of the WebSphere Application Server environment by allowing the system to proactively respond to problems as they occur. The notification also simplifies production management of a WebSphere Application Server system by removing from the administrator the burden of having to constantly search log files for problems.

Performance monitoring allows a user to understand the performance characteristics of the running application in terms of the components that comprise the application and the systems the application is using. Performance monitoring also provides the basic data necessary to determine how to provision and rebalance the on demand system. It provides real load information from the servers to the workload manager to help it decide how to distribute workload among a cluster of WebSphere Application Server servers. There are two types of performance information: aggregate performance metrics and per-request timing data.

Performance Monitoring Infrastructure. Aggregate performance data are provided through a system called the Performance Monitoring Infrastructure (PMI).[38] PMI is comprised of a set of configurable counters at critical points in the WebSphere Application Server runtime that keep track of statistics for all requests that pass through that part of the system. PMI can keep simple counters or compute running averages. PMI counters exist in all of the major subsystems of WebSphere Application Server, such as the Web container, EJB container, connection manager, transport layers, and session management system. PMI keeps track of, for example, the average response time for a particular servlet or the average wait time to obtain a JDBC** (Java Database Connectivity) connection from a connection pool.

The PMI counters can be configured to control which and how much performance data are collected by the runtime. PMI is designed to place as little extra load on the system as possible, allowing it to be enabled in a production environment. But, of course, there is some overhead in collecting performance data, and therefore, it is not advisable to have all PMI counters on all the time. Both the configuration of PMI and the data collected are available through JMX MBean statistics. As a result, there is a standard way to access performance metrics from any WebSphere Application Server server.

WebSphere Application Server provides a tool called the Tivoli Performance Viewer (see Figure 6) that allows a user to explore and record the performance data being collected by a server. However, because PMI is exposed through JMX, any external program, such as a Tivoli product or the on demand manager, can access the performance data collected by a WebSphere Application Server server.

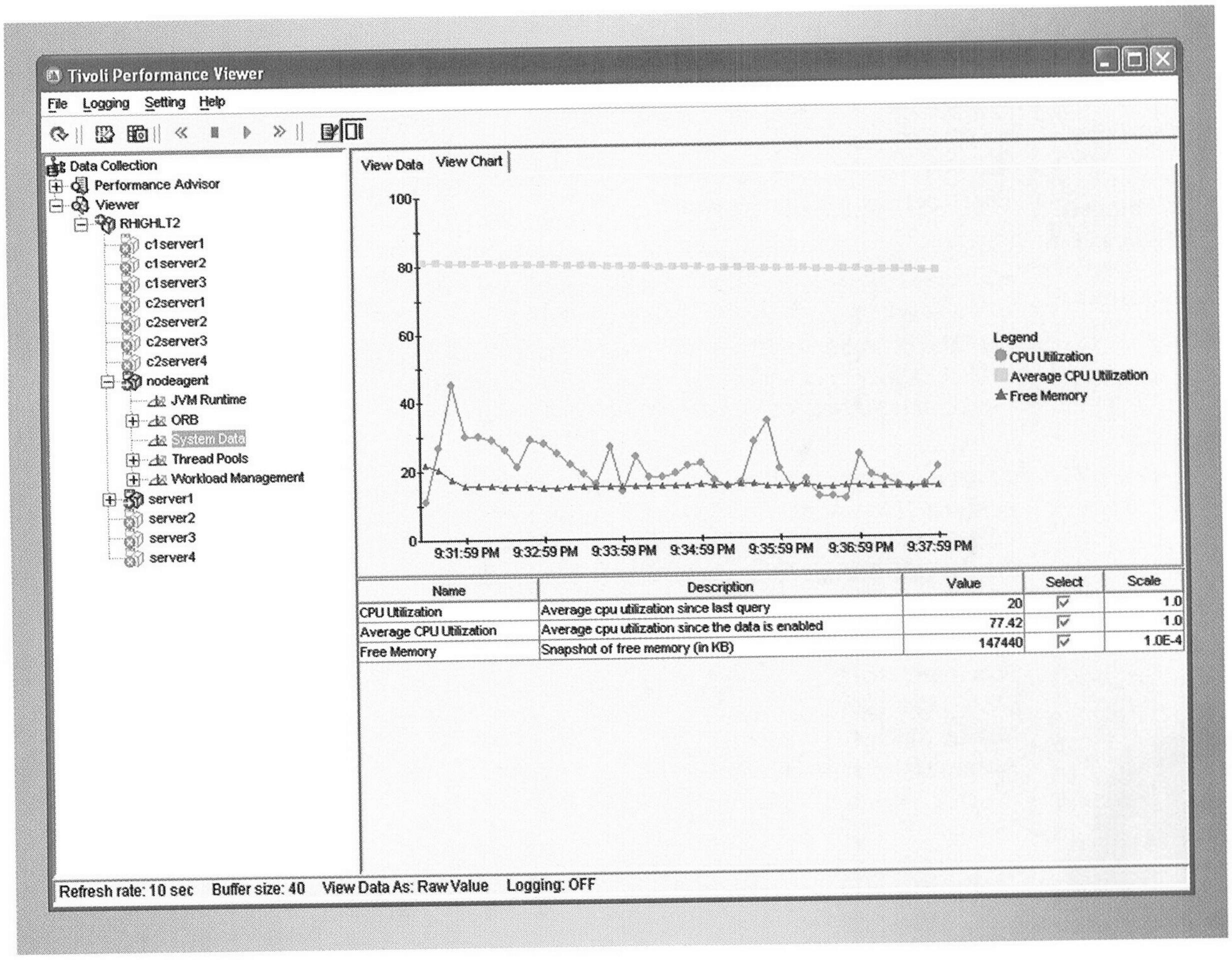
Application response measurement. Per-request data are provided by PMI Request Metrics (PMI-RM). PMI-RM collects data by timing requests as they travel through WebSphere Application Server components. These data help to identify runtime and application problems. PMI-RM logs the time spent at major subsystems, such as the Web server plug-in, Web container, enterprise bean container, and database. These statistics are recorded in logs and can be written to Application Response Measurement (ARM)[39] agents used by Tivoli and other vendor monitoring tools. ARM is an Open Group standard composed of a set of interfaces implemented by an ARM agent that provides elapsed-time statistics for process hops. WebSphere Application Server does not provide its own ARM agent but will work with an ARM agent from Tivoli, Hewlett-Packard Development Company, L.P., and others. Support for ARM is critical to on demand computing because it allows a standards-based mechanism to understand response time for application requests, which can then be used to control provisioning and workload management configuration.

Using this continuous flow of data, the on demand manager can analyze the system on the basis of a knowledge repository and make recommendations on how to reprovision the system to meet a policy or goal.[40] Monitoring provides the input that drives the autonomic nature of on demand computing. WebSphere provides that crucial flow of information through standard JMX interfaces from the PMI system, and through JMX event notifications when critical errors occur. This flow of information is then used to make dynamic workload management decisions and to reconfigure the system on the basis of administrative policy.

Performance advisors. One example of self-tuning that exists in WebSphere Application Server Version 5.0 is the performance advisors[41] (see Figure 7). By collecting data from PMI over time and analyzing the data using rules that have been built through experience with real world applications, the performance advisors make concrete recommendations on system-configuration changes that will result in better performance for the application.

This is a small example of how the WebSphere platform builds on its own capabilities to provide higher-level functions. Although the current performance advisors do not transparently implement the suggested actions, they represent a step toward the vision of a fully autonomic self-tuning system. In a fully autonomic system, the generated recommendations will be applied to the running system automatically while the system is in operation, enabling the closed-loop feedback that is the hallmark of on demand computing.

Figure 6 Viewing PMI data through Tivoli Performance Viewer



Name	Description	Value	Select	Scale
CPU Utilization	Average cpu utilization since last query	20	✓	1.0
Average CPU Utilization	Average cpu utilization since the data is enabled	77.42	✓	1.0
Free Memory	Snapshot of free memory (in KB)	147440	✓	1.0E-4

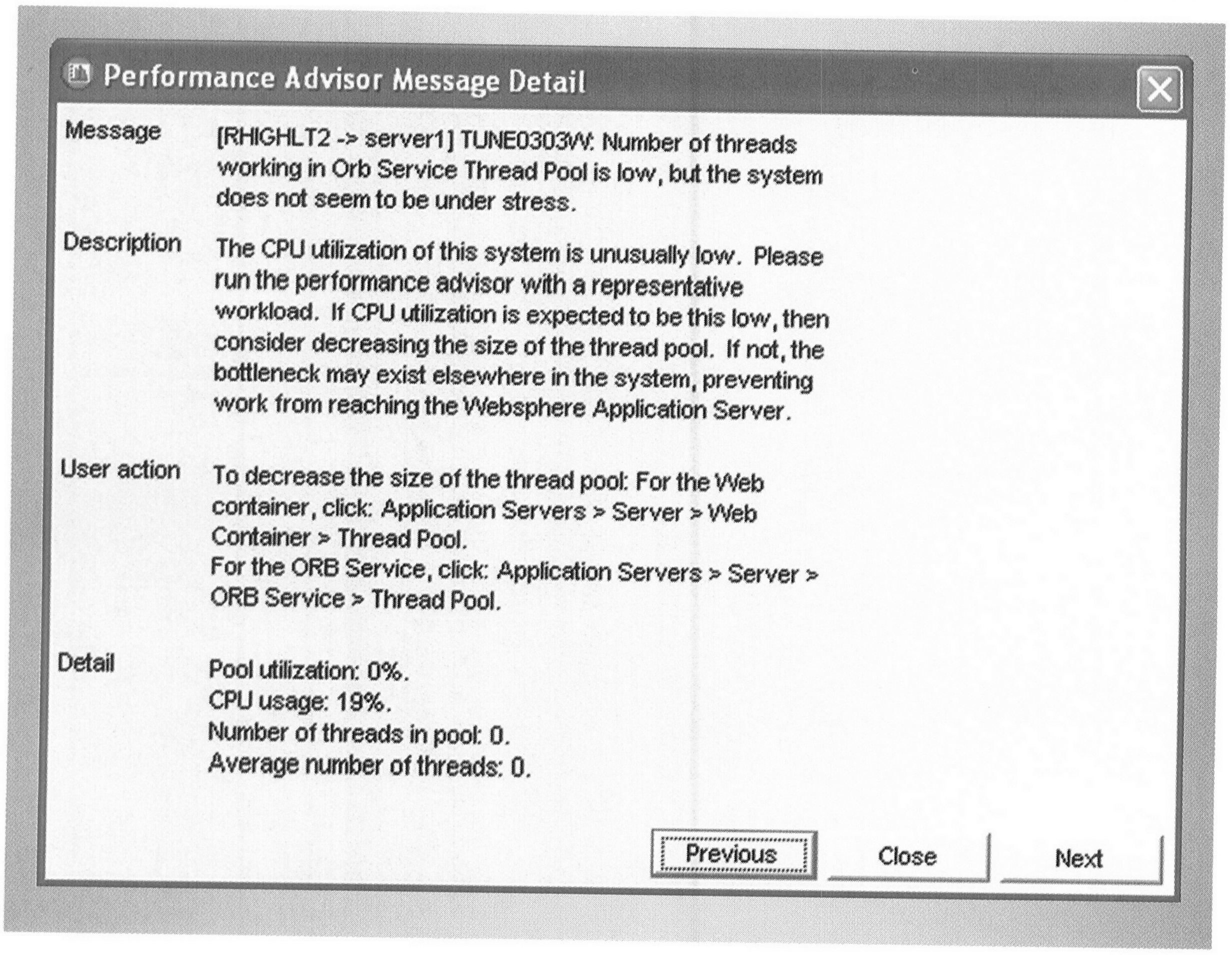
Refresh rate: 10 sec Buffer size: 40 View Data As: Raw Value Logging: OFF

Workload management

Much of on demand computing is about improving the utilization and efficiency of existing computing facilities. A key element to this aspect is ensuring workloads are balanced; that is, they are distributed over those facilities in a way that obtains maximum utility from them, softens the impact of large spikes in workload, and yet ensures computing goals are being met. This is achieved in WebSphere Application Server through workload management services that classify inbound workload, measure the utilization and availability of computing facilities, and prioritize and distribute the work.

Clusters. Workload management in WebSphere Application Server is based on the fundamental concept of a cluster. A *cluster* is composed of two or more application server instances; each instance is hosted in its own Java Virtual Machine (JVM**) process. Each application server instance has the same configuration; that is, it is configured to host the same set of Java applications, to use the same resource definitions in the same domain, and so forth. The cluster operates effectively as a single logical server. Individual server instances within the cluster can be stopped and started, or the cluster can be started and stopped as a whole. Likewise, applications configured to run on the cluster can be stopped and started, resulting in stopping or starting the application running on all instances of the application server in the cluster. From the perspective of the client, however, it appears as though there is just one instance of the application running on one application server instance.

Figure 7 Example of Performance Advisor display



Message [RHIGHLT2 -> server1] TUNE0303W: Number of threads working in Orb Service Thread Pool is low, but the system does not seem to be under stress.

Description The CPU utilization of this system is unusually low. Please run the performance advisor with a representative workload. If CPU utilization is expected to be this low, then consider decreasing the size of the thread pool. If not, the bottleneck may exist elsewhere in the system, preventing work from reaching the Websphere Application Server.

User action To decrease the size of the thread pool: For the Web container, click: Application Servers > Server > Web Container > Thread Pool.
For the ORB Service, click: Application Servers > Server > ORB Service > Thread Pool.

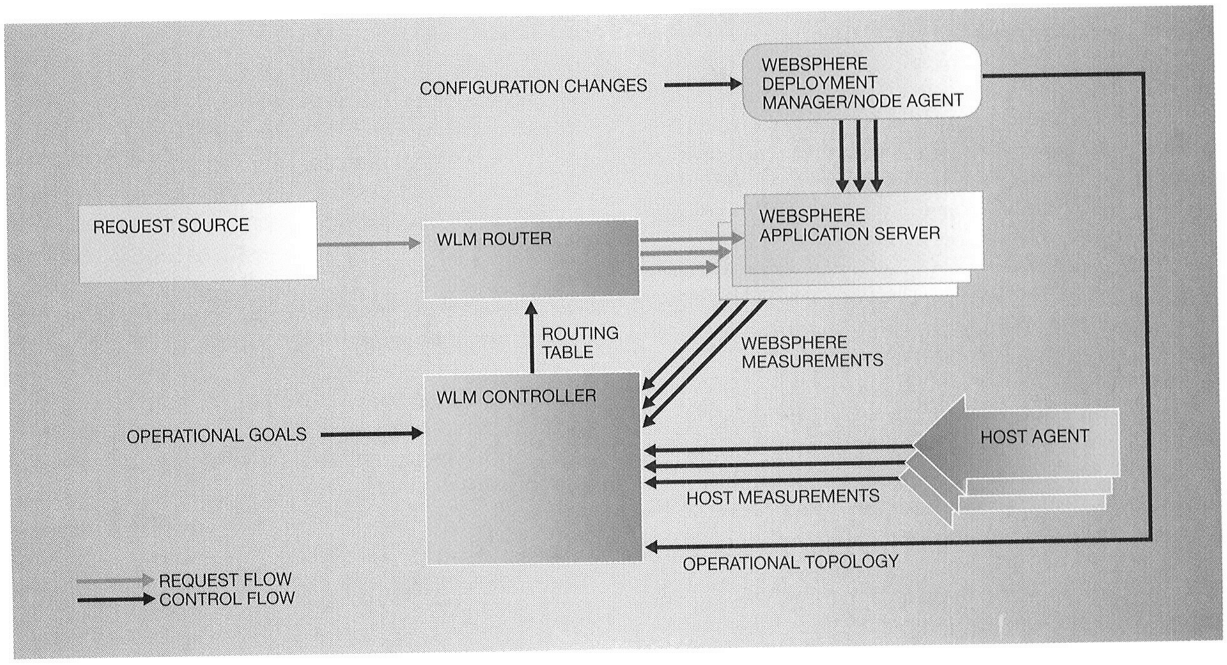
Detail Pool utilization: 0%.
CPU usage: 19%.
Number of threads in pool: 0.
Average number of threads: 0.

Multiple application server instances can be hosted on a single computer (a *vertical cluster*) or on different computers (a *horizontal cluster*), or some combination of the two.[42] Vertical clusters are valuable for better utilization of the computer when the operating system otherwise constrains the availability of resources on a process boundary. For example, if a JVM process is pinned to a single processor on a symmetric multiprocessor (SMP) computer, introducing additional application server instances allows the process to utilize other CPUs on the same computer (presuming they would be assigned to the other processors). Or, if the operating system limits the number of connections that can be formed to a single process, then an increase in the number of effective connections to the computer can be made by increasing the number of application server instances.

Similarly, horizontal clusters let a workload be distributed over more computers. Greater distribution might be necessary if all the memory or CPU cycles that can be allocated to the application servers on one computer are being used.

As with the rest of the topology, the configuration of the cluster (that is, the number and location of application servers in the cluster) is retained in the configuration repository of the WebSphere Application Server management facility. The application server members in the cluster can be added or removed through any of the WebSphere Application Server management user, programming, or command-line interfaces, associating them with an existing or new cluster, or even configuring the appli-

Figure 8 Workload management controller



cation server instance to operate as a stand-alone server (independent of any cluster).

The workload management (WLM) facility is being updated in WebSphere Application Server Version 6.0. As in Version 5.0, WLM will build on the fundamental premise of a cluster. The rest of the facility is composed of three parts: the *host agent*, *workload controller*, and *workload router* as depicted in Figure 8. The host agent is responsible for detecting how much utilization is being obtained from a set of computing resources. Within WebSphere Application Server, resource utilization is measured and collected through the PMI. PMI acts as the host agent for WebSphere Application Server resources. Other workload sensors may supplement or replace PMI. This is most applicable when considering that the actual utilization of the computing facilities may be loaded by other data management, on-line transaction, batch, and other specialized middleware and programs. WebSphere Application Server cannot, by itself, account for all of the workloads that may be assigned to the available computing facilities targeted to other middleware products.

The workload controller is responsible for coalescing the topology of the cluster configuration, the utilization measurements of the available computing facilities, the operational goals of the on demand computing environment, and the classification of an inbound workload to make a routing decision for that workload. It retrieves configuration information about which application server instances are assigned to a given cluster, their end-point addresses, and priorities and weights from the deployment manager for a WebSphere Application Server cell. PMI continuously updates the workload controller with information about available application server instances and their utilization. All of this information is factored into the routing tables.

The workload controller then feeds routing information to the workload router, which then prioritizes and directs that inbound workload to an appropriate server instance within the cluster. Lower-priority work is queued behind higher-priority work. Higher-priority work is moved higher in the dispatch queue, adjusting to priority and policy demands.

It is usually not enough to just consider any given work request by itself. Most work requests occur within a broader context; that is, they are normally part of some larger unit of work, session, or transaction. This context implies something about the state of the target object (i.e., the component on which the work is to be run). In many cases, the state of a target object will be cached in memory in a given application server instance for the duration of that unit-of-work period. The state is loaded in response to the first request that initiated the unit of work. It remains in memory until the unit of work is completed. Subsequent requests within the unit of work are best served at the application server on which the unit of work was started and are said to have *affinity* to that application server.

Furthermore, a given target object may, in turn, have cascading dependencies on other downstream objects. If a routing decision is made to send a work request to a target object on one application server and then that target object in turn calls another object that requires a significant amount of computing resources, the workload controller needs to be aware of the situation and consider it in its routing decision. If the downstream object is hosted on a computer that is overutilized at the time of the request, it may be appropriate to prioritize other work ahead of it, which does not depend on the bottlenecked resource. This is discussed further in the section "Application profiling."

The workload controller is generally responsible for determining whether a cluster has the correct amount of capacity allocated to it for the workloads that are being routed to it. If demand surges at a certain time of day, the workload controller will examine the capacity management policies and, presuming more capacity is warranted, it will direct the WebSphere Application Server management infrastructure to increase the capacity of the cluster by adding more application server instances to the cluster. Conversely, if capacity is needed elsewhere, it will reallocate capacity from a cluster that has an excess by removing application server instances from the cluster. This capability is being added in Version 6 with relatively simple policy processing support. It will be possible to plug in more sophisticated policy management facilities such as those being developed by Tivoli.

However, an application server instance cannot be taken off-line without potentially affecting the in-flight requests associated with that server instance. WebSphere Application Server Version 5.0 employs a transactional quiescence mechanism that will block new requests from being dispatched on a server, but will allow previously in-flight transactions to be completed. That includes letting new requests enter the server if they are part of an already existing transaction. The additional requests likely will need to be completed before the transaction can be completed successfully.

WebSphere Application Server Version 6.0 will offer the same support for workload, affinity, and capacity management for all the major application protocols supported by the application server, including HTTP messages carrying Web-application and Web-service requests, RMI/IIOP messages for EJB requests, JMS, and the underlying protocol of the WebSphere Application Server messaging engine for asynchronous messaging and notifications. Workload will be routed for messages flowing through all elements of a WebSphere Application Server network, including the proxy server, Web server, Web services gateway, in-network messaging engines, and the application server itself.

Application profiling

J2EE promises to allow the building of reusable and shared components hosted in a managed environment that maintains a separation between the business logic and the underlying infrastructure. One consequence of this promise is that the component, and especially the underlying container assigned to manage that component, must be prepared to be used under a variety of different conditions.

This promise, of course, also requires that the container know something about the dependencies that each object has on other objects. In some cases, an object may have a dependency on another object but only exercise that relationship occasionally, perhaps depending on input arriving with the request.

To understand how different clients can use the same shared component differently and how such use will affect the management of a component, consider the following example. Imagine that a customer Account object—something that maintains the balance of the account and has operations for getting that balance and crediting or debiting the account—has been implemented. A banking application needs very accurate balance information. The container should reread the balance data from the underlying database for every operation on that Account object, lest the balance were changed by some other activity within the information system.

In contrast, a demographic analysis application might be building statistical information about account value trends for a set of customers with a given set of attributes. The demographic analysis application does not need absolutely current balance information, but does need to optimize the path length for obtaining balance information over a potentially large number of accounts. Thus, if the balance value is already cached in memory somewhere and is not too old, it should be returned, without rereading the balance from the disk (an operation with a longer path).

A third application may be responsible for computing interest on the average balance of the Account object over a period of time. This application will need to traverse the related AccountHistory object. The container should read the state of the related AccountHistory objects at the same time that it is reading the state of the Account object.

To ensure maximum efficiency in the information system, while also fulfilling the integrity requirements of the application, the container needs to know something about the client and what it expects. Information about whether a given client is likely to use an object in a certain way, or that causes it to exercise a known relationship, can be associated with an application profile. The task identity for the client is correlated by the runtime with an application profile[43] that represents its usage pattern. The runtime containers then will manage the object accordingly for any requests coming from that client. The application profile is a way of expressing usage policies and associating them implicitly with the target object. The application does not have to explicitly code the management of the components it uses.

Policies within the profile can be set to govern which object relationships are likely to be exercised and therefore which related objects to preload, whether the use of the object is likely to result in updating its state (and therefore the isolation policy that should be applied to its state in the data system), the rate at which elements of the results set are going to be consumed (and therefore the optimal prefetch size), the currency requirements for the state, and several other issues. The result is to dramatically improve the efficiency of the overall system.

High availability and failover

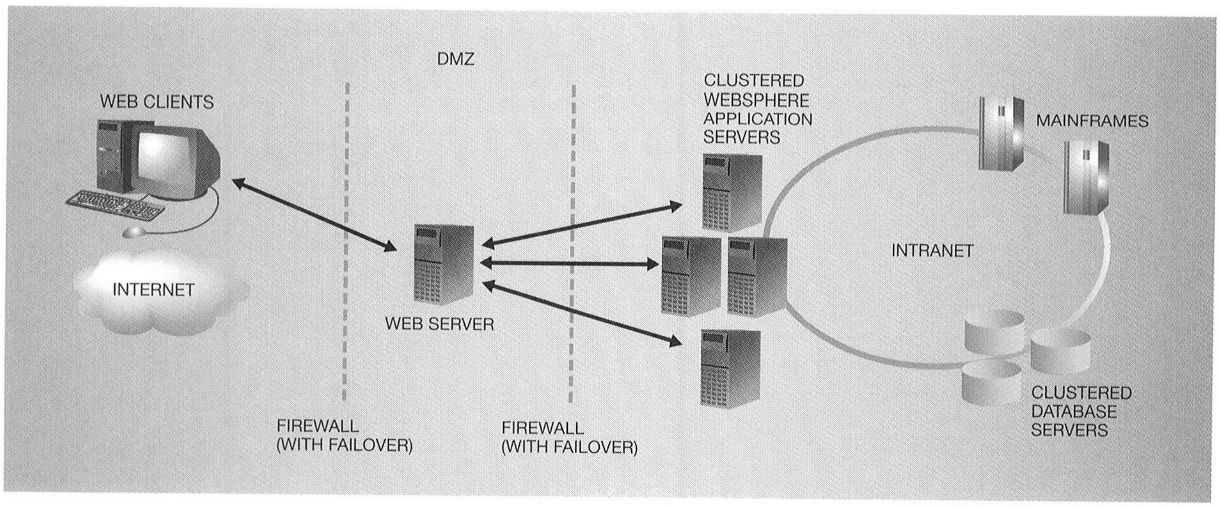
On demand computing is not just about responding to fluctuations in demand against available capacity based on a set of policy goals. Even in an on demand computing center, computers do and will fail unexpectedly. Both planned and unplanned outages are a part of managing a large data center. Actually, the probability of some computer in the data center being out of service at any given moment goes up as the number of computers in the center increases.

WebSphere Application Server handles outages with the same clustering technology that it uses for on-line workload management. When a computer fails, that computer's capacity is simply removed from the list of available computing capacity for the system, and workload is routed to other on-line computers in the information system. When failed computers are brought back on-line, they are automatically added back into the available resource list and are then utilized as soon as capacity requirements suggest shifting workload to those computers.

It is common when a computer fails suddenly to have the work that was in flight on that computer become orphaned. A good example of this involves message queues.[44] Assume that a particular queue manager is configured to run in a particular application server instance for messages that have accumulated in a particular queue and persisted in a queue associated with that queue manager. If the computer on which that queue manager is located fails, none of the messages that had been queued for the queue manager will be processed until that computer and queue manager are brought back up. Whereas on the one hand, messaging systems do not generally presume any particular temporal assurances, orphaning these messages for a long period of time could have an adverse affect on the customer's business. On the other hand, multiple queue managers should not be operating on a message queue. Doing so would require complex coordination between them to ensure that each message is processed only once. (One would not want the same debit message removing money from one's account twice, for example.) Coordinating these kinds of assurances between two or more queue managers operating on a single shared message queue can be very expensive.

In an on demand computing system, it is expected that the queue manager will be automatically reconfigured to run on another computer, possibly on an application server that is already running. The messages in the original queue should be reassigned to the new instance of that queue manager. WebSphere Application Server will be adding support to do exactly this by managing ownership within a cluster of queue managers. At any given time only one queue manager will operate on a given message queue (al-

Figure 9 Basic WebSphere Application Server clustering topology



though other queue managers in the cluster can be operating on other queues). If any queue manager fails, another queue manager will be assigned to operate on that message queue, including recovering any failed transactions that may have been in flight on the queue manager that failed.

The worst kind of failure that a data center can suffer is a disaster that brings down the entire data center. For this situation, many on demand enterprises will operate two or more data centers that are geographically dispersed, but interconnected by broadband channels. WebSphere Application Server allows the interconnection of two or more cells located in different data centers. One cell can act as a failover site for a cell in another data center. If all of the resources in a cluster in one cell fail, then the workload clients will reroute workload to the other data center. This action, of course, presumes that other provisions have been made to synchronize or transfer the data of the underlying data system from one site to the other.

The clustering, workload distribution, failover, and recovery mechanisms built into WebSphere Application Server are all essential to ensuring that the information system is always available, balanced, and responsive to varying computing needs and fluctuations in utilization and capacity, and thus is on demand.

Edge computing

The typical information computing topology is evolving. At one point, the standard topology for Web and business computing consisted of a browser on a desktop computer, attached through the Internet or intranet, through a firewall, to a Web server, perhaps through a second firewall to the application server hosting Web and business components, and finally attached to a back-end data system as depicted in Figure 9. Perhaps the browser was actually a client application running on a handheld device or mobile telephone. Perhaps an Internet Protocol (IP) sprayer was injected in front of the Web server to statically distribute workload. But the nature of many of these components is changing. More often, customers are introducing forward-proxy caching servers in front of their Web servers, or even out in the network to cache the content of Web pages for performance and scalability. Web servers themselves are being enhanced with caching, security, and intrusion-detection features to guard against denial-of-service attacks. The role of Web services has introduced the need for gateway servers to convert between external and internal networks and to hide private services. Messaging engines are being inserted into corporate networks to perform mediation, routing, and flow control. Data systems are being clustered for scalability. Even business applications are being decomposed and distributed to multiple separate application servers.

These changes are altering the shape of traditional topologies.[45] The roles of many network components are evolving and, in some cases, converging. Already, the role of the application server is evolving to include serving some of the static content that is in the traditional domain of Web servers. Proxy servers are sharing some of the same security and caching technologies that are often found in firewalls and application servers. Even browsers are being extended to support hosting of business components for localized, mobile, and off-line processing.

In addition to this evolution, the relationships and management of all of these components are crucial to maintaining an on demand computing facility. If different Web servers are assigned to handle the request traffic for different applications, and those applications are being moved around the on demand data center in response to fluctuations in utilization and capacity, the Web servers need to be reconfigured accordingly. If proxy servers cache Web content for specific applications, they may need to be flushed if the application is stopped or taken off-line. If the workload balancing policies for an application are changed, then the workload routers and gateways need to be updated.

WebSphere has responded to this situation by reengineering the WebSphere caching proxy (also referred to as the edge server), IBM HTTP Server (IHS), Web Services Gateway (WSGW), and WebSphere messaging server to fundamentally build on the same server underpinnings used in WebSphere Application Server. Doing so yields two benefits. These additional servers can exploit the same underlying management infrastructure that is integral to WebSphere Application Server, and by extension, then, these servers can be configured and managed along with WebSphere Application Server within a single topology configuration model.

The entire breadth of the WebSphere Application Server programming model, including the J2EE programming model for Web and business components, Web services, handlers, mediations, and plug-ins, can be enabled or disabled for use anywhere in the network. In this way, function can be placed where it is needed, based on the specific combination of needs of the customer, whether that is to gain better response time performance in the network or in branch offices, or to extend the functionality of the DMZ (see Figure 9) with system services written according to the same programming model used by business application developers, or to apply additional security controls or workload classification at the back end of the internal networks—on demand.

This rebasing of the peripheral servers on a common runtime infrastructure and the enabling of the full breadth of the programming model at various positions within the topology enables a new business model: *on demand edge computing*. Several vendors are already building large delivery networks within the backbone of the Internet to leverage application servers. Akamai Technologies, Inc. is an obvious example of such a vendor.[46] They are running a network of some 15000 servers in the Internet, all supporting WebSphere Application Server servers. Web applications can be built and arranged to be hosted by Akamai in their network. The result is service delivery that is much closer to the Internet end users of an enterprise, creating a much more responsive experience for them, and leveraging the capacity of their vast network in response to spikes in demand.[47]

Several changes have been made to WebSphere Application Server to enable this type of infrastructure. In particular, one can imagine that Akamai would prefer not to have all of the applications they are hosting for their broad set of customers installed on every one of their 15000 servers. Generally, the distribution of a given application is relatively sparse most of the time, based on where that application is actually being used at any given time in the world. For example, if the customer had hosted a trading application in the wide area network, it would be expected that the application is to be used by customers primarily after work, in the evening of their own local time. This peak utilization rolls around the world throughout the day. The application may, in fact, be infrequently used for some 18 hours a day at any given point in the world.

Edge server providers would prefer not to have the application installed and consuming server resources when it is sitting idle. They actually want to remove the application from the computers where it is not being used. Conversely, they want to be able to install the application and bring it up to a running state very quickly when demand does pick up for the application. For this, WebSphere Application Server has introduced support to preinstall an application, resolve its reference bindings to dependent resources, then allow the application to be "zipped up" and subsequently removed from the computer. Later, when the customer needs to reinstall the application, it can simply unzip the application into its original directories and point WebSphere Application Server to the presence of the application.

Edge server providers are also concerned about applications inadvertently consuming underlying server resources unnecessarily, and are concerned about reducing the risk of applications destabilizing their execution environment. For both of these reasons, WebSphere Application Server introduced configuration options that turn off automated deployment and generation mechanisms. In particular, the automatic JSP compiling function within WebSphere can be turned off. When automated JSP compiling is turned on, if an application is loaded and a JSP is referenced that has not been compiled, WebSphere will automatically compile it and proceed to service the request. However, the compilation time takes away from the execution cycles of other requests and delays the outstanding request while the compilation takes place. Further, the fact that the JSP had not been previously compiled in an environment where the application should have already been preinstalled, and where the application is used frequently, raises questions: Why is the JSP not already compiled? The biggest concern is that someone may access the application and change a JSP—perhaps inadvertently, as part of a deployment step that is executed out of sequence. There is a concern that this sort of mistake may imply a result that will fail in the execution environment. Thus, it is possible to turn off automated compilation to remove on demand preparation of the application with the intent of increasing the efficiency and availability of the on demand computing environment.

Grid computing

Computing grids represent a core topology structure for on demand computing, and even more so in utility computing environments. The idea that a data center can be composed of a large collection of computing resources interconnected in a grid, or that a data center can be interconnected with other data centers to extend the grid even further, and then be able to dynamically allocate resources from that grid in response to on demand requirements for a given set of applications is a central theme in on demand computing. We have already discussed how WebSphere Application Server provides the virtualization capabilities and programming model that enable applications to be deployed and configured dynamically across the grid infrastructure.

The relationship of grid computing to WebSphere Application Server is more specific. The Global Grid Forum (GGF)[48] defines the Open Grid Service infrastructure (OGSi).[49] OGSi, in part, defines extensions to contemporary Web services standards for stateful business components that can be deployed in a grid computing infrastructure. In fact, these extensions are very generally applicable to business applications, regardless of whether they are deployed to a grid infrastructure or just a simple stand-alone server. In many ways, the grid infrastructure is not specific to grid computing; it just happens to introduce a need for more general-purpose Web services functions sooner than they are being standardized in the mainstream Web services communities.

Nonetheless, because WebSphere Application Server serves as the foundational underpinnings for IBM's Web services runtime, it is, by extension, the foundation for OGSi. IBM's reference implementation of OGSi is implemented on it.[50] Further, WebSphere Application Server is internalizing the runtime and component model support for grid services. WebSphere Application Server is driving the standards work in Web services to ensure that grid service components are fully embraced in those standards. The principles of service interface inheritance, stateful services (resources), service references, service state, soft-state life-cycle management and state notifications, introduced by OGSi, are being incorporated into the general-purpose Web services infrastructure and will be supported by WebSphere Application Server and made available to a wide variety of deployment topologies, including those that are most associated with grid computing.

Concluding remarks

The role of information computing in businesses continues to grow. Few activities of a modern business are done without the automation and productivity benefits that computers bring. For the most part, information computing is in principle a cost of doing business—a necessary activity to remove the tedium of repetitive tasks or to assist employees to perform their jobs. Information computing has taken such a strong hold on the basic ways in which businesses operate that it is hard to imagine doing anything without it. In this role, businesses need to be able to obtain more utility from their investment in computing resources. They need to manage their computing resources more efficiently to be able to share resources over a broader range of applications and to be able to dynamically adjust the allocation of those resources to different applications as demand for each application fluctuates throughout the day.

We stand on the threshold of a major shift in the role of computing—from being the foundation on which businesses are built, to being the driving force behind business competitiveness. On demand computing will enable businesses to model their business processes and from that be able to detect their own business weaknesses and advantages. Through business process modeling, a business can quickly adjust its processes to remove unnecessary activities, or to improve its own competitive strengths, and expect that changes in that model will immediately result in changes to the underlying computing applications that implement those business processes. Ultimately, on demand computing enables a business to drive and respond to changes in its markets at the rate at which changes occur in the marketplace. Enterprise computing moves from being simply a cost center weighted down by the complexity of information systems to being a true cost-effective asset for a business.

WebSphere Application Server is at the leading edge of this transformation, enabling the separation of business logic from the underlying information technology needed to allow applications to be dynamic and mobile. It virtualizes the computing environment for these applications. It is instrumented to enable the execution environment for these applications to be monitored, to allow provisioning managers to determine what utilization they are obtaining from the system, and then to adjust the system to improve its efficiency. It supports clustering for workload management and availability and can be dynamically reconfigured to increase or decrease the capacity of the different applications hosted on the application server. It has built-in support for maximizing the efficient management of application components based on different usage patterns for different clients. WebSphere Application Server is the foundation for IBM's strategy for on demand computing.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Open Software Group, Object Management Group, or Corporation for National Research Initiatives.

Cited references and note

1. "Meet the Experts: Irving Wladawsky-Berger on E-business on Demand," *developerWorks*, IBM Corporation (February 2003), http://www7b.software.ibm.com/dmdd/library/techarticle/0302iwb/0302iwb.html.
2. T. Francis, E. Herness, R. High, J. Knutson, K. Rochat, and C. Vignola, *Professional IBM WebSphere 5.0 Application Server*, ISBN 0-7645-4366-0, John Wiley & Sons, Inc., New York (2002).
3. WebSphere Application Server has become the leading application server, gaining more market share than any other application server, including .NET from Microsoft Corporation and WebLogic from BEA Systems.
4. E. Millard, "IBM's WebSphere vs. Microsoft's .NET—Who's Winning?" *NewsFactor Network* (July 30, 2002), http://www.newsfactor.com/perl/story/18818.html.
5. "How IBM Hopes to Waltz around Microsoft," *GRID Today* **1**, No. 4 (July 8, 2002), http://www.gridtoday.com/02/0708/100083.html.
6. M. LaMonica, "IBM Pulls Away in App Server Race," *C/Net News.Com* (May 6, 2003), http://news.com.com/2100-1012_31000046.html.
7. U. Richter, L. Ostdick, and C. Diep, *Architecture for Virtualization with WebSphere Application Server, Version 5*, IBM Corporation (2003).
8. R. Willenborg, K. Brown, and G. Cuomo, "Designing the WebSphere Application Server for Performance: An Evolutionary Approach," *IBM Systems Journal* **43**, No. 2, 327–350 (2004, this issue).
9. R. Bakalova, A. Chow, C. Fricano, P. Jain, N. Kodali, D. Poirer, S. Sankarar, and D. Shupp, "WebSphere Dynamic Cache: Improving J2EE Application Performance," *IBM Systems Journal* **43**, No. 2, 351–370 (2004, this issue).
10. C. M. Saracco, M. A. Roth, and D. C. Wolfson, "Enabling Distributed Enterprise Integration with WebSphere and DB2 Information Integrator," *IBM Systems Journal* **43**, No. 2, 255–269 (2004, this issue).
11. R. Will, S. Ramaswamy, and T. Schack, "WebSphere Portal: Unified User Access to Content, Applications, and Services," *IBM Systems Journal* **43**, No. 2, 420–430 (2004, this issue).
12. M. Kloppman, D. Koenig, F. Leymann, G. Pfau, and D. Roller, "Business Process Choreography in WebSphere—Combining the Power of BPEL and J2EE," *IBM Systems Journal* **43**, No. 2, 270–296 (2004, this issue).
13. J. Ponzo, L. D. Hassan, J. George, G. Thomas, O. Gruber, R. Konuru, A. Purakayastha, R. D. Johnson, J. Colson, and R. A. Pollak, "On Demand Web-Client Technologies," *IBM Systems Journal* **43**, No. 2, 297–315 (2004, this issue).
14. S. M. Fontes, C. J. Nordstrom, and K. W. Sutter, "WebSphere Connector Architecture Evolution," *IBM Systems Journal* **43**, No. 2, 316–326 (2004, this issue).
15. F. Marinescu, *EJB Design Patterns; Advanced Patterns, Processes, and Idioms*, ISBN 0-471-20831-0, John Wiley & Sons, Inc., New York (2002).
16. J. Adams, S. Koushik, G. Vasudeva, and G. Galambos, *Patterns for E-business—A Strategy for Reuse*, ISBN 1-931182-02-7, IBM Press (MC Press), Lewisville, TX (2001). Also see "IBM Patterns for E-business; Navigating You to a New Generation of E-business Applications," IBM Corporation, http://www.ibm.com/developerworks/patterns/.
17. P. Giangarra, "J2EE: Not Just a Language—It's a Platform," *Proceedings of SHARE 2001* (2001), http://www.share.org/proceedings/sh97/data/S3546.PDF.
18. G. Desai, E. Sanchez, and J. Fenner, "Web Application Servers Come of Age," *CMP Network Computing* (July 23, 2001), http://www.networkcomputing.com/1215/1215f4.html.
19. *Java Servlet Technology*, Sun Microsystems, Inc., http://java.sun.com/products/servlet.
20. *JSR 168 Portlet Specification*, Java Community Process, http://jcp.org/en/jsr/detail?id=168.
21. *JavaServer Pages Technology*, Sun Microsystems, Inc., http://java.sun.com/products/jsp/index.jsp.

22. W. Appel, "Enterprise Architecture: An In-Depth Study," Enterprise Architecture Community, http://www.eacommunity.com/articles/openarticle.asp?ID=1840.
23. R. Sharma, B. Stearns, and T. Ng, *J2EE Connector Architecture and Enterprise Application Integration*, ISBN 0201775808, Addison-Wesley Publishing Co., Boston, MA (2001).
24. G. Van Huizen, "JMS: An Infrastructure for XML-Based Business-to-Business Communication," *Java World* (February 2000), http://www.javaworld.com/javaworld/jw-02-2000/jw-02-jmsxml.html.
25. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business Process Execution Language for Web Services" (May 5, 2003), ftp://www-106.ibm.com/developerworks/webservices/library/ws-bpel11.pdf.
26. T. Mikalsen, I. Rouvellou, and S. Tai, *Reliability of Composed Web Services from Object Transactions to Web Transactions*, IBM Corporation, T. J. Watson Research Center, Yorktown Heights, NY, http://www.research.ibm.com/people/b/bth/OOWS2001/mikalsen.pdf.
27. M. Stevens, "Service-Oriented Architecture Introduction, Part 1," *developer.com* (April 16, 2002), http://www.developer.com/services/article.php/1010451.
28. "Service-Oriented Architecture (SOA) Definition," *Barry & Associates*, http://www.service-architecture.com/web-services/articles/service-oriented_architecture_soa_definition.html.
29. J. Waldo, G. Wyant, A. Wollrath, and S. Kendall, *A Note on Distributed Computing*, SMLI TR-94-29, Sun Microsystems, Inc., Mountain View, CA (November 1994), http://research.sun.com/techrep/1994/smli_tr-94-29.pdf.
30. D. Orchard, "Myth of Loose Coupling," W3C (January 2003), http://lists.w3.org/Archives/Public/www-ws-arch/2003Jan/0115.html.
31. P. Kovari, D. C. Diaz, F. C. H. Fernandes, D. Hassan, K. Kawamura, D. Leigh, N. Lin, D. Masic, G. Wadley, and P. Xu, *WebSphere Application Server Enterprise V5 and Programming Model Extensions*, SG24-6932, WebSphere Handbook Series, Redbooks, IBM Corporation (August 2003), http://www.redbooks.ibm.com/redbooks/pdfs/sg246932.pdf.
32. Java 2 Platform, Enterprise Edition (J2EE), Sun Microsystems, Inc., http://java.sun.com/j2ee/index.jsp.
33. L. Williamson, "System Administration for WebSphere Application Server V5, Part 4; How to Extend the WebSphere Management System (and Create Your Own MBeans)," *IBM WebSphere Developer Technical Journal*, IBM Corporation (April 23, 2003), http://www.ibm.com/developerworks/websphere/techjournal/0304_williamson/williamson.html. See also Part 1, http://www.ibm.com/developerworks/websphere/techjournal/0301_williamson/williamson.html.
34. *J2EE Management Specification*, Sun Microsystems, Inc., http://java.sun.com/j2ee/tools/management/reference/docs/index.html.
35. I. K. Lam and B. Smith, "Jacl: A Tcl Implementation in Java," *Proceedings of the Fifth Annual Tcl/Tk Workshop*, USENIX, Boston, MA (July 1997), http://www.usenix.org/publications/library/proceedings/tcl97/full_papers/lam/lam.pdf.
36. Overview of Jython Documentation, http://www.jython.org/docs/index.html.
37. IBM Tivoli Provisioning Manager and IBM Tivoli Intelligent ThinkDynamic Orchestrator Enable on Demand Computing to Improve Server Utilization, IBM Corporation, http://www-3.ibm.com/software/tivoli/products/prov-mgr/.
38. "Meet the Experts: Ruth Willenborg on WebSphere Performance," *developerWorks*, IBM Corporation (August 2003), http://www7b.software.ibm.com/wsdd/library/techarticles/0308_willenborg/willenborg.html.
39. Application Response Measurement-ARM, The Open-Group, http://www.opengroup.org/tech/management/arm/.
40. M. Pistoia and C. Letilley, *IBM WebSphere Performance Pack: Load Balancing with IBM SecureWay Network Dispatcher*, SG24-5858, Redbooks, IBM Corporation (October 1999), http://www.redbooks.ibm.com/redbooks/pdfs/sg245858.pdf.
41. G. Trotta, "WebSphere 5's New Tools: From Back-End to End-User," *ebiz* (September 8, 2003), http://www.ebizq.net/topics/dev_tools/features/2708.html.
42. *Clusters*, IBM Corporation (March 27, 2003), http://publib.boulder.ibm.com/infocenter/ws51help/index.jsp?topic=/com.ibm.websphere.nd.doc/info/ae/crun-srvgrp.html
43. *Application Profiles*, IBM Corporation (March 28, 2003), http://publib.boulder.ibm.com/infocenter/wasinfo/index.jsp?topic=/com.ibm.wasee.doc/info/ee/appprofile/tasks/tapp_assembleprofiles.html.
44. S. Meridew, "WebSphere MQ Clustering and High Availability," *Proceedings of SHARE 2002*, (March 4, 2002), http://www.share.org/proceedings/sh98/data/S1103.PDF.
45. *Computing at the Edge*, White Paper, Sun Microsystems, Inc. (2003), http://www.sun.com/servers/entry/lx50/pdfs/whitepapers/whitepaper.edge.pdf.
46. C. Moore, "Akamai, IBM Team Up for Edge Computing," *InfoWorld* (May 8, 2002), http://www.infoworld.com/article/02/05/08/020508hnibmakamai_1.html.
47. C. Haley, "IBM, Akamai Boost 'Virtual Capacity'," ASPnews.com (May 2, 2003), http://www.aspnews.com/news/article.php/2200501.
48. "Global Grid Forum Overview," *GGF*, http://www.ggf.org/L_About/about.htm.
49. S. Tuecke, K. Czajkowski et al, "Open Grid Services Infrastructure (OGSI) Specification," *GGF* (April 5, 2003), http://www.gridforum.org/ogsi-wg/drafts/draft-ggf-ogsi-gridservice-29_2003-04-05.pdf.
50. J. Mears, "IBM Adds Grid Computing to WebSphere," *Network World Fusion* (July 28, 2003), http://www.nwfusion.com/news/2003/0728grid.html.

Accepted for publication January 6, 2004.

Eric N. Herness *IBM Software Group, 3605 Highway 52 North, Rochester, Minnesota 55901 (herness@us.ibm.com).* Mr. Herness is a Distinguished Engineer with the IBM Software Group. He is currently the chief architect for WebSphere Business Integration. He is a senior member of the WebSphere Foundation Architecture Board and a member of the Software Group Architecture Board. He has also been heavily involved in championing and implementing the EJB 2.0 specification in WebSphere, especially those parts that enable container-managed persistence. Mr. Herness has been involved in object technology and servers that host objects since 1989. In the early years, he drove work on object analysis and design methods, defining how to practically leverage these concepts in large-scale software projects within and outside IBM. He played a lead role in IBM's implementations of CORBA and the early component model definition work that planted many of the seeds we now see flourishing in J2EE. He holds a B.S. degree in business administration with an information systems emphasis from the University of Wisconsin at Eau Claire and an M.S. degree in business administration from the Carlson School of Management at the University of Minnesota. He has also been an adjunct computer science faculty member at Winona State University.

Rob H. High, Jr. *IBM Software Group, 11501 Burnet Road, Austin, Texas 78758 (highr@us.ibm.com).* Mr. High is a Distinguished Engineer and the chief architect for the WebSphere Application Server foundation. He has 26 years of programming experience and has worked with distributed, object-oriented, component-based transaction monitors for the last nine years, including SOMObject Server and Component Broker, prior to WebSphere. He helped to define, and then later refine, the basic concepts of container-managed component technology, which is now intrinsic to the EJB specification and implemented by WebSphere and other J2EE application servers. He started his career with IBM in 1981 in Charlotte, North Carolina, and during his 12 years there, he primarily worked in the finance industry sector as a developer on the 4700 controller and on 4730 and 4736 ATM microcode with responsibility for the device access methods. He led the development of Application Foundation PC software for retail branch computing, culminating in responsibility for the Financial Application Architecture. In 1993 he moved to Austin to lead IBM's participation in the Object Management Framework of the Open Software Foundation, which led eventually to his involvement in SOMObjects®, and later Component Broker and WebSphere. Mr. High received a B.S. degree in computer and information science from the University of California at Santa Cruz in 1981.

Jason R. McGee *IBM Software Group, 4205 South Miami Boulevard, Research Triangle Park, North Carolina 27709 (jrmcgee@us.ibm.com).* Mr. McGee is a Senior Technical Staff Member and chief architect for the Base and Network Deployment versions of WebSphere Application Server. He is also a senior architect on the WebSphere Foundation Architecture Board and an associate member of the Software Group Architecture Board, focusing primarily on WebSphere family programming model design issues. He joined IBM in 1997 and has been a member of the WebSphere Application Server product team since its inception. He helped to define the concepts of servlets and JavaServer Pages (JSP) for processing Web presentation logic on the server and has been instrumental in leading those parts of the J2EE specification. He was responsible for the design and implementation of the Web container in WebSphere Application Server. Mr. McGee has been heavily involved in leading the architecture for key parts of the WebSphere Application Server, including the server runtime framework and the XML-based systems management architecture. He graduated with a B.S. degree in computer engineering from Virginia Polytechnic Institute and State University in 1995.