WebSphere. software

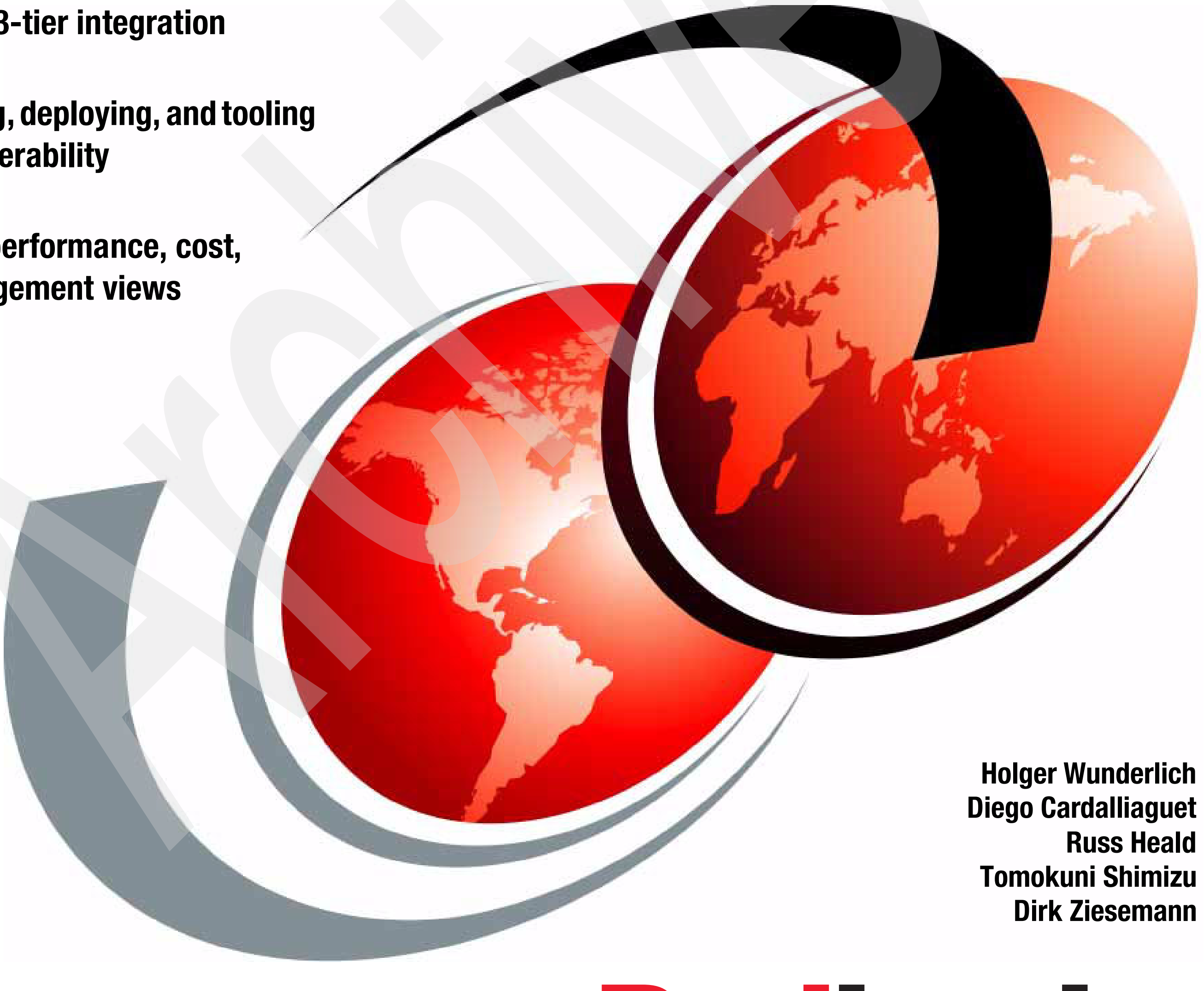
IBM

Building Multi-Tier Scenarios for WebSphere Enterprise Applications

Architecting an infrastructure for seamless 3-tier integration

Developing, deploying, and tooling for interoperability

Security, performance, cost, and management views



Holger Wunderlich
Diego Cardalliaguet
Russ Heald
Tomokuni Shimizu
Dirk Ziesemann

Redbooks

ibm.com/redbooks

International Technical Support Organization

Building Multi-Tier Scenarios for WebSphere Enterprise Applications

August 2003

SG24-6956-00

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (August 2003)

This edition applies to WebSphere Application Server V4.01 for z/OS and OS/390.

Contents

Notices

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

eserver™
AIX®
AS/400®
CICS Connection®
CICS®
DB2 Connect™
DB2®
DRDA®
Encina®
HiperSockets™
IBM®
IMS™
iSeries™
MQSeries®
MVS™
Net.Data®
OS/390®
OS/400®
Parallel Sysplex®
pSeries™
RACF®
Redbooks™
Redbooks(logo) ™
RMF™
SupportPac™
Tivoli®
xSeries®
z/OS®
z/VM®
zSeries®
WebSphere®

The following terms are trademarks of other companies:

Intel, Intel Inside (logos), MMX and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook will help you build multi-tier scenarios for WebSphere® Enterprise Applications. It applies to WebSphere Application Server V4.01 for z/OS® and OS/390®.

We cover the aspects of architectural, organizational, and technical issues that you need to consider when selecting an application and runtime design. This book can be used in conjunction with Patterns for e-business when you are faced with making decisions about application patterns and are looking for supporting information.

Because our analysis is done from the perspective of the z/OS platform, we discuss strategies for offloading Web applications from z/OS or from WebSphere for z/OS. We provide ano overview of different scenarios and give guidance on platforms, security, deployment, performance, scaling, and EIS integration.

Using this redbook will enable you to architect an infrastructure for seamless three-tier integration by helping you to develop, deploy, and tool the application for interoperability, as well as install and configure the different infrastructures.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Figure 0-1 The team, from the left: Tomokuni, Russ, Diego, Holger and Dirk

Holger Wunderlich is a Project Leader at the IBM International Technical Support Organization, Poughkeepsie. He writes extensively and teaches IBM classes worldwide on all areas of e-business with IBM zSeries®. Before joining the ITSO in 2001, he was a cross-server consulting IT Specialist in Germany. He worked for the IBM eServer Technical Support Team. He also worked in the OS/390 area as a systems programmer, systems engineer, and as technical support specialist in the USS and WebSphere world for three years.

His areas of expertise include cross-server consulting, OS/390 UNIX System Services, e-business infrastructure, e-business security concepts, large scale ISP architectures and

Java/390. He was involved in course development for the IBM OS/390 Web Server and z/OS WebSphere performance classes. He has authored several Redbooks™ featuring large systems and e-business topics.

Diego Cardalliaguet is an IT Specialist who works for IBM Spain with INSA-IBM Global Services. He has three years of experience in the z/OS environment. Diego holds a degree in Theoretical Physics from the University of Salamanca, Spain. His areas of expertise include WebSphere Application Server for z/OS, IBM HTTP Server, and UNIX System Services in the z/OS environment. He works for the Software Support Center in Madrid as a technical support specialist.

Russ Heald is a Senior IT Specialist who works for the IBM Software Services for WebSphere team based in Hursley, UK. He has worked for IBM for 14 years, gaining experience as a software performance analyst, a field technical sales specialist, and most recently as a services consultant. His areas of expertise include WebSphere Application Server for z/OS, CICS® Transaction Server for OS/390, CICS Transaction Gateway, and WebSphere Studio tools for z/OS services.

Tomokuni Shimizu is an IT Specialist in Japan. He has five years of experience in Advanced Technical Support (AP/ATS). He has been involved in a WebSphere Application Server project for z/OS for the last three years and is responsible for developing testing tools in IBM Japan WebSphere for the z/OS team.

Dirk Ziesemann is an IT Specialist in Germany, currently working as a field technical sales specialist for the IBM Software Group in Stuttgart. He has 13 years of technical experience with IBM. His areas of expertise include the e-business infrastructure and the WebSphere on z/OS platform.

Thanks to the following people for their contributions to this project:

Richard Conway
IBM Poughkeepsie, IBM International Technical Support OrganizationPoughkeepsie

Hilon Potter
IBM Poughkeepsie, IBM Design Center for e-transaction processing

Ivan Joslin
IBM Poughkeepsie

Thomas Hackett
IBM Poughkeepsie, Technical Marketing Support WebSphere Application Server

Michael Everett
IBM Poughkeepsie, WebSphere Application 390 Integration Test

Alfred Schwab
IBM International Technical Support Organization, Poughkeepsie Center

Viviane Anavi-Chaput
IBM Poughkeepsie, ITSO

Greg Geiselhart
IBM Poughkeepsie, ITSO

Franck Injey
IBM Poughkeepsie, ITSO

Bob St. John
IBM Poughkeepsie, WebSphere Performance

Bart Steegmans
IBM Almaden Research Center, ITSO

Vicente Suarez
IBM Hursley, EMEA WebSphere Laboratory Services

Egide Van Aerschot
IBM Montpellier, New Technologie Center

Richard Johnson, Simon Knights, and Phil Wakelin for answering CICS Transaction Gateway questions.

Curt Cotner, Sean Lee, Enzo Cialini, and Grant Hutchinson for answering DB2® Connect™ questions.

Cristina Bazal Periañez
EMEA BO, IBM Technical Support Center, Spain.

Vince Pedroza for providing the PetStore application.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

`ibm.com/redbooks/residencies.html`

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  `ibm.com/redbooks`

- Send your comments in an Internet note to:

  `redbook@us.ibm.com`

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJ Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Part 1

Integrated and multi-tier solution concepts

In Part 1, we introduce the basic architectural, organizational and technical issues involved in hybrid deployment scenarios. We provide a high level view of e-business infrastructures, and discuss the pattern approach with regard to e-business applications.

We also cover physical and logical tiers and topologies, application architecture, and packaging, and discuss handling the Web components of a complex application in a different way than the EJB components. Since we treat this topic from a zSeries perspective, we also discuss when it may valid to take apart the application and move the Web content or the Web container to a different platform than a zSeries server.

In the first two chapters, we provide a detailed view of a hybrid deployment scenario based on our findings, including security, performance, availability, transaction integrity and systems management aspects.

In Chapter 3, we cover interaction characteristics between the components of an e-business application. We mention EJB access, database access and JCA access to backend systems, also including security, availability and performance aspects.

In Chapter 4, we describe the possibilities for handling static Web content in an e-business environment.

1

Integrated and multi-tier WebSphere application deployment

Currently there are different methods in a WebSphere environment of serving static content of Web sites or complete Web applications. You can do this either on one single platform, or split up the application and deploy the parts to different platforms.

In this chapter we explain why you might want to split up the application and possibly host the Web content or the Web container on a different platform than z/OS. (Although a cross-platform scenario is not the only deployment choice for loosely coupled applications, the Model-View-Controller architecture leads to this option.)

An overview is given of possible infrastructures and topologies, as well as the application architecture and packaging.

1.1 Multi-tiered environment considerations

In the past, data centers of large enterprises grew more or less independently of the outside world. The frontends to these data centers were mainly coaxial terminals or PC application clients connected via SNA architectures. The applications either had a limited functionality, or extended efforts were necessary to administer and maintain them.

Today, the e-business world has reached most companies, and dealing with such challenges is one of the most important tasks facing enterprise IT departments.

1.1.1 Today's e-business infrastructures

Today's business needs, such as user-to-business or business-to-business relations, are forcing organizations to open up their data centers to a certain extent and connect them to the open Internet world, and new functionality must be provided to make these processes as easy as possible.

- Standards are necessary in order to plug in the different environments from vendors, business partners or customers, without additional effort.
- Enterprises have to deliver the same functions of existing enterprise information systems (EIS), in combination with new business functionality that provides services to a broad range of users.
- Pure green-screen applications are migrated to complex and high function graphical user interface applications.
- Fat PC application clients become centralized and are represented to the user via an easy-to-maintain standard Web browser.

To manage these challenges, a complete infrastructure layer is introduced which eliminates the need for maintaining all the physical terminals and PC applications. A new kind of software (called middleware) handles all the workload, data flow, and connectivity between users and the traditional backends.

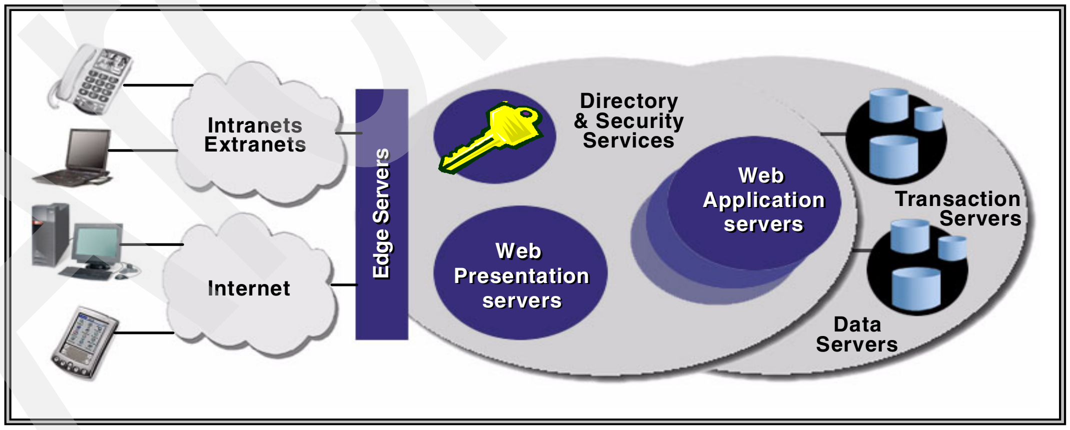


Figure 1-1 High level view of e-business infrastructures

Figure 1-1 represents a typical high level e-business infrastructure view. It shows the user tier and the backend or EIS tier, connected by middleware, the so-called middle tier. Each of these tiers handles a particular set of functions.

The middle tier handles security aspects; end-to-end connectivity between users and EIS functions; serving Web content (with Web servers like the IBM HTTP Server); providing integration business logic (with Web application servers like the WebSphere Application Server). We describe this new infrastructure layer in detail later in this book.

The EIS tier handles processing database transactions (transaction and database servers, such as IMS™, CICS and DB2).

In the last few years, e-business has become one of the most rapidly growing business areas in the world. Many enterprises have seen the possibility of reaching more potential customers and the integration of vendors and partners, as well as the need to automate their processes. But once an enterprise is opened up to the world, the IT department faces new challenges. To have e-business infrastructures enabled means:

- Unpredictable and unlimited growth
- “Peaky” and unpredictable workload
- Once established, a direct affect on other businesses and processes

Furthermore, all these challenges must be handled in the face of constantly growing pressure from competitors, higher user expectations, and tighter IT budgets. But because of the economic downturn, and because people and organizations are becoming more familiar with these technologies, e-business is now considered business as usual.

In the meantime, many enterprises have implemented architectures which rely on all the components that a robust and flexible e-business is made of. These enterprises have to reduce their costs and improve their business processes on one side, while on the other side they have to extend their reach to customers, business partners and suppliers and have to meet service level agreements.

Since this part of the outside world is unpredictable, the architecture and the infrastructure needs to be flexible, highly reusable, well performing, reliable and scalable. To help enterprises ensure the delivery of their EIS services, the Java 2 platform, Enterprise Edition (J2EE) was introduced.

The J2EE platform helps to reduce the cost and complexity of developing applications that deliver the EIS services. It is designed to provide solutions that can be rapidly deployed and easily enhanced independent of the physical platform they are running on. Further details about the J2EE architecture can be found in 1.3, “An introduction to tiers and architectures” on page 14.

Now let’s examine overall considerations to keep in mind when implementing new e-business solutions.

1.1.2 Platforms to run e-business applications

Before we discuss why you might want to handle Web components of e-business applications in different ways, let's take a look at the advantages of different platforms.

IBM offers servers for all needs and all operating system platforms:

- zSeries - offers "mainframe" qualities of service with OS/390 and z/OS, z/VM® and Linux
- pSeries™ - offers fast UNIX servers with AIX®
- iSeries™ - offers integrated business application servers with OS/400®
- xSeries® - offers Intel-based Windows and Linux servers

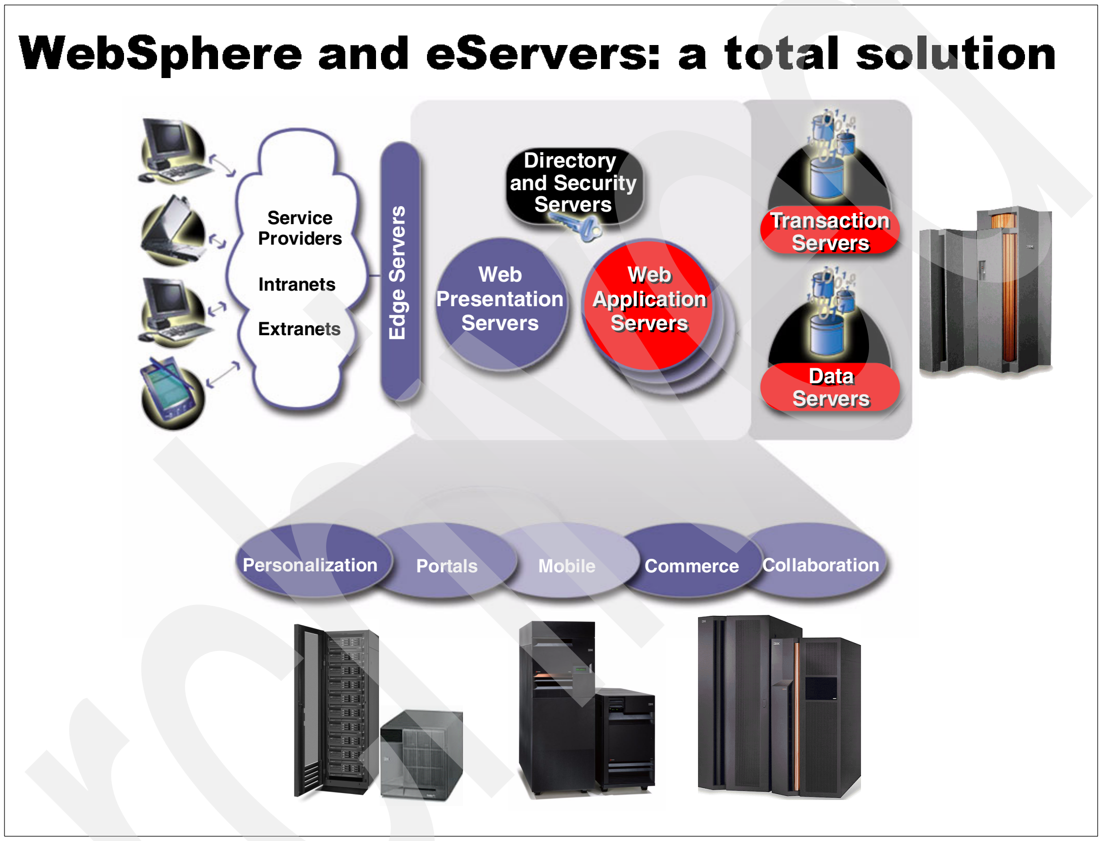


Figure 1-2 Mapping infrastructure segments to different machine architectures

As a possible environment for handling hybrid applications, we envision zSeries machines in combination with RISC or Intel servers, as represented by IBM pSeries or xSeries servers.

zSeries and z/OS

The "z" in zSeries stands for "zero downtime", which is the goal of this platform. To provide continuous availability, it differentiates from other server platforms in terms of hardware and software reliability.

zSeries server architecture relies on redundant and self-healing components to prevent any outage due to hardware failures. The z/OS operating system (like its predecessor MVS™) was designed to ensure high availability to users. It is highly integrated with its underlying hardware and takes advantage of all the features the zSeries server provides.

To obtain the highest continuous availability, many new functions were added to the operating system, such as recovery services for operating system code, address space isolation, and storage key protection. The availability of applications is ensured by functions such as

Workload Manager (WLM), Resource Recovery Services (RRS), and Automatic Restart Manager (ARM).

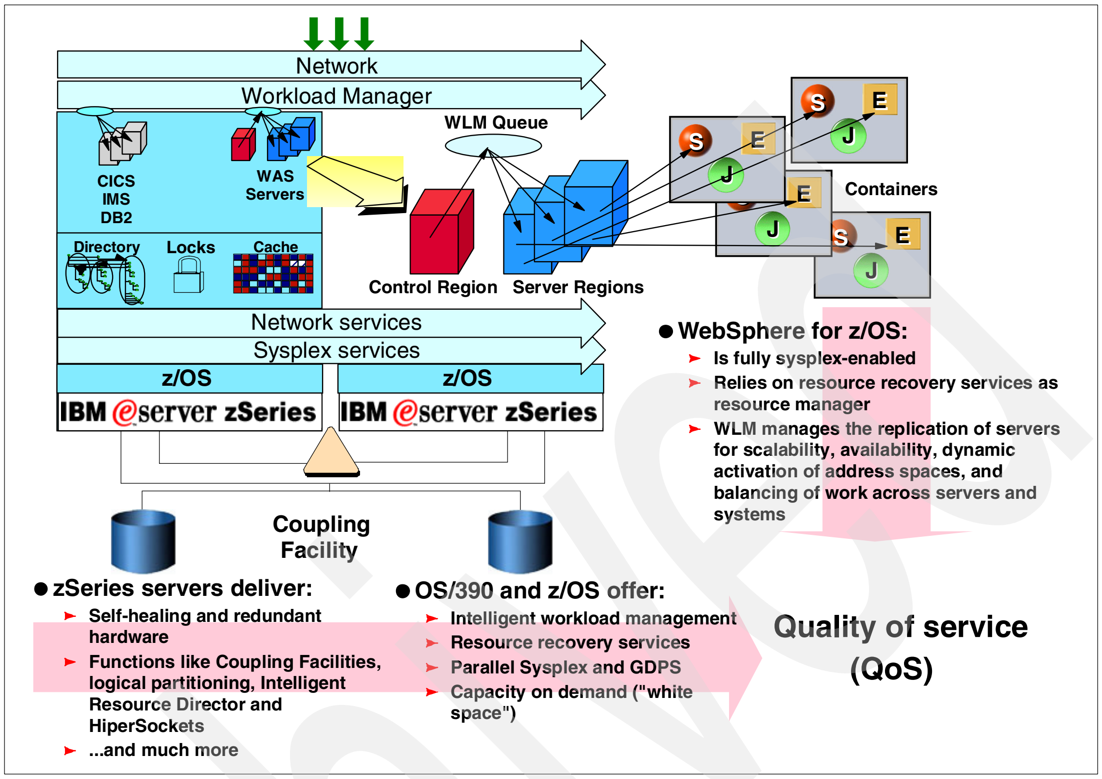

Network
Workload Manager
CICS
IMS
DB2
WAS
Servers
WLM Queue
Directory
Locks
Cache
Control Region
Server Regions
Containers
S
E
J
Network services
Sysplex services
z/OS
z/OS
IBM @server zSeries
IBM @server zSeries
Coupling
Facility
● WebSphere for z/OS:
Is fully sysplex-enabled
Relies on resource recovery services as resource manager
WLM manages the replication of servers for scalability, availability, dynamic activation of address spaces, and balancing of work across servers and systems
● zSeries servers deliver:
Self-healing and redundant hardware
Functions like Coupling Facilities, logical partitioning, Intelligent Resource Director and HiperSockets
...and much more
● OS/390 and z/OS offer:
Intelligent workload management
Resource recovery services
Parallel Sysplex and GDPS
Capacity on demand ("white space")
Quality of service
(QoS)

Figure 1-3 Advantages of the zSeries platform

The zSeries platform offers the following QoS:

- Lower total cost of ownership than competitors
- Higher availability than competitors
- Unmatched reliability
- Innovative technology (HiperSockets™, virtualization using z/VM)
- Outstanding flexibility and scalability for growth
- Simplified systems management
- System and transactional security
- Accounting

pSeries with AIX

The pSeries, with its AIX operating system, is IBM's representative in the arena of RISC processor-based systems. (Similar products are available from other vendors, and the advantages of this architecture apply to their products as well.)

The "p" in pSeries stands for "performance", which is the goal of this platform. It offers the following qualities of service:

- Extremely powerful and reliable systems
- Logical partitioning - for flexibility and scalability
- Clustering technology - for high availability solutions
- Ease of installation, maintenance, and integration
- Large number of applications available
- Widespread skill base available

As we mentioned earlier this book provides the perspective of a mainframe audience but we've seen that for several reasons it can be valid to handle specific workload on other platforms because of it's advantages.

1.1.3 Basic architectural considerations

The infrastructure consists of hardware, software, and network components selected for their ability to meet the needs of today and tomorrow. Following is an overview of general considerations:

- Scalability
  - Vertical and horizontal techniques, caching, workload segmenting
  - Coordinate work and data
- Availability
  - Eliminate single points of failure
  - Ensure redundancy and reliability
- Security
  - A rather closed system vs. complex infrastructures
- Cost aspects
  - Hardware and software cost for a given infrastructure
  - Complex infrastructure causes more infrastructure costs

In the following sections, we examine these considerations in more detail.

Scalability

Proper understanding and implementation of scalability is a key factor in improving the availability and performance of your e-business infrastructure. Scaling techniques are especially useful in multi-tier architectures when using different components in combination like edge servers, web servers, web application servers, transaction servers and databases.

Scalability can be achieved using several techniques:

- *Vertical scaling* refers to having one large machine and if necessary, utilizing more resources in the same machine. That is the case for WebSphere Application Server for z/OS when the Workload Manager (WLM) starts new server regions within the same LPAR.
- *Horizontal scaling* refers to having a number of small machines or LPARs and, if the workload requires more resources, either an additional machine will be added or WLM will spread the workload over more LPARs. (In that case, you will have one more problem to solve: the coordination of work and data across multiple servers.)
- *Caching* is also a kind of scaling technique wherein the path length of an issued request between the request time and resulting response is reduced. Different platforms provide different availability features or functions. For your application, you have to determine the probable demand to decide how scalable the infrastructure needs to be.

Availability

The most common way to achieve availability is to provide redundant components, with the objectives being to avoid single points of failure and to provide correlating resources to handle the maximum workload. Additionally, the platform itself provides individual solutions to achieve availability. Before deploying an application to a specific platform, however, you need to determine which availability requirements the platform must fulfill.

Security

The most secure system is a single and isolated small system without any connections to the outside world. Whenever you add more layers to the architecture, or implement redundancy and failover structures to the infrastructure, the complexity of handling security problems grows exponentially. The more new customers, vendors, or partners you have—each with its own environment—the more complex the security infrastructure will be.

Cost

Regarding the issue of costs, you'll face similar challenges. With an isolated small system, you'll have the lowest possible cost, but probably won't be able to fulfill any user demands. The more complex an infrastructure you are going to build, the more costs have to be calculated. Higher complexity can necessitate integration of different building blocks, specialized on certain functions, which can lead to further integration of different hardware and software components. Additional and more highly skilled staff may also be needed to implement and maintain such complex environments. And finally, there is a direct relationship between costs and scalability, availability and security.

As these examples point out, when considering the general aspects of an architecture, you'll encounter unpredictable challenges such as growth and workload. When meeting these challenges, you also must find the optimal balance between endless scalability and low response times, between availability and outages, between total security and unrestricted access to your data, and ultimately between an expensive, complex solution which will never deliver a return on investment, and a very simple solution that nobody can use.

There are, of course, many other aspects to consider, but these items provide an outer framework for all the other architectural considerations.

1.1.4 Separating Web components from business logic

Spreading an application over two different platforms is a process that requires careful planning, development and deployment. For the development process, it means that you must separate the Web components from the business logic and provide two applications. For the deployment process, it means you have to run two independent deployment processes and maintain two different runtimes.

So why would anyone want to separate Web components from the business logic? Customers cite many different reasons; for example, they may need functionality available on other platforms. They may have seen that they can handle some kind of workloads more efficiently on non-z/OS platforms. From an organizational perspective, they may have existing infrastructure to reuse, or have a dual vendor strategy that leads to two different runtimes.

In the following sections, we discuss these issues in more detail—and also examine other possibilities for implementing a separated deployment configuration, including a hybrid deployment model.

Performance

Meeting performance demands is an important criteria for successful e-business. Whenever possible, functionality to gain performance increases should be implemented as soon as possible. Depending on the type of application or the infrastructure, you will get different performance characteristics by using mechanisms like caching or offloading the application components from z/OS.

Performance considerations also take into account workload; for example, which platforms are better at handling particular kinds of workload? If an application is focused on presenting static content, the workload should be handled as close as possible to the user to reduce the pathlength of requests. It makes sense then to separate the Web application from the business logic.

If the application focuses on business logic, uses transactions, or heavily accesses databases, then these operations should take place as close as possible to the backend. Tighter backend integration ensures the exploitation of qualities of service of the zSeries platform. If the overhead of the Web content is not very significant, then there is less value in separating the application, running two deployment processes, and maintaining two runtimes. A good rule of thumb is probably the ratio of instruction consumption of the Web application and the business components.

Note: With regard to performance, keep in mind that specialized systems (caching server, load balancer, transaction server, and so on) are available for all the different runtimes. Because of this specialization, these systems are able to handle specific kind of workloads more efficiently.

There is an ongoing user demand for new functionality. If there's a demand for a piece of functionality that is not supported on the z/OS platform, you may have to offload this part of the application to another platform. (This scenario can occur when a framework is used or a customer buys "off the shelf" software, and the software or the framework exploits APIs that are not supported with WebSphere on z/OS.)

Flexibility

Separating the applications can lead to greater flexibility. The models and concepts used to run hybrid applications lead to physical and logical independence of their components (see 1.2, "Concepts and building blocks for hybrid WebSphere solutions" on page 11).

The presentation layer requires changes more often than the business logic. When the Web application runs on a separate platform, you can easily change the Web layout without affecting backend structures and without redeploying the complete application.

Cost considerations

Whenever customers deploy new applications to their zSeries platform, it may increase the workload so much that a machine upgrade will be necessary. Any processor upgrade directly affects software costs, and therefore a Total Cost of Ownership (TCO) study needs to be done.

On the other hand, the separation of Web components brings in new physical machines, and probably they have to be redundant—so what will it cost to offload Web applications to a separate platform, where new hardware and software will be needed? What does it mean to your overall z/OS IT cost to handle the workload on this platform? Regarding quality of service, you also need to determine—is it really necessary to apply to a Web application the quality of service of the zSeries platform?

Existing infrastructure or organizational requirements

Existing infrastructure or organization requirements probably exert the most influence when enterprises introduce e-business. Usually the structures in an enterprise's data center have existed for years and represent a significant investment; it would be very expensive to change everything just to deploy a new kind of application.

Hybrid deployment scenarios offer greater flexibility in terms of geographical considerations in separating data centers. The backend resides in a high security area, whereas the Web components of an application do not need such an infrastructure.

Many customers also have a organizational structure that separates their IT shop into “mainframe world” and a “distributed world” departments. And as mentioned earlier, they may also favor a dual vendor strategy: on the mainframe side, IBM zSeries is dominant, while in the distributed world, they rely on environments provided by other vendors. If applications are developed to meet this requirement, it makes it easier for the responsible departments to deploy and maintain their parts of these applications.

Security

The existing security infrastructure is also an important consideration. If an entire e-business application runs on the zSeries platform, it means that the presentation layer is also located there. This topology can result in direct access to the mainframe. Many enterprises have rules that authentication must be handled in their demilitarized zone (DMZ) environment. Furthermore, their policies do not allow any zSeries machines to be part of their DMZ. Again, this automatically means that the application is a candidate for a hybrid deployment.

Summary

These are some of the main reasons why you might consider using a hybrid application environment. Hybrid deployment scenarios offer greater flexibility in terms of geographical considerations in separating data centers. The backend resides in a high security area, whereas the Web components of an application do not need such an infrastructure.

1.2 Concepts and building blocks for hybrid WebSphere solutions

There are several strategies you can use to run specific application components on specific servers, such as caching static content, offloading static components, and offloading the Web applications to another platform. In the following sections we provide an overview of methods and concepts for developing hybrid WebSphere solutions which will help you build the infrastructure and design the application.

1.2.1 Using the Patterns approach

We follow the “Patterns for e-business” approach, which is a proven method for implementing e-business solutions through the exploitation of existing structures and the reuse of components. These structures and components are modeled as levels of the “layered asset model”, where each level builds upon the previous one; see Figure 1-4 on page 12.

We map our approach to the levels of the layered asset model. For a more detailed description of the Patterns model, refer to the IBM Redbook series *Patterns for e-business*.

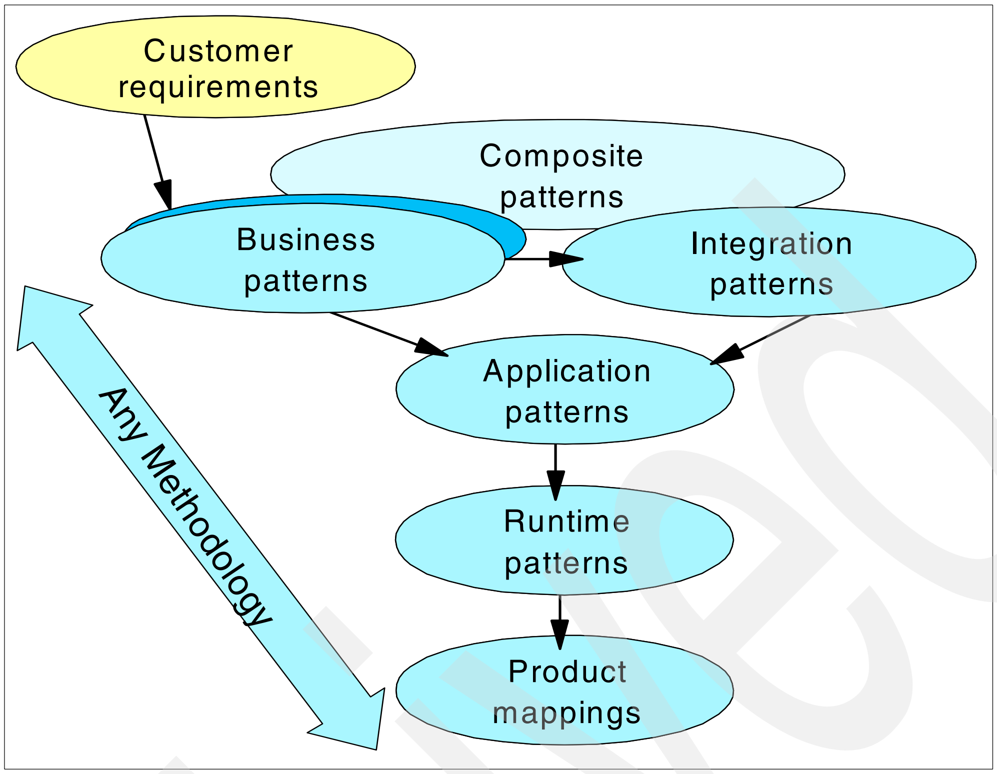


Figure 1-4 The layered asset model of Patterns for e-business

This structure is very useful as a template for a roadmap for implementing new e-business solutions. It shows how you can bind existing requirements, business processes, and environment structures together as a base for making application, runtime, and product decisions.

Note: Refer to the Patterns for e-business Web site for more detail about this structure, and to learn how to navigate easily from the top down through the layered Patterns asset model:

`http://www.ibm.com/developerworks/patterns/`

1.2.2 Mapping the patterns to our identified motivations

In 1.1.4, "Separating Web components from business logic" on page 9, we identify many organizational and operational drivers for separating Web components from business logic. Now we'll map these drivers to the Patterns model, in order to determine at which step in the process of introducing a new e-business application we will have to consider which issue. Figure 1-5 on page 13 illustrates this mapping.

Most of the drivers fit into several layers, but when moving from the top levels downward, they evolve from abstract description to specific solution. An example for this development is security; in the Business patterns, general security policies should exist—while at the Application and Runtime pattern levels, the policies become more granular with the implementation methodology.

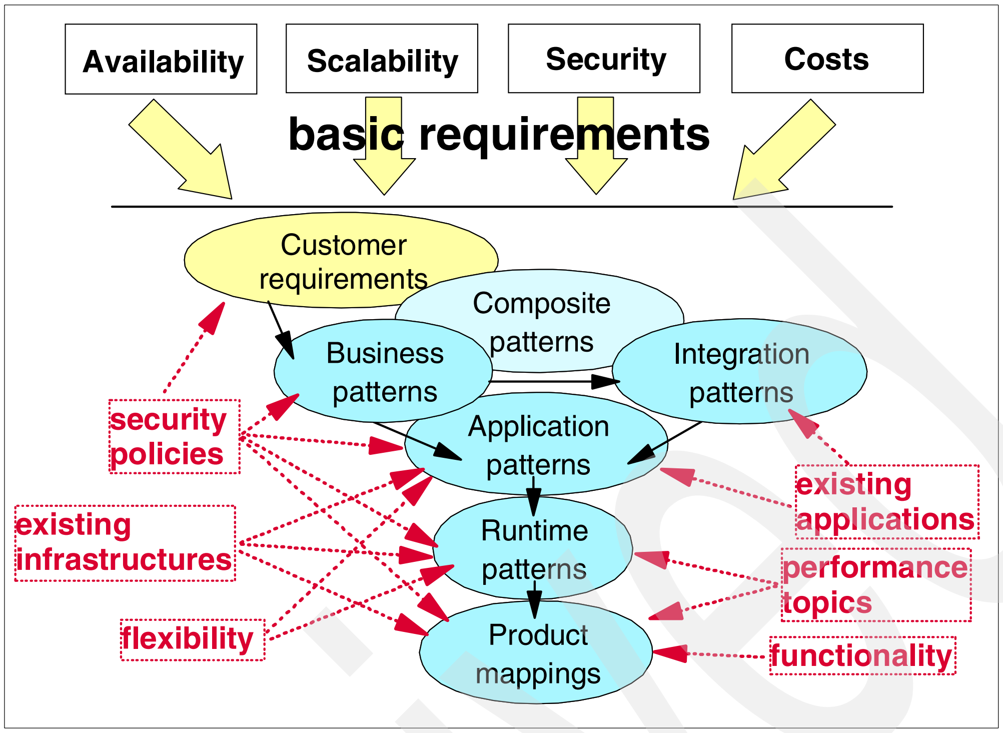


Figure 1-5 Mapping patterns with reasons for hybrid deployment scenarios

Business patterns

Business patterns reflect relationships between users, business organizations or applications, and the data to be accessed. This area is mainly influenced by existing organizational and infrastructure requirements.

For example, what kind of customer business processes will have to be handled by this application? Which service level agreements with users and partners will have to be handled? Which security policies have to be considered?

Integration patterns

Integration patterns can integrate multiple Business patterns to solve a complex business problem. They are represented by access or application integration, which means they integrate either a number of businesses through one common entry point, or concentrate multiple applications and data sources without direct user invocation.

This is where our example using existing applications like content management systems fits—can we integrate such systems seamlessly into a new application?

Composite patterns

Composite patterns are combinations of several commonly used Business and Integration patterns.

Application patterns

After identifying the Business and Integration patterns, we then define the high level logical components that make up the entire application, and how these components interact. The Application patterns split up the application into its basic components (such as presentation, application, and backend tiers).

If existing infrastructures already provide asynchronous communication, they have to be considered here (for example, when defining the method of communication between Web applications and the backend. If customers use MQseries as their common way of communication in the enterprise, then the application will exploit these capabilities, too).

Runtime patterns

Runtime patterns refine the Application patterns with more specific functions to be performed. The focus of Runtime patterns is on the logical nodes required to run these functions, and where they are placed in the overall network structure.

Currently, it is irrelevant which physical machines they exist on. In our example using existing infrastructures, the issue is how to map the identified Application pattern with existing runtime nodes in the existing network. Questions such as how to position the Application patterns to meet security policies and their physical implementation will arise here.

Product mappings

Performing product mappings is the last step in defining the network structure for an application. In this step, you correlate runtime nodes (from the previous step) with real products. This allows you to find the products that best suit your e-business application. Customer requirements such as dual vendor strategy should be addressed at this point, and performance issues can be addressed with appropriate products.

Summary

This brief excursion into the world of Patterns should help you in identifying at which step of an e-business application introduction process the decision to use a hybrid application can be made. This decision can take place at a very early stage in this process or during the last step, based on the customer's motivation.

1.3 An introduction to tiers and architectures

In the following sections we introduce the logical and physical components that comprise an e-business solution.

1.3.1 Introducing multi-tier architectures

In the past, many installations used *two-tier* application topologies; the user was directly connected to the enterprise information systems (EIS) tier that was also running the application, and the EIS services had to be delivered directly to each user.

Initially, these two-tier (also called *client-server*) application models promised functionality and scalability. However, with increasing numbers of users, the complexity of delivering services to each individual user led to major limitations.

The administrative overhead caused by installing and maintaining the business logic for each user, or managing each user access within the EIS tier, demanded practical solutions. By implementing enterprise services as multi-tier applications, these two-tier limitations can be avoided; see Figure 1-6 on page 15.

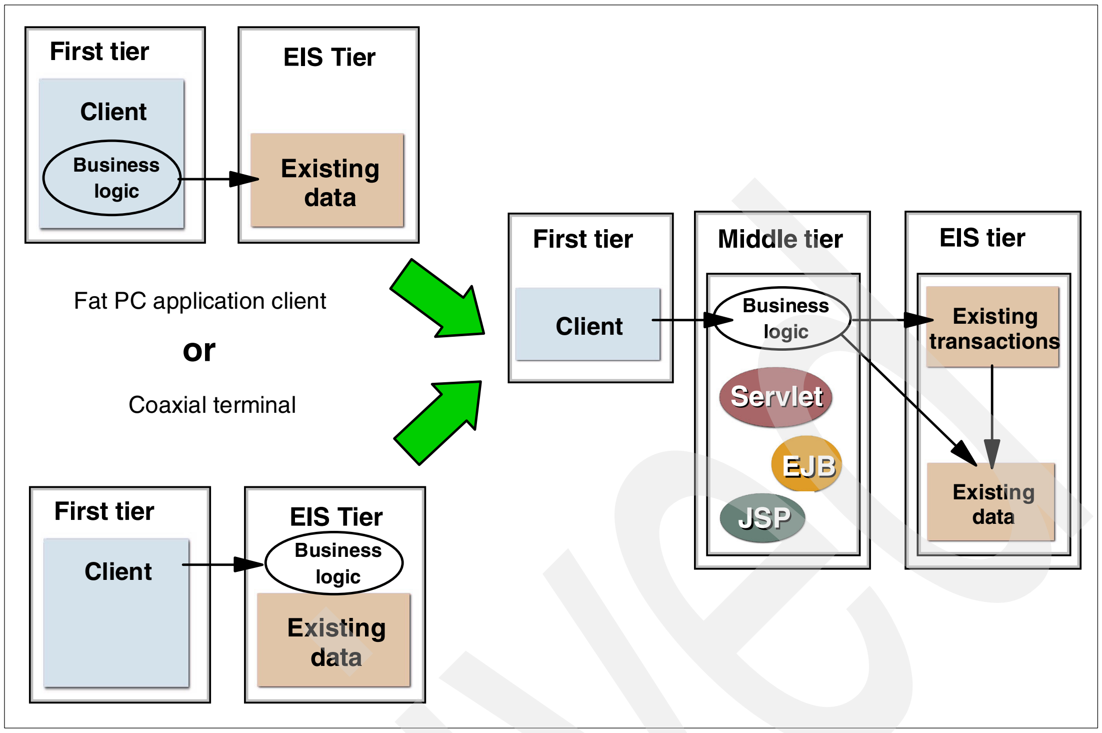


Figure 1-6 Moving from two-tier to multi-tier application models

By implementing a middle tier, you gain tremendous flexibility. The direct impact to the EIS tier is reduced by flexible and customizable middleware. The client can be reduced to a fairly maintenance-free browser interface, while the middle tier takes care of services to the user. With multi-tier architectures, you are able to deliver the necessary manageability, accessibility, and scalability.

As previously mentioned, enterprises rely on the J2EE architecture when introducing new e-business applications. J2EE is specially designed to support applications that implement enterprise services for users, customers, vendors, and business partners.

If you follow Sun's definition of the Java 2 Enterprise Edition (J2EE) application model, then you automatically implement multi-tier applications.

Note: For detailed information about Sun's definition of the J2EE architecture, refer to the Simplified Guide to the Java2 Platform, Enterprise Edition; this and other relevant documents can be found at:

`http://java.sun.com/j2ee/`

Details of the individual tiers

In its architectural blueprint, Sun identifies the following tiers (these tiers are illustrated in Figure 1-7 on page 16):

Client tier - Web client or Java client

- The Client tier can be represented by several types of clients. Many J2EE services are designed to support Web browser clients. These clients communicate with the Web application represented by servlets and JSPs via the HTTP protocol.

  The other types of clients are Java clients. They can interact directly with EJBs through Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP).

Web or Presentation tier - Servlets, JSPs, HTML pages

- The Web tier hosts static content (HTML, GIF, JPG), dynamic content (the presentation layer, JSPs) and controller functions (servlets) for the application. The presentation of data to the user can be done in different ways. In a Web-based presentation, this tier is responsible for formatting the output of the EJB tier to a format the client is able to read.

EJB tier - business logic and data access

- The EJB tier hosts the business logic of the entire application. It contains all the logic necessary to perform all the application's functions.

EIS tier - databases and existing transaction systems

- The EIS contains databases and data used and produced by the application. The transaction systems that are accessed by the EJB tier are also located here.

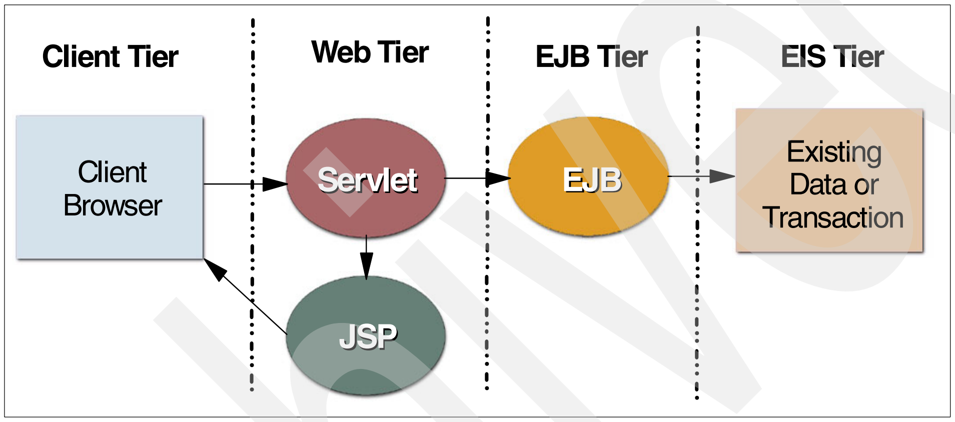


Figure 1-7 Tiers in a multi-tiered architecture

Keep in mind that even though four tiers are listed, you do not need to use or implement all of them. However, you should understand the function of these tiers, and their relationships, in order to design a proper application and its corresponding infrastructure.

1.3.2 Multiple logical and physical tiers

Tiers cannot exist alone in an IT infrastructure; you need to map the individual tiers to e-business architectures. These are separated into logical and physical architectures using multiple-tier topologies.

Why use multiple tiers

Many new e-business applications are designed by following the model-view-controller (MVC) design pattern. This pattern separates an application logically into three parts (the model, the view, and the controller), which represent different functionalities within this application:

- The model represents the business logic.
- The view is the page constructor to present data to users.
- The controller is responsible for all interactions.

This flexible application design represents multiple logical tiers. It is now a basic requirement for separating applications on physical tiers, as well. For more information about the MVC pattern, see 1.4, "Application architecture and packaging" on page 25.

This design pattern for the application offers several advantages:

- Looser coupling of application components

  - Easier reuse of components
  - Easier to change parts of the application
  - Separated and more granular defined responsibilities
- Different components can be deployed to different tiers
  - Platform flexibility
  - Different qualities of service requirements can be met
  - Tiers can scale individually (horizontally and vertically)

Remember, however, that using the MVC pattern does not necessarily lead to a hybrid deployment scenario. It simply provides the flexibility to decide on different runtime nodes, based on given requirements.

Logical tiers

The Java 2 Enterprise Edition reference architecture is a three-tier architecture (or four-tier, when including the client tier). Note that this is always a *logical* architecture, and does not tell you anything about the physical implementation of the application.

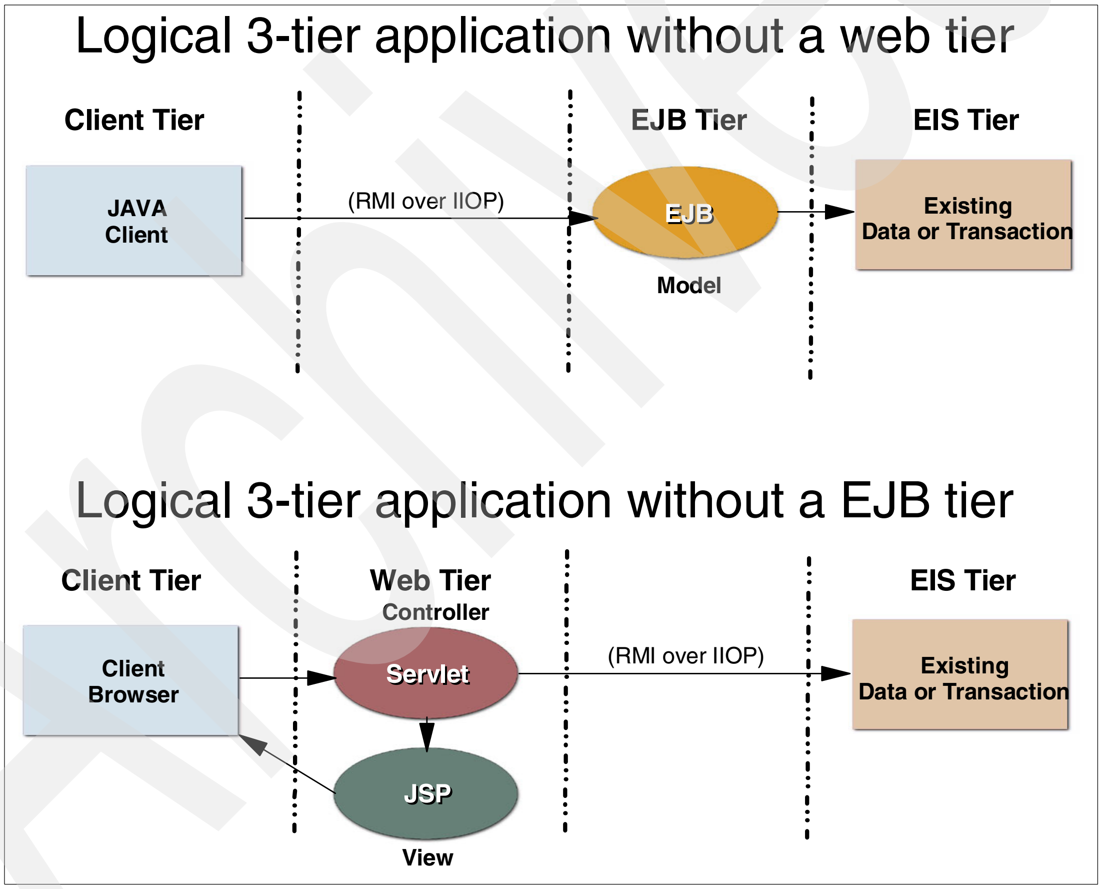

Logical 3-tier application without a web tier
Client Tier
EJB Tier
EIS Tier
JAVA Client
(RMI over IIOP)
EJB
Model
Existing Data or Transaction
Logical 3-tier application without a EJB tier
Client Tier
Web Tier
Controller
EIS Tier
Client Browser
Servlet
(RMI over IIOP)
Existing Data or Transaction
JSP
View

Figure 1-8 Applications with different numbers of logical tiers

Note: We describe this as a *three-tier* application because we count the client tier in our examples of logical architectures. We apply this method to all of the following examples and discussions, as well.

When navigating from the top down through the pattern model, you first make decisions about the logical structure of an application. By the time you reach the end of the Application pattern, the logical layout is fixed.

The decisions you make should be based on business needs that you previously defined. The dotted lines shown in Figure 1-7 on page 16 illustrate that tiers can be combined, or even skipped; note the following examples:

- If it is impossible or unnecessary to implement a Web tier because of existing requirements, be aware that clients can access the business logic directly.
- You also do not need the EJB tier with the business logic. If the purpose of an application is to present database content only, be aware that the Web tier can handle this by itself.

Physical tiers

After you decide on the logical architecture, you can then plan the physical implementation. There is great variety and flexibility available when mapping logical architectures to physical topologies; again, this task depends on existing business needs and infrastructures.

In our case, we assume that a four-tier logical architecture exists, including a Web tier, an EJB tier, and an EIS tier. Several constellations are possible, from a four-tier topology, (where all four tiers will run separated on different servers) to a one-tier topology (where all tiers will be combined on one server); see Figure 1-9.

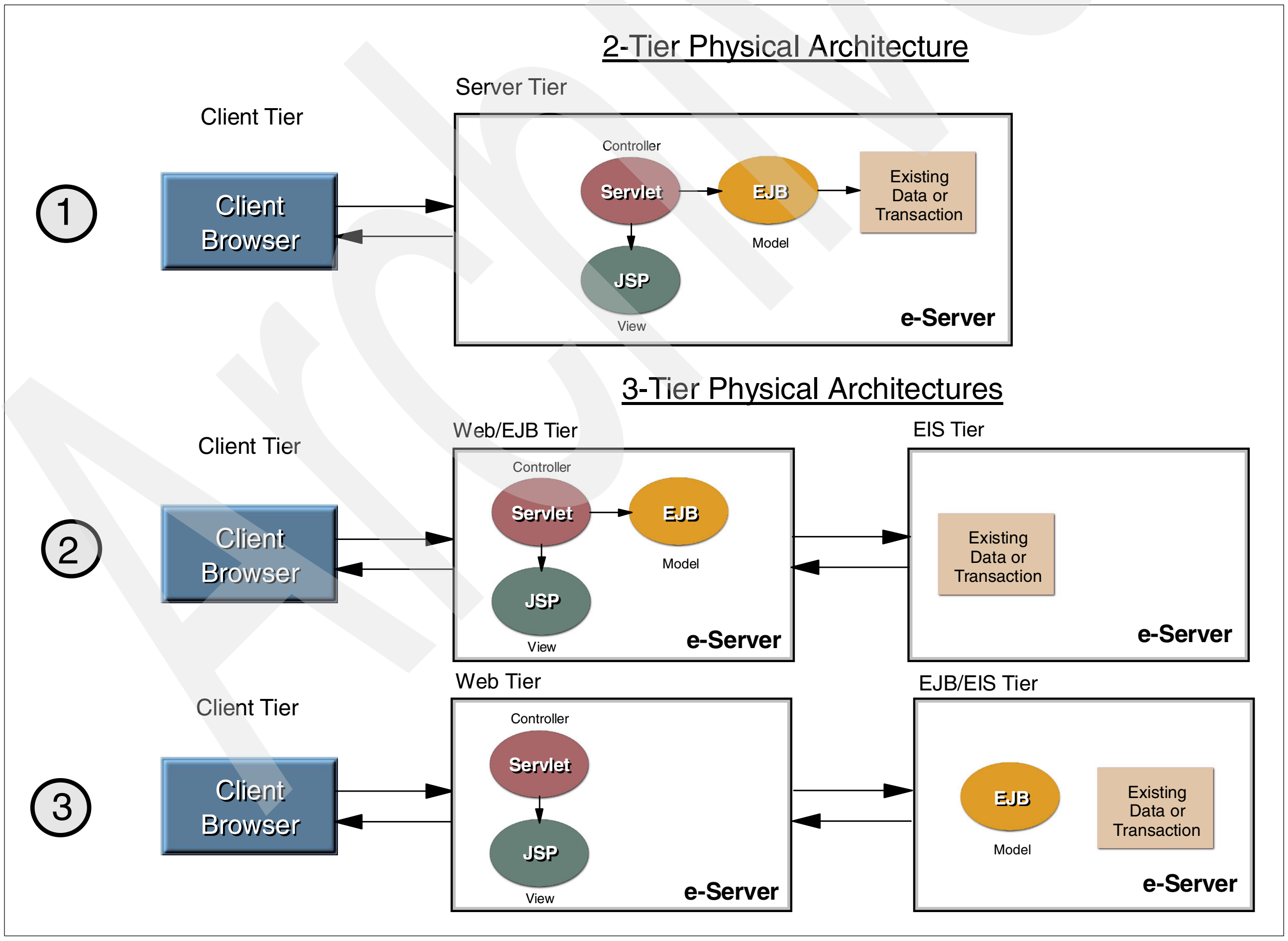


Figure 1-9 Mapping of logical J2EE components to physical tiers

Figure 1-9 illustrates the following three configurations:

Two-tier architecture

1. This is the simplest architecture. When dealing with static content, pages, and pictures, it is the easiest to implement and support. In a two-tier architecture for dynamic e-business applications, the Web, EJB and EIS logical tiers reside on the same server.

   This is the way WebSphere for z/OS applications are usually designed. This two-tier architecture is almost never used on other platforms because only the WebSphere Application Server for z/OS can reside on two images in a Parallel Sysplex®. This model describes the overall structure of an integrated solution as an alternative to a hybrid solution.

Three-tier architecture

In a three-tier architecture, there are at least two options:

2. Place the Web and EJB components on the same server, as a middle tier. The EIS component would then reside on a third (or backend) tier/server.
3. Place the Web tier (servlets and JSPs) on the middle tier and the EJB and EIS tier on the backend server. This option isolates the presentation layer from the business logic and data access. This solution is sometimes referred to as a *distributed application model*; we refer to it as a *hybrid deployment scenario* or *hybrid solution*.

When we map these options to our deployment scenarios, we are led to two different physical architectures: the integrated deployment scenario with a physical two-tier topology, and the hybrid solution with a physical three-tier topology. Both solutions represent a logical three-tier architecture.

Possible topologies with logical and physical tiers

When developing and implementing applications, the choice in most cases is between two-tier and three-tier architectures. Multi-tier e-business infrastructures provide opportunities for improved flexibility, scalability, and performance. Unfortunately, along with these advantages come increased complexities and challenges.

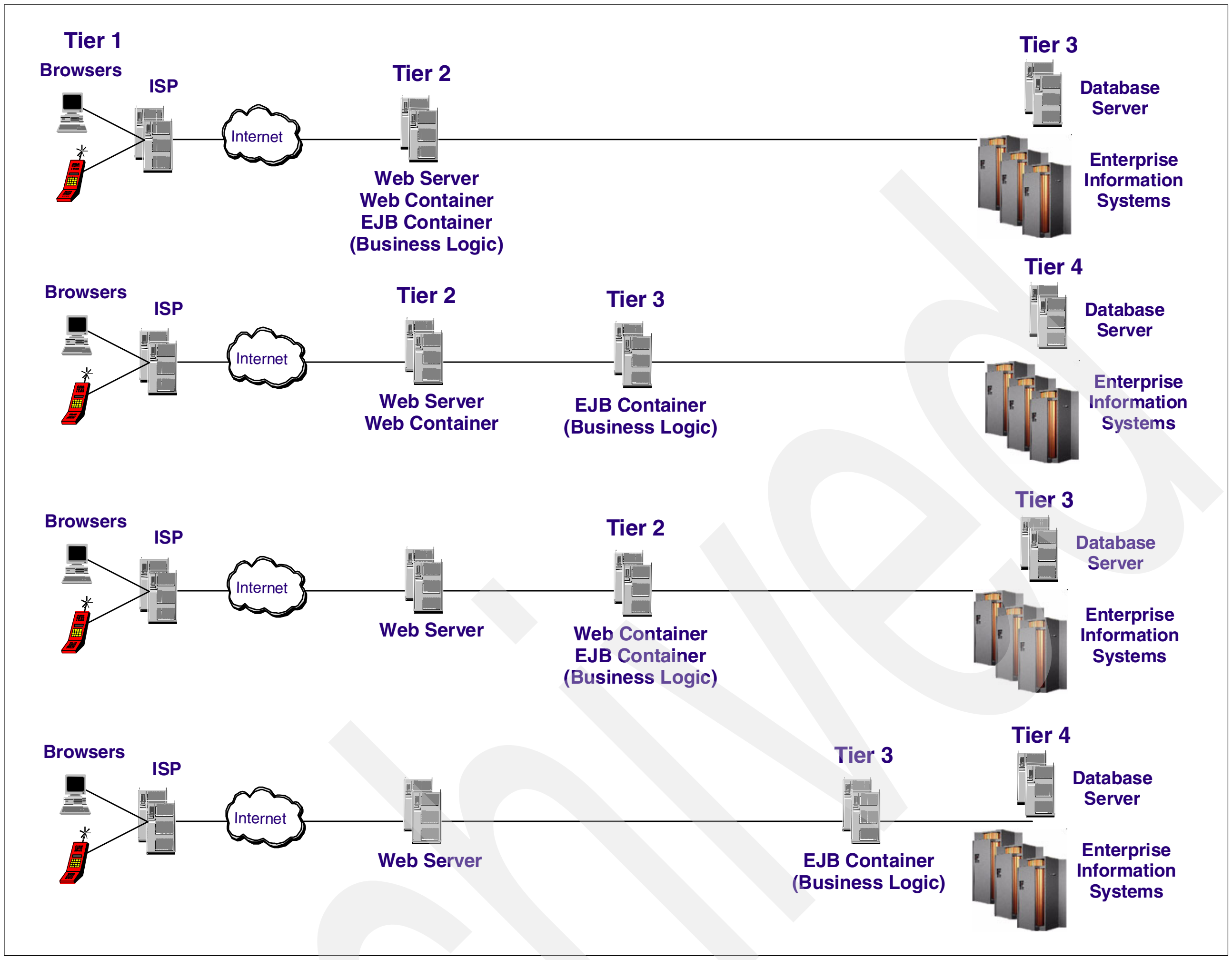


Figure 1-10 Mapping multiple logical and physical tiers

Most modern application architectures require flexibility and thus are usually divided into logical layers for presentation logic, business logic, and data serving. The infrastructure of most large sites is comprised of two or more physical tiers, with application function distributed among participating servers; see Figure 1-10.

Depending on the workload pattern, presentation and business logic may co-reside in one tier, with data serving in a separate tier—or, each may have its own tier, thus creating a three-tier infrastructure.

In some cases, it may make sense to consolidate all logical layers into a single large system complex. At the highest level, this physical two-tier infrastructure is simple, performs well, and is the easiest to maintain. In many two-tier implementations, customers place both presentation logic and business logic on the backend tier.

In this implementation, there is usually an HTTP server in front of the WebSphere Application Server. You can separate out the HTTP server (Web server) and locate it within an untrusted network. The application server has to be placed behind a second firewall. This allows you to integrate into a secure infrastructure to secure the business logic and data.

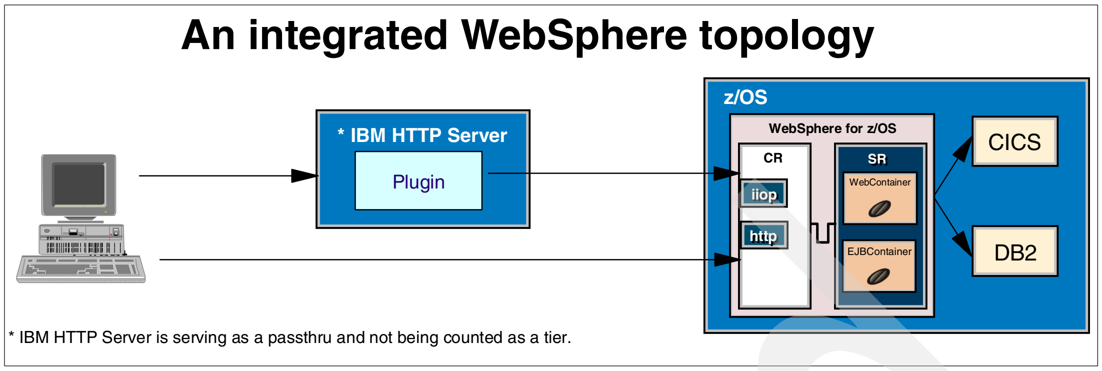

An integrated WebSphere topology
* IBM HTTP Server
Plugin
z/OS
WebSphere for z/OS
CR
SR
iiop
http
WebContainer
EJBContainer
CICS
DB2
* IBM HTTP Server is serving as a passthru and not being counted as a tier.

Figure 1-11   Physical two-tier implementation with WebSphere for z/OS

The three-tier architecture is the most commonly used infrastructure. However, as each tier is added, there are scalability and performance considerations. In most cases the WebSphere application is used to access existing data or transactions on a production z/OS system.

In our configuration, the middle-tier server is a WebSphere Application Server on a non-zSeries platform; refer to Figure 1-12. (Of course, this tier could reside on an additional z/OS system or on a Linux on zSeries system.)

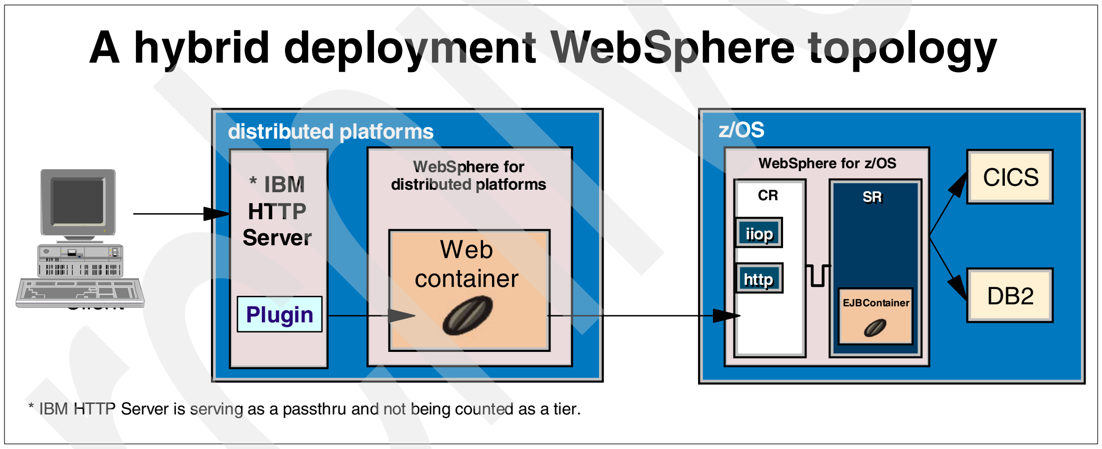

A hybrid deployment WebSphere topology
distributed platforms
* IBM HTTP Server
Plugin
WebSphere for distributed platforms
Web container
z/OS
WebSphere for z/OS
CR
SR
iiop
http
EJBContainer
CICS
DB2
* IBM HTTP Server is serving as a passthru and not being counted as a tier.

Figure 1-12   Physical three-tier implementation with WebSphere for z/OS and WebSphere for distributed platforms

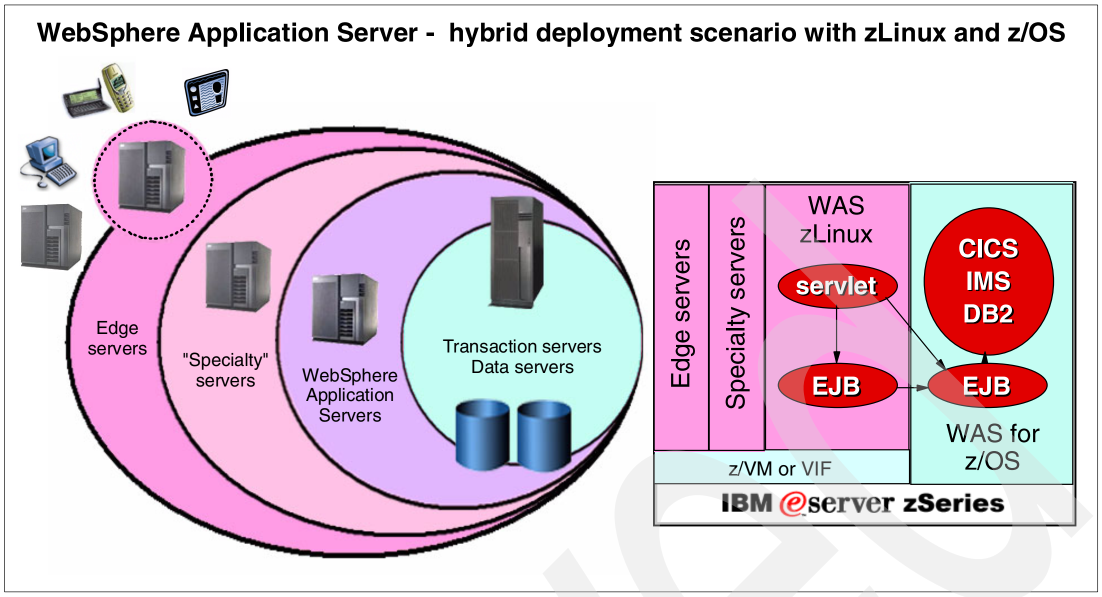

WebSphere Application Server - hybrid deployment scenario with zLinux and z/OS
Edge servers
"Specialty" servers
WebSphere Application Servers
Transaction servers Data servers
Edge servers
Specialty servers
WAS zLinux
servlet
EJB
CICS IMS DB2
EJB
WAS for z/OS
z/VM or VIF
IBM eserver zSeries

Figure 1-13 Three-tier physical implementation using WAS Linux on zSeries and WAS z/OS

- Balance Java processing and business logic between the Linux on zSeries image and z/OS
  - Use servlets or EJB session beans on Linux on zSeries
  - Use z/OS CMP/BMP EJB entity beans, persisting data to z/OS
- Integration considerations
  - It is possible to coordinate transactions with CICS and IMS
  - Security integration is possible
- Topology considerations
  - Interaction between the servlet/EJB and z/OS is a network call (that is, multi-tier topology in one physical box); however, the zSeries functionality of HiperSockets can be used

If a z/OS system is used for both the middle tier and backend tier and they are both in the same Parallel Sysplex or even in the same LPAR, then you enjoy benefits such as better security, support for two-phase commit, and higher quality of service (QoS).

The security model in this architecture is more in line with a traditional z/OS 3270 application. The identity that the request is executing under can flow with the request from the WebSphere application to the backend CICS, IMS, or DB2 without having to pass the userid/password in the “conspec”. The request is running under one security context[1]. The application itself does not need to take care of security constraints.

The overall security implications are identical to the two-tier architecture in terms of initial authentication and authorization. The main difference with a three-tier architecture is that you have to consider an additional connection between the second and third tier. This applies especially when the presentation tier is located on a non-z/OS platform. You have to carry security identities over the network.

[1] A structure or an object within the operating system or application server which encapsulates the shared security state between two entities.

Keeping the business tier on zSeries platform

In a scenario where an application needs to access CICS or IMS transactions or databases on z/OS, it seems appropriate to keep the EJB tier on the zSeries platform for the following reasons:

- The EJB container is closer to the data, which might already be on z/OS.
- A well-performing and reliable local two-phase commit can be used.
- The EJBs are using CICS or IMS connectors, which are available as local connectors.
- The z/OS sysplex can provide scalability modifications (new engine, new machine in the sysplex, and so on) without a service interruption.
- Due to smart workload balancing and white space computing capabilities, peaks in the workload are well managed in the z/OS system.

Therefore, it is especially rewarding for the business tier to reside on a z/OS platform, while it might be acceptable to separate the presentation tier on a distributed system.

The optimal solution for your site depends on your workload characteristics and application environment. For example, a publishing site such as weather.com has a great deal of presentation logic and data, but not much business logic. In contrast, an online trading site typically has moderate presentation logic and a great deal of business and data logic. The flexibility of having multiple tiers and multiple servers creates opportunities, challenges, and complexities. WebSphere is very well suited to provide the flexibility and robustness needed for these multi-tier alternatives.

1.3.3 The network layer

When discussing environmental considerations, you also have to examine the network structure itself. These considerations are part of the discussion about Runtime patterns as well.

The network represents the boundary of an enterprise to the Internet, and is therefore exposed first to any influence from the untrusted outside world. Most network security policies in an enterprise are created to ensure protection against network attacks from outside; in most cases, infrastructures already exist and they must be considered when you implement applications.

Within a typical Runtime pattern, the logical network structure is divided into three segments: the outside world, the Demilitarized Zone (DMZ), and the internal network. We will concentrate on the DMZ since this is the part of the network structure where newly developed e-business applications have to plug in.

The Demilitarized Zone (DMZ)

The DMZ is the second segment within the network tier. It is located between the outside world and the internal network. It is the first zone within an enterprise network that can be accessed from outside, and it is therefore exposed to possible attacks.

The DMZ is separated by firewalls from the outside world and the internal network. These firewalls should only have open approved ports between the different tiers in the network. Since the DMZ is considered a high-risk zone for attacks, it should not contain any business logic, user data, or security information. It must be considered as the first barrier to an enterprise network, and as a Denial of Service (DoS) buffer to protect internal data.

A DMZ topology fits into our integrated or hybrid environment considerations. When discussing performance increases, the DMZ should be considered as a place for caching the static and dynamic pages of a Web application.

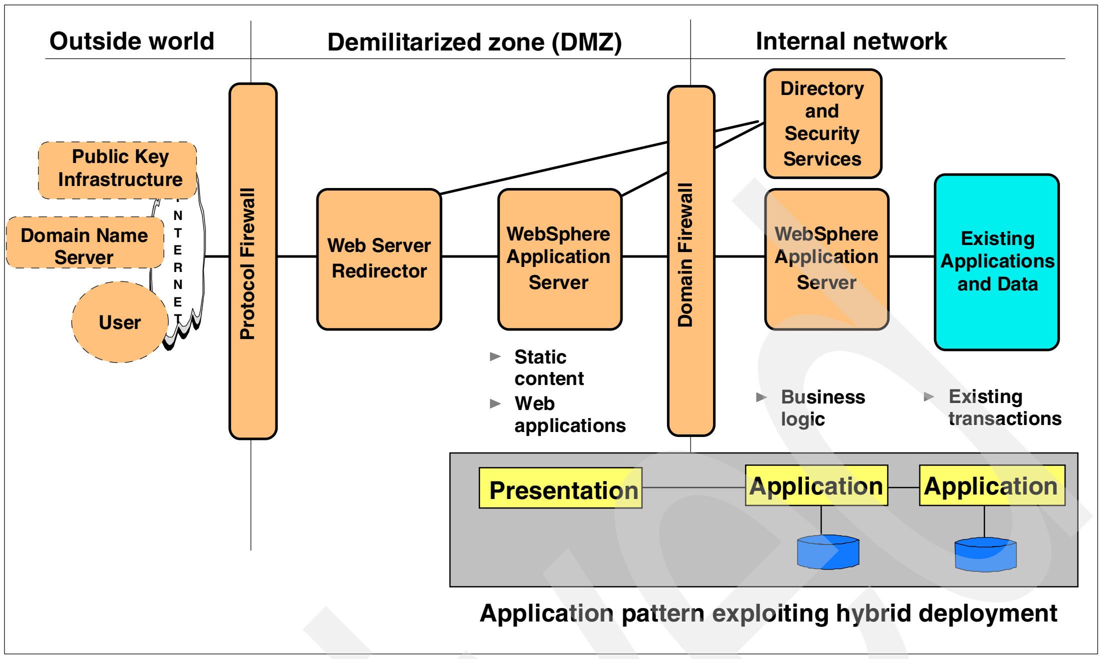


Figure 1-14 A typical network structure and its components mapped to an Application pattern

A general recommendation about where to cache static or dynamic Web content is to keep the data as close as possible to the user to reduce the path length of requests. However, you also you have to reach as many users as possible with this approach. Since the DMZ is the point where all external users access the enterprise network first, it satisfies both requirements.

Caching in the demilitarized zone has the following effects:

- It reduces the path length for requests, while minimizing network routes that must be passed.
- It reduces the response time for client requests.
- It reduces the internal network load and bandwidth requirements between runtime nodes.
- It releases internal resources and achieves better scalability.

For an integrated solution, it means that the static content can be served by a Web server in the DMZ. Furthermore, security processes can be handled in the demilitarized zone while intercepting requests for authentication purposes.

For our hybrid environment, it means the DMZ can serve the static and dynamic pages as well as the entire Web application of our solution. In this case we need an additional Web application server installed in the DMZ.

The scenario shown in Figure 1-14 is applicable for business patterns like Business-to-Customer and Business-to-Business, where the EJB tier of the enterprise is accessed from the Internet.

Internal network topology

You can also implement applications to handle enterprise internal business processes (backend access does not have to take place only from the outside world).

Firewalls are normally not utilized between users and the backend tier. Of course, individuals users must identify themselves to the application, but the internal network is considered a safe zone without the need for any additional zones to defend against attacks.

It is even easier to handle hybrid applications in the intranet because you have fewer security constraints in terms of keeping internal data. Figure 1-15 shows that the same hardware and software runtime components can be used for both topologies.

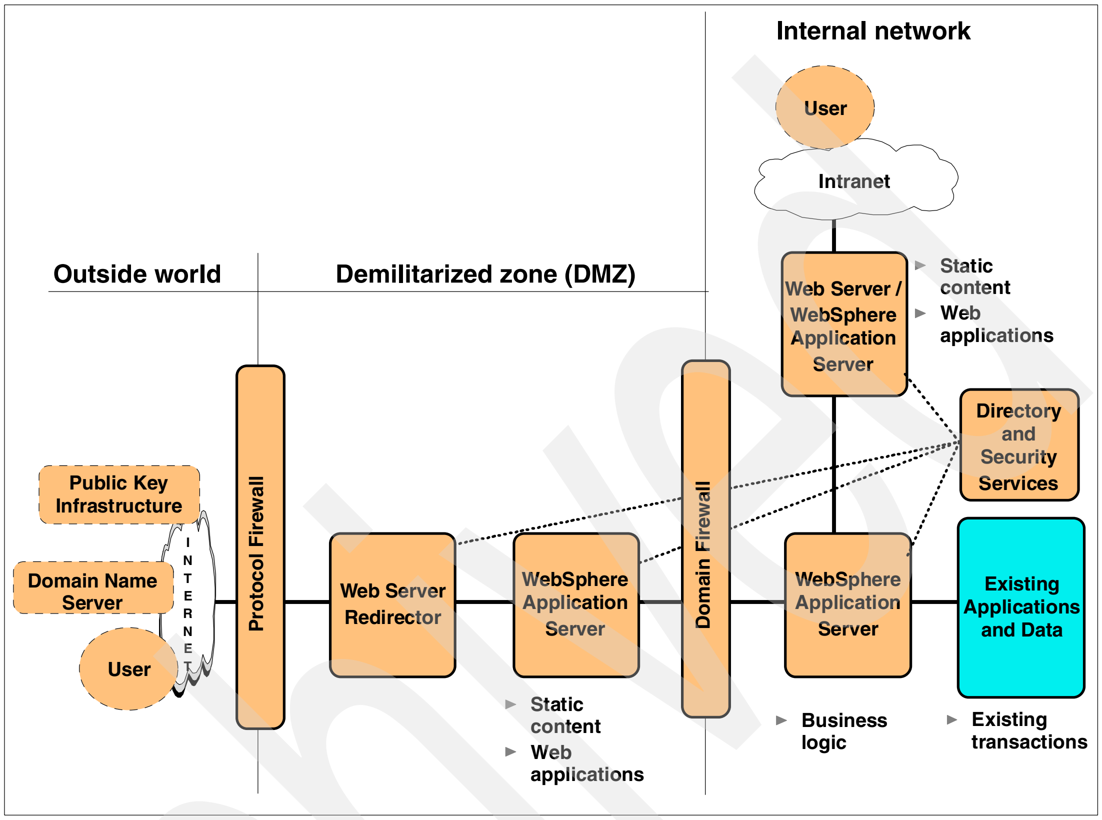


Figure 1-15 Combined network structure for backend access from Intranet and Internet

Note: To improve internal network performance, you need to integrate the same special servers (such as load balancer or caching proxies). All the caching concepts introduced previously with the DMZ apply to the internal network, as well.

1.4 Application architecture and packaging

Earlier, we described how to proceed from a business issue to a physical implementation of a new e-business solution. Now we can take a closer look at the application itself and see how it fits into that process.

When we talk about developing e-business applications, in most cases we follow the J2EE application programming model, a standard programming model for developing multi-tier, thin-client applications. The J2EE architecture is designed to support applications that provide enterprise services to customers, suppliers, business partners or just simply users. Such applications are normally very complex, realizing data access from a variety of sources and distributing services to a variety of clients.

For better management of these applications, the business functions to support the various users are conducted in the middle tier. Figure 1-16 on page 26 illustrates a sample J2EE application and the interaction between its components.

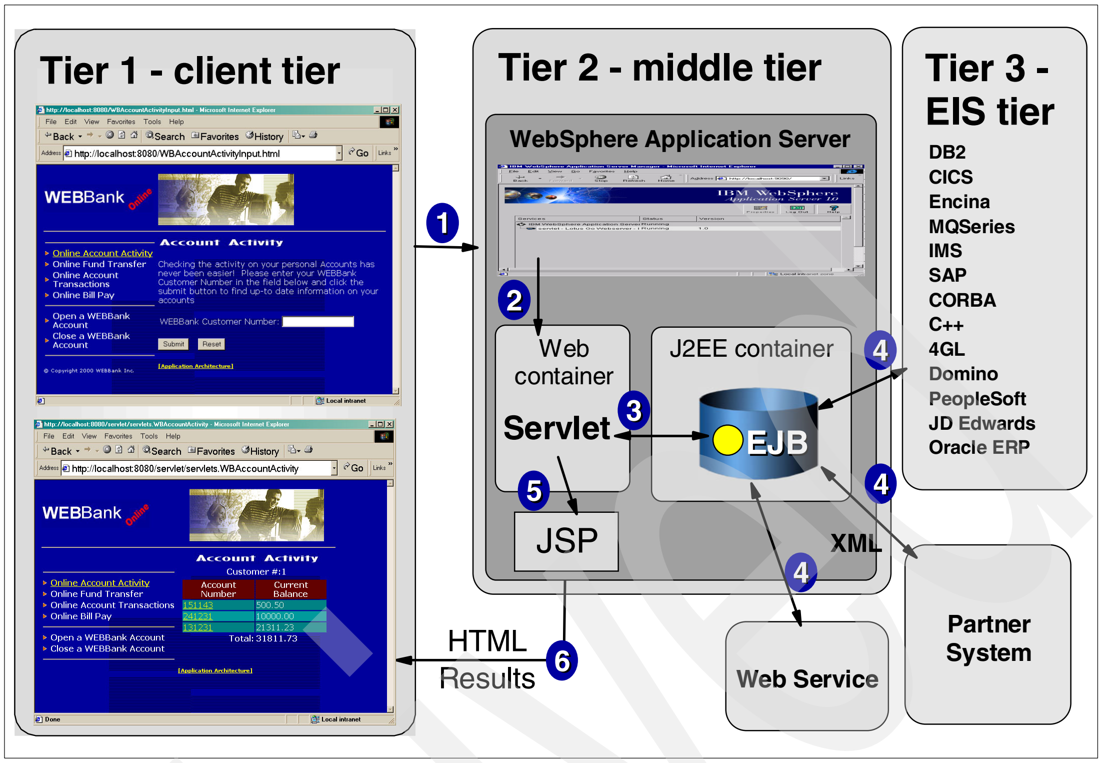


Figure 1-16 Typical architecture of a J2EE application

The flow is as follows:

1. Clients send their requests to the middle tier.
2. The WebSphere Application Server receives the requests in the Web container, where they will be handled by a servlet which acts as an interaction controller.
3. The servlet forwards this request to the appropriate EJB in the J2EE container.
4. The business logic processes the requests and sends back the request, with the retrieved data, to the servlet.
5. The servlet passes the request forward to a JSP, which builds an HTML page.
6. The JSP presents the results as an HTML page to the client.

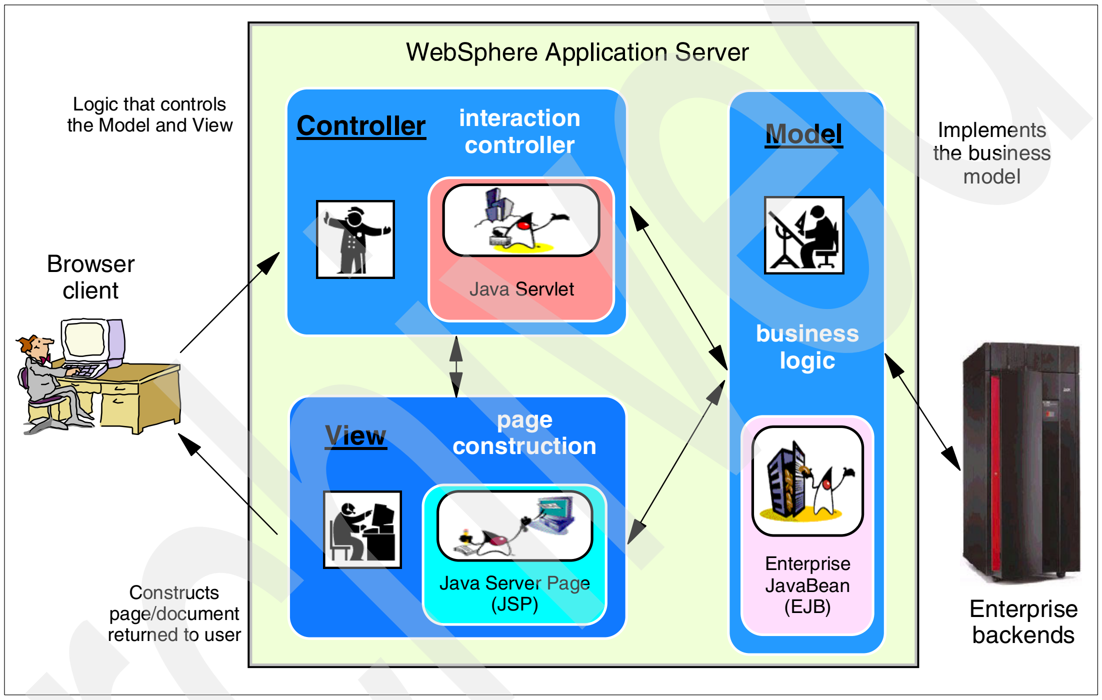
Most application designs follow certain documented patterns that have been proven in many successful installations. Over time, the Model-View-Controller (MVC) design pattern has become established as a powerful and well-tested design pattern for graphical user interface (GUI) client/server applications. (It is quite similar to the previously introduced J2EE architecture because it follows the J2EE application programming model.)

But isn't the J2EE architecture itself already a framework to solve complex e-business application development problems? The answer is yes, in a certain way, it is. But the MVC pattern refines the structures of this architecture even more and gives it a greater flexibility. Many development organizations have successfully applied the mapping of J2EE APIs to the roles in the MVC pattern.

1.4.1 Model-View-Controller (MVC) design pattern

As mentioned, the MVC design pattern has developed as a useful structure for e-business applications. This pattern has three, distinct, separate forms of functionality within an application: the model, the view, and the controller.

- The business logic is the model, and it accomplishes the goal of the interaction with enterprise backends (this may be a query or an update to a database).
- The user interface logic is the view and contains the logic that is necessary to construct the presentation.
- The servlet acts as the controller and contains the logic that is necessary to process user events and to select an appropriate response.



WebSphere Application Server
Logic that controls the Model and View
Controller
interaction controller
Java Servlet
Model
Implements the business model
Browser client
business logic
View
page construction
Java Server Page (JSP)
Enterprise JavaBean (EJB)
Constructs page/document returned to user
Enterprise backends

Figure 1-17 The Model-View-Controller (MVC) design pattern

Let's look at the individual functions of the components of the MVC pattern in more detail:

- **Model** - with JavaBeans and Enterprise JavaBeans (EJBs)

  A *model* is a set of objects that represent the business logic of the application. This usually includes classes to represent business abstractions (such as accounts, purchases, and so forth), as well as real-world objects (such as employees and customers).

  The business logic has to address a broad range of requirements, such as transactional integrity, maintaining and accessing application data, and integrating new applications with existing applications.

- **View** - with Java Server Pages (JSPs)

  A *view* is a particular way of presenting a set of information to a user. The page constructor is responsible for the generation of the HTML page that will be returned to the client. Think of a view as a particular Web page or screen that displays a single set of linked data to the user.

  WebSphere supports both ways to display pages, with servlets or JSPs. JSPs allow template pages to be developed directly as HTML pages, with inserted Java scripting logic for the dynamic elements.

- **Controller** - with Java Servlets

  A *controller* is the layer in an application that handles the details of application flow and navigation. The controller ties protocol-independent business logic to a Web application, which means it maps HTTP-specific input into the input format required by the business logic. Afterwards it translates information coming back from the model layer into a form the view layer can understand.

  The key here is that the model is kept separate from the details of how the application is structured (the controller) and how the information is presented to the user (the view).

Benefits of the MVC design pattern

The most important aspect of this design pattern is the *separation* of components. This separation into parts with individual functionality leads to the following advantages:

Increased flexibility

When the components of an application are developed in modular structure, these parts can be easily deployed on different platforms. You can determine, for each part of the application, which platform delivers the qualities of service the application needs.

Increased reusability, extensibility and maintainability

In a complex application, there are usually display pages that can be called by multiple servlets. The business logic can also be reused by several Web applications and interactions.

If new application parts need to be plugged into the presentation layer, or user interfaces have to be changed, this can be done without affecting the business logic.

Can support multiple user interfaces

A complex e-business application often supports various user interfaces, like HTML-based Internet clients or thick application clients. Separating the presentation logic from the business logic allows the reuse of the business components for these different types of clients.

Leverage different sets of skills

Developing an e-business application, with all its various components, requires different skills and tools. There is a great difference in the skill sets required to design HTML pages and to code Java business logic. In order to effectively leverage these resources, the separate MVC components let them fit nicely.

MVC architecture - summary

The MVC architecture provides flexibility, reusability, extensibility and clear design roles for application components. The multi-tier design allows flexible choice of implementation technologies as well as scalability, and the modular design decomposes application functions into intuitive, loosely coupled subsystems.

Unfortunately, the MVC design, the differentiated job roles, and the resulting modular application packaging often lead to the idea that application runtimes need to be separated. Blueprints and patterns are often blindly adapted on a one-to-one basis to infrastructures—or, even worse—the infrastructures are built up accordingly.

In fact, this design does not represent a “must-use” hybrid deployment process; it is simply a possibility. The advantage of loosely coupled applications is an advantage for the development process. Decisions regarding the deployment processes and the runtime nodes are made in a different step of the navigation through the Pattern model.

1.4.2 Application packaging

A complete WebSphere application usually consists of three different modules: Web archives (.war files), Java archives (.jar files), and a resource adapter archives (.rar files); see Figure 1-18.

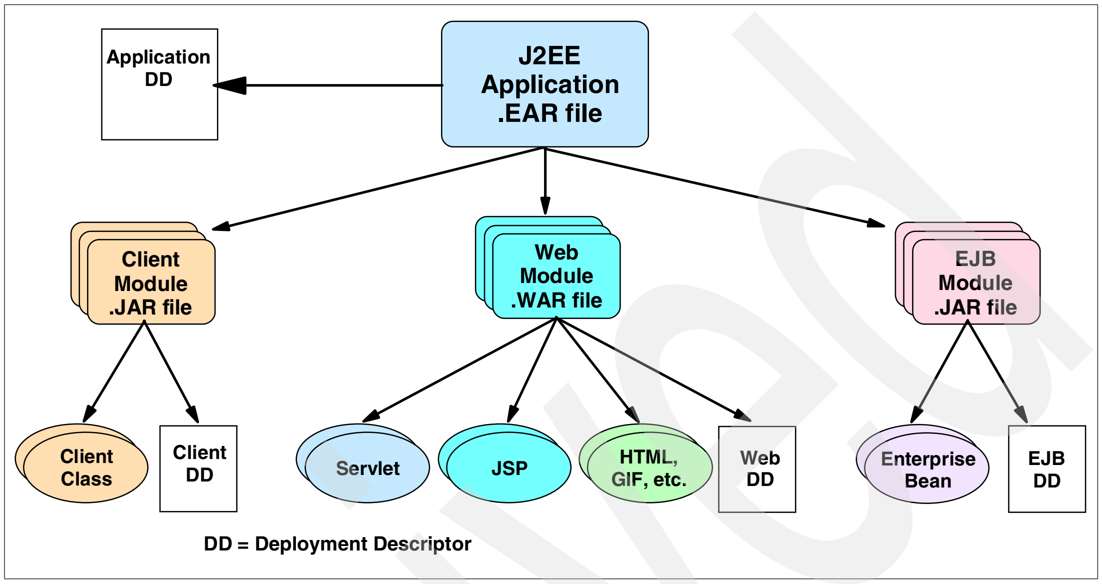


Figure 1-18 WebSphere application components and packaging

The Java archive can contain client modules with its client classes and deployment descriptors, or it can contain EJB modules with its EJBs and the appropriate deployment descriptors. The Web archive contains the presentation components of the application, the Web module with servlets, JSPs, and the static content (such as HTML pages, JPEGs, and GIFs).

These modules will be packaged, in the application assembly process, to an enterprise archive known as the .ear file. This enterprise archive is a package which is ready to be deployed into the application server.

Following the J2EE Application Programming Model, Figure 1-18 shows the components of a WebSphere application. Looking at WebSphere Application Server structure, the EJB modules will be deployed automatically into the J2EE container, and the Web modules will be deployed automatically into the Web container.

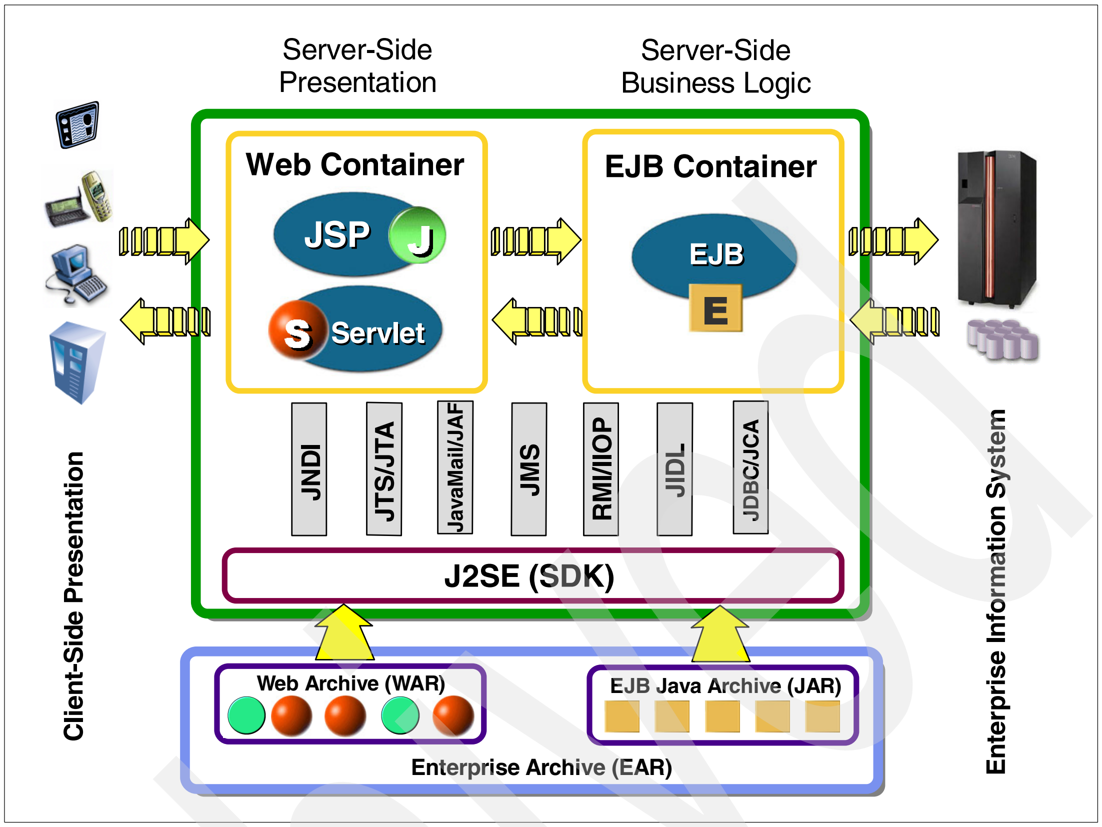


Figure 1-19 Generic WebSphere Application Server component structure

Terminology

When we refer to a "WebSphere application" or a "J2EE application", we mean all the components of an e-business application. When we refer to a "Web application", we mean the part that contains only the Web components, such as servlets, JSPs, and HTML pages.

The application assembly process and the deployment process are different for a hybrid deployment model as compared to an integrated solution:

- For an integrated solution, the application will be packaged as an entire .ear file that contains all parts of that application. The .ear file has to be provided to the application deployer, who deploys this application in one deployment process.

- In a hybrid model, different runtime nodes must be considered. That means, to the assembly process, that two different .ear files are needed: one .ear file contains the business logic, and the other .ear file contains the Web components.

  Because of the deployment to two different runtimes (maybe WebSphere z/OS for the business logic and WebSphere for Linux on zSeries for the Web components), these enterprise archives must be deployed separately, using two deployment processes.

1.5 Decision guidelines for handling Web applications

At some point, you will have collected all your business needs and requirements, and will have developed a flexible application using the MVC design pattern. Using the Pattern model, you will have now reached the phase where you will have to decide which deployment model to use.

1.5.1 Deployment choices

A WebSphere solution may call for a variety of different operational requirements. Each of the available platforms has attributes that make it unique; no architecture is the right choice 100% of the time. When considering the zSeries platform as a runtime for at least the business logic and the connected backend systems, you have a variety of development and deployment choices.

- From the development and deployment perspective, three integrated solutions are presented.
- When considering having several distributed platforms where you can deploy, more than than three hybrid solutions are available.

Table 1-1 lists high-level development and deployment choices that are based on the concepts we discussed and the available platforms. This table implies greater complexity if you choose a *hybrid* solution.

Table 1-1 High-level development and deployment choices

Application development perspective	Deployment and platform perspective
Integrated application, Web application and business logic packaged together	Integrated solution with deployment to only one platform
Hybrid application, Web application and business logic are packaged separately	Hybrid solution with two deployment processes to two different platforms

In turn, the high-level deployment choices may have further variations; these are listed in Table 1-2. As shown in this table, from the application developer's perspective, the integrated solution can be identical in all cases. Likewise, for the deployment process, an integrated solution makes no difference. However, there is increased administrative and organizational effort involved in an integrated solution.

Table 1-2 Integrated solutions

Development	Deployment	Platform and administration
Integrated application	Integrated solution	One deployment process, no additional changes
Integrated application	Integrated solution with caching on zSeries	One deployment process, appropriate setup of HTTP server environment
Integrated application	Integrated solution with off loading static content to a different platform	One deployment process, off loading of static content to a distributed world considering impacts to security,

The situation is quite different for hybrid solutions; application developers have many choices regarding how to split up the application. However, deployment efforts are doubled because of the need to deploy to two different platforms, and the administrative and organizational effort involved is also drastically increased.

We list only a few hybrid solution variations in Table 1-3, although others are undoubtedly possible.

Table 1-3 Hybrid solutions

Solution	Development	Deployment	Platform and administration
Separate deployment to two platforms using IIOP	Develop two parts of application which are loosely coupled using IIOP	Hybrid deployment to z/OS and distributed platform (Linux on z/Series/ UNIX/NT)	Two different deployment processes, considering proper environment setup
Separate deployment to two platforms using JMS	Develop two parts of application which are loosely coupled using JMS	Hybrid deployment to z/OS and distributed platform (Linux on z/Series/ UNIX/NT)	Two different deployment processes, considering proper environment setup including MQSeries®
Separate deployment to two platforms using WebServices	Develop two parts of application which are loosely coupled using WebServices	Hybrid deployment to z/OS and distributed platform (Linux on zSeries/UNIX/NT)	Two different deployment processes, considering proper environmental setup

You can also consider the scenarios listed in Table 1-2 and Table 1-3 as steps into a hybrid WebSphere environment, because the complexity grows with every step. In the following chapters, we discuss this solution in more detail.

2

Integrated and hybrid WebSphere application deployment scenarios

In this chapter, we explore the scenarios for Web component processing in more detail.

- In 2.1, “Static Web component relocation” on page 34, we describe the integrated WebSphere deployment scenario, in which all of the programmatic application components are deployed to a single J2EE application server. Optimizations to Web component processing take the form of configuration adjustments to exploit caching, and alternatives for the delivery of static HTTP content.
- In 2.2, “Dynamic component relocation” on page 36, we describe approaches to splitting the application for deployment into two separate J2EE application servers. These solutions involve making changes to the application architecture and programmed components.
- In 2.3, “Evaluation criteria for remote component and EIS access” on page 58, we discuss in detail some considerations to keep in mind when evaluating hybrid deployment options.

2.1 Static Web component relocation

Every Web application is structured in a tiered model. The presentation (or view) tier is usually made of HTML and images, regardless of whether it is directly presented or generated through a JSP. It is well worth taking the time to critically think about this content and how best to manage it.

Because static content is always present in a Web application, it is good practice to get the best possible management for it. It is very likely that this static content will be related to the look and feel of your shop, or to the presentation layer created for your products, or both. With the subsequent workload, then, requests to serve this content will be numerous and frequent.

The techniques and solutions for static content management discussed here are widely applicable, independent of other infrastructure decisions you may make about your application server architecture after reading this redbook. Static content optimization can and should be done, regardless of which of our solutions you adopt.

2.1.1 Architectural elements for static Web content acceleration

There are different ways to serve static content. The most common is to have a Web server that can work in different ways. This Web server can be embedded in another product (like the Transport Handler in WebSphere Application Server), or it can be a different product (like the IBM HTTP Server, or an equivalent).

When the Web server is embedded in the application server, you do not have many configuration options; the Web server simply accepts requests and executes them in the application server. So, the challenge is to use something more, in front of the application server, to get inbound requests and add functional elements that will help you manage all static content. In our case, we used the IBM HTTP Server.

This Web server can be configured in different working modes, as follows:

- Web content server

  In this mode, the server takes care of the static content directly; the requests are not forwarded to any other kind of server. However, it can have some added functionality through the use of scripts.

- Proxy[1] and reverse proxy[2] modes

  In this configuration, the Web server usually gets an inbound request, processes it to decide how to manage it, and sends the request to the appropriate place. Responses are also forwarded back to the client. Some added functions, like caching, may or may not be present (refer to “Web server and application server: proxy mode” on page 35 for more information).

- Web server working with an HTTP plug-in

  In this case, the inbound requests are always forwarded to another product, like an application server. The plug-in is specific for the product. Some added functions, like caching, may or may not be present (refer to “Web server and application server. Plug-in mode” on page 35 for more information).

[1] A proxy server exists between a client (application) and a server. It intercepts all requests to the real server to check if it can apply some processing at this point. Then it forwards the request to the real server. It can also be used to hide the real server’s network address.

[2] A reverse proxy server is an alternate configuration of a proxy server. It intercepts request and applies some processing. This processing could be authentication, a rewriting of the request to add supplemental information, or simply removing data to make the request more secure.

Since we intend to discuss what to do with our WebSphere Application Server for z/OS and OS/390, we will focus on specific solutions for applications working in this environment.

We do not use Web content server working mode, since we are working with an application server. The Web server has to forward inbound requests to be executed in WebSphere Application Server for z/OS and OS/390. This means that we can work with the IBM HTTP Server in proxy mode or with the plug-in.

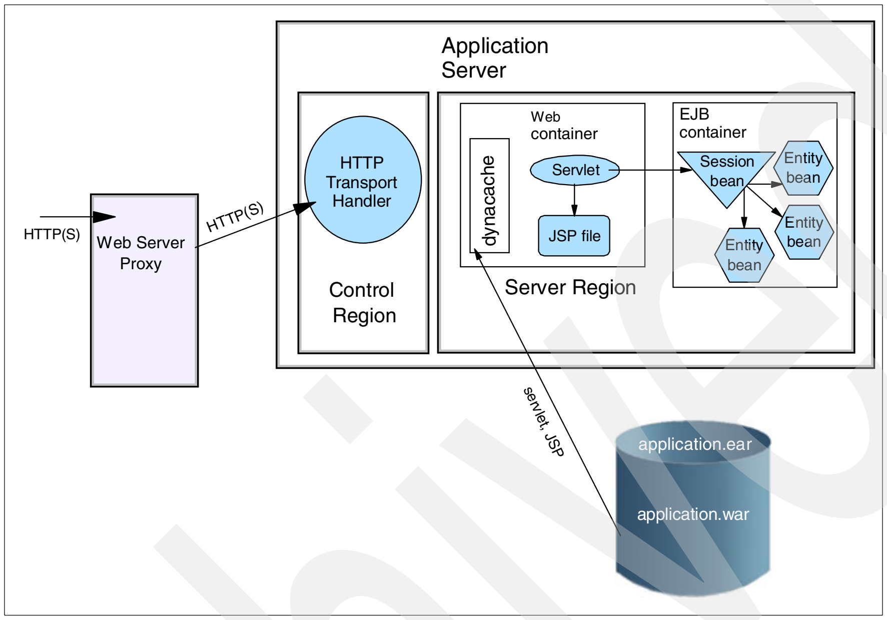


Figure 2-1 Web server and application server: proxy mode

With proxy mode, we can forward any incoming request without taking into account where it is going; if the request matches the proxy rules, it gets forwarded. Though the request gets rewritten, no other functionality is added. We may have several requests, forwarding them to many different places; it is an all-to-all relationship.

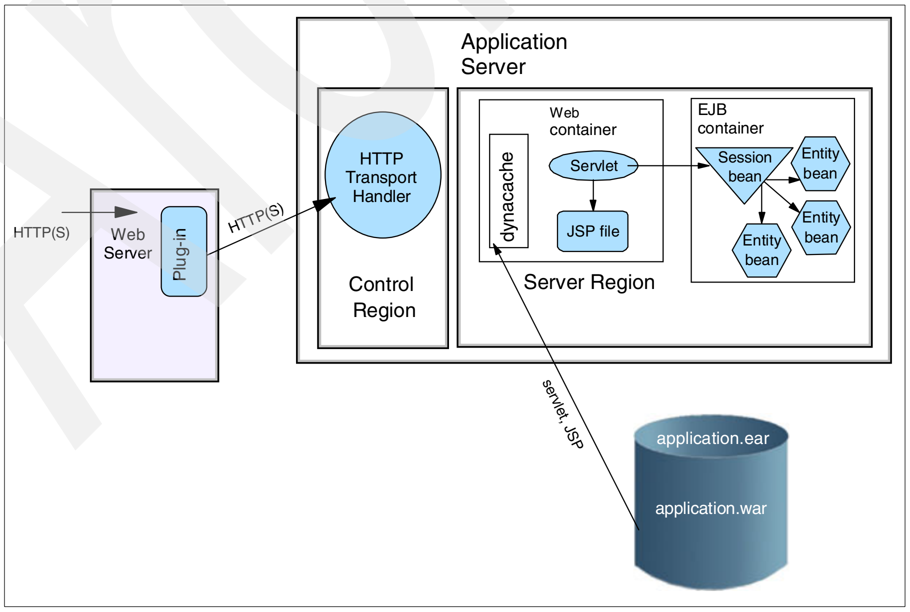


Figure 2-2 Web server and application server. Plug-in mode

If we use the plug-in option, we will probably need a dedicated Web server for specific application server inbound requests. In exchange, we get some added functionality that may be useful for our applications:

- We may have session affinity among server regions of a J2EE server.
- The Web server may be used as an endpoint for SSL in intranets.
- We can address different URIs to different server instances in the same application server.
- Other functions may also be available (refer to *WebSphere Application Server for z/OS and OS/390 v4.0.1: Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications*, SA22-7836).

For details about infrastructure and implementation, refer to Chapter 4, “Static Web component optimization” on page 101 and “Infrastructure implementation” on page 140.

2.2 Dynamic component relocation

The solutions discussed in 2.1, “Static Web component relocation” on page 34 involve no changes to the application code; all the enhancements described can be implemented using configuration parameters or the relocation of static Web content.

In a hybrid deployment configuration, however, we take the process a step further and relocate some of the *programmatic* elements of the application. The application components are divided between two different application server environments, with part of the application being deployed to one application server, and the remainder deployed to another, separate server. The components deployed within the two application servers communicate to provide the total application solution.

Note: The application may be deployed over any number of servers, but for the sake of simplicity we will restrict the discussion to a configuration consisting of two servers.

Important: Unless otherwise stated, when we refer to two separate J2EE servers, we mean two *distinct* J2EE servers, with different applications installed.

The meaning is *not* that of two cloned application servers, both running the same applications for scalability or availability reasons. We are not referring to “cloned” servers in WebSphere Application Server Advanced Edition terms, or “server instances” in WebSphere Application Server for z/OS and OS/390 terms.

In 1.1.4, “Separating Web components from business logic” on page 9, we discuss possible reasons for separating the application into a hybrid deployment. After you decide that a hybrid deployment could be a possible solution, the next step is to consider the options for the new application architecture, and assess the implications for functional and administrative capabilities.

2.2.1 Application elements

Figure 2-3 provides an overview of the various elements of the application architecture that illustrates the relationships between the various types of J2EE component, servlets, Enterprise Java Bean (EJB), and Java Server Page (JSP) files. It also shows the enterprise information systems (EIS) tier.

EIS comprises existing corporate resources which the J2EE application may need to access, such as databases, Enterprise Resource Planning (ERP) systems, or transaction processing systems.

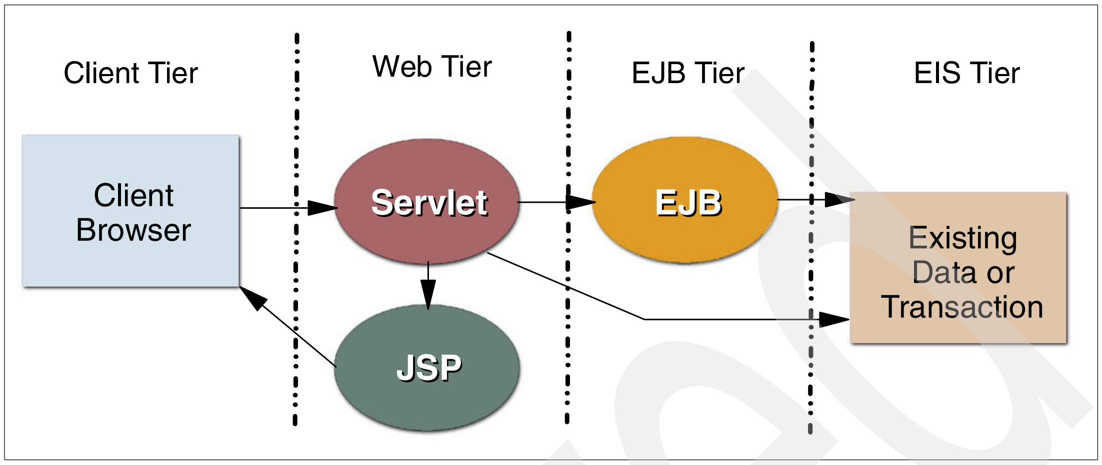


Figure 2-3 J2EE application components

This figure shows EIS being accessed from both enterprise beans and servlets. As discussed in “Web-EJB separation option” on page 43, best practices dictate that EIS access should be issued from enterprise bean components. However, there are many applications currently in production which issue calls to EIS directly from the servlet. Where this is the case, there are significant implications for moving the Web tier into a remote application server. For this reason, servlet access to EIS is included in our study.

2.2.2 Overview of hybrid deployment assessment criteria

When we deploy an application into a hybrid environment, we introduce a topological separation into the infrastructure. The application will be deployed across two platforms that require some form of network communication to interact. The presence of the network and the transformation from a local to a remote request will change some of the characteristics of the interaction, and these effects need to be carefully evaluated when considering a hybrid deployment solution.

Note: The topological separation could involve a physical separation if the application servers are running on separate platforms or physical boxes. However, it could also involve a virtual separation.

For example, if one of the servers is WebSphere Application Server for z/OS and OS/390, and the other is WebSphere Application Server Advanced Edition running on Linux for zSeries, both servers could reside on the same physical box, but there still needs to be a network connection in order for them to communicate.

Reference configuration

In order to assess the feasibility of a hybrid deployment, we need a reference or reference configuration with which to compare. The configuration we use is a theoretical one, in which all the application elements are hosted on a z/OS system.

The Web components and EJB components are packaged together in the same enterprise application, and deployed to a WebSphere Application Server for z/OS and OS/390 J2EE server. All databases and EIS resources are located on the z/OS system (which is where many EIS resources will actually be found); this is illustrated in Figure 2-4.

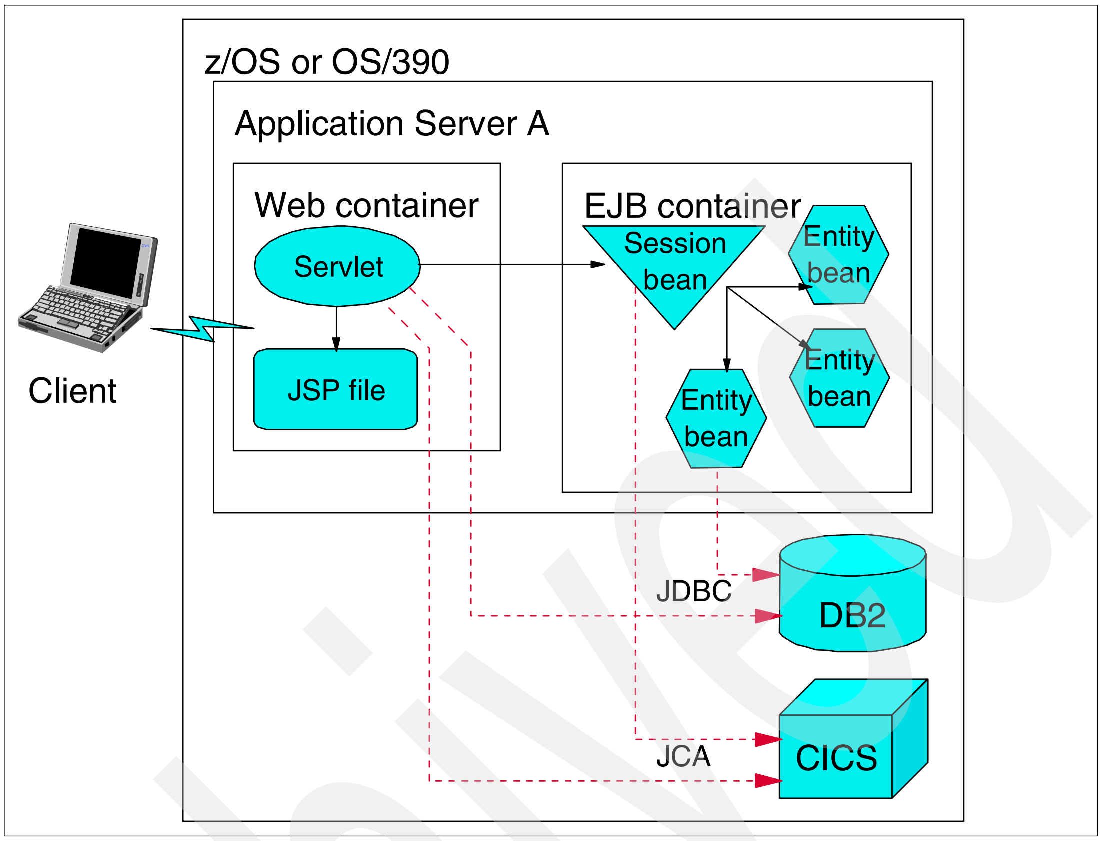


Figure 2-4 Reference configuration with all application components on z/OS

The assessment criteria are described in detail in 2.3, "Evaluation criteria for remote component and EIS access" on page 58. We provide an overview of them here so that we can assess our reference configuration against them.

The criteria may be placed into two broad categories: those related to the functional capabilities; and those related to usability and management issues.

Functional capability aspects

The functional capability aspects are those in which—by converting the application to a hybrid deployment—it may not be able meet its original functional or service level specifications. They are related to the characteristics of the technologies involved with the hybrid deployment.

Performance

Consider how efficiently requests that flow across a networked communications channel get serviced. In the reference configuration, the interaction between components will be efficient.Their close proximity allows for communication optimizations to be implemented to improve performance, whereas requests that flow across a network will incur additional path length and delays as they pass through the network infrastructure.

Note: The EJB 2.0 specification introduces the concept of Local Interfaces for enterprise beans, which will further improve the efficiency of bean interaction within an application server which implements it. In WebSphere Application Server for z/OS and OS/390 V4.01, a similar functionality is implemented by setting com.ibm.CORBA.iiop.noLocalCopies=true.

The potential for exploitation of the z/OS workload management capabilities should also be considered. In the reference configuration, it is possible for there to be several server instances of the application server in the Parallel Sysplex. Technologies exist which are able to take advice from the Workload Manager for placement of the HTTP(S) request which has been submitted from the user. Examples of such technologies are connection optimization and WebSphere Edge Server workload distribution, and these are described in more detail 2.3.1, “Performance” on page 59. Providing that there are no impediments such as session affinities[3], the requests will be intelligently distributed around the server instances in the Parallel Sysplex.

The z/OS Workload Manager also addresses scalability issues. When more application server instances are required due to an increase in the workload, WLM will start them up for us. A server instance possesses a Web container and an EJB container, so both the Web and EJB application tiers benefit from this capability.

WLM routes requests around the system in a attempt to satisfy workload goals which have been set up. In the case of a WebSphere Application Server for z/OS and OS/390 workload, the goals would have been set in terms of request response times. The response time of a request is managed through the use of an enclave, which stays with the request as it get processed by different address spaces. For the reference configuration, the enclave is assigned when the HTTP(S) request is delivered to a server for processing, so the goal which has been defined covers both the Web component and the EJB component processing.

Availability

Examine whether or not the architecture and infrastructure is able to meet application availability goals. The reference configuration provides reduced risk of failure when compared to a hybrid deployment, because there are fewer points at which an error could occur as an application request is processed. Network connections introduce more complexity into the infrastructure, providing an additional risk of failure and increased difficulties in pinpointing and fixing a possible failure.

In the reference configuration, both the Web tier and the EJB tier benefit from the zSeries server architecture and infrastructure, which includes self-healing attributes to prevent downtime caused by system crashes and non-disruptive hardware reconfiguration. The application servers take advantage of Workload Manager and Automatic Restart Manager (ARM) capabilities to ensure the availability of applications.

Security

The security requirements for J2EE applications are broadly the same as they have been for years. You require a means of being certain that the user of the application is who he claims to be (user authentication), and once you know who the user is, you want to restrict access to those resources necessary for the user to complete the task.

Security management in the reference configuration is simplified by the fact that all elements of the application will be working with the same security subsystem: RACF® (or an equivalent SAF-based product). Within a single application server instance, once the end user has been authenticated, the user identity may be propagated without further authentication on access requests to other components, databases, and EIS resources.

If the requests traverses WebSphere Application Server for z/OS and OS/390 application server boundaries, it is possible to set up trust between the servers and use a technique called *asserted identities* to pass the client identity on with the request. In this way, end users

[3] A session affinity exists when session data has been stored in-memory in a specific server instance. It is therefore only available to requests which run in that server instance, so all requests that require access to that data must run there.

only need to authenticate once, and all downstream authorization checks will be against the entered user identity.

Transaction integrity

An important aspect of any transaction system is that of *transactional integrity*. You have transactional integrity when all updates that need to be kept synchronized are either all committed or all rolled back together as a unit. There should be no circumstances which could arise when only part of the transaction was completed.

WebSphere Application Server for z/OS and OS/390 uses Resource Recovery Services (RRS) as the transaction coordinator. Resource managers such as DB2 and CICS register with RRS, which enables them to be instructed to commit or roll back any updates which they are holding as part of a transaction.

Remote Method Invocation (RMI) calls (which are used to issue calls to enterprise beans) preserve transaction integrity, whether the call is to a local bean which is located in the same application server as the caller, or to a physically remote bean in another application server.

If the application running in the reference configuration persists entity beans to DB2 or issues JDBC calls to a DB2 database, then RRS is involved in the transaction coordination through the Resource Recovery Services Attach Facility (RRSAF), a DB2 attachment facility. The CICS Transaction Gateway for OS/390, which is used to issue calls from WebSphere Application Server for z/OS and OS/390 to CICS, runs over a CICS protocol called the External Call Interface (EXCI), which is also enabled for RRS. So in the reference configuration, you have full transactional integrity across all enterprise bean and EIS resources.

Infrastructure

Infrastructure considerations address a broad range of issues which are introduced by the hybrid solution architecture. They include possible changes to component interaction protocol and network implications for setting up the communication channels, including firewall considerations.

In the reference configuration, with all the parts of the application located in the same physical environment, communications are efficient and secure. Any firewall technologies protecting the z/OS system will have had to permit passage for the HTTP(S) request from the client, so there is no requirement to configure it to permit other types of protocol such as IIOP (which is something that some organizations are reluctant to do).

Depending on the size of the installation, the total cost of ownership could be reduced. With everything hosted on the z/OS system, license costs could be reduced; in particular, there will be no license requirements for WebSphere Application Server Advanced Edition and EIS connector images. A consolidated solution also offers opportunities to save in the people and skills costs required to support the runtime.

Usability and systems management aspects

The usability and systems management aspects relate to the amount of extra effort and process that is required to develop and run a hybrid deployment.

Development and deployment

When considering development and deployment, look at application design and programming issues, application assembly, and application deployment. Keep in mind throughout the degree of support that you obtain from tools which are available.

A hybrid solution will inevitably add complexity to all these aspects of application development. In the reference configuration, you are dealing with a single, consolidated application. The IBM tool for J2EE application development is the WebSphere Studio Application Developer V4 family of products. This tool contains a built-in test environment that performs unit testing on the application; the single application is published to the test environment for testing. Enterprise bean components are all registered in the same JNDI namespace under the same initial context, so no special processing has to be incorporated into the application to cater for components located in different places.

When assembling the application, you are only packaging up a single .ear file for deployment. And when deploying the application, because it is in a single file, you do not have to worry about coordinating the deployment of multiple separate pieces. In addition, the deployment tools for WebSphere Application Server Advanced Edition and WebSphere Application Server for z/OS and OS/390 are completely different, whether the deployment is performed manually or using automated scripts. Because you are only deploying to a single application server, you only need a single tool.

Systems management

Systems management comprises the tasks and duties required to maintain the application runtime. These tasks are considerably simplified when you are dealing with a single application server, as with the reference configuration.

For performance monitoring and capacity planning, you need to capture performance metrics which record how efficiently the user requests are being processed. WebSphere Application Server for z/OS and OS/390 records its performance data to SMF 120 records. These records include data for both the Web container and the EJB container, so you are able to track the efficiency of both Web and EJB components using the same monitoring mechanism.

There is also data made available through WLM services. Even more important performance monitoring information for WebSphere Application Server for z/OS and OS/390 actually comes out in RMF™ (SMF records 70-79). If you set up your report classes correctly, you can see the transaction rates, response times, and CPU consumption for each of your applications in the RMF Monitor I Workload Activity Report.

When errors occur in the application, as well as providing a user friendly response to the client request, certain error conditions may need to be raised as *alerts* so that corrective action may be taken by automated operations or manual intervention. The infrastructure that needs to be put in place to provide this function is simpler with all the application components in the same place. You have fewer infrastructure elements (such as application servers, network connections, and routers) to be monitored for error conditions.

In addition, once an error alert has been recognized, with the reference configuration you have fewer logs and traces to analyze because you are only dealing with a single application server.You do not have to coordinate and consolidate diagnostic data from multiple application servers.

To recover from significant failures, backups should be taken of the application data, the application code, and the application server configuration. In the case of the latter two, a single application server makes this process relatively simple when compared with a hybrid deployment, for which each application server will need to be backed up, and both backups need to be coordinated to ensure that following a recovery you do not get mismatched application or configuration components. This is not an issue with the backups of the reference configuration.

Strategic considerations

When thinking about a hybrid deployment, you have to consider whether the solution is appropriate in the long term. How well does the solution align with the strategic and marketplace developments of IBM and the industry?

These considerations are more relevant once a decision has been made to use a hybrid solution. Once the application has been separated, there are a number of technologies available to bridge the separation. A consolidated deployment of Web and EJB components together in the same application server is the simplest option. In fact, the EJB 2.0 specification is introducing the concept of Local Interfaces to optimize the invocation of methods on enterprise beans situated in the same application server. It therefore appears that although application architects are thinking about taking the business logic of applications and making it available as Web Services, local access from more tightly bound and efficient clients will coexist with loosely bound Web Services clients.

2.2.3 Options for logical application separation

When evaluating the feasibility of a hybrid solution, one consideration should be how the application is to be separated. Which components should be retained in the WebSphere Application Server for z/OS and OS/390 environment, and which should be deployed remotely? There are different ways to split the application, and some may prove to be more favorable than others. Although the answer to this question would seem to be obvious (that is, along the Web container and EJB container boundary), in this section we explore some of the reasons why this should be the case.

Where should the application split occur

We are dividing the processing responsibilities of the application by locating components in separate application servers. A component that needs to access another component, or a resource in the other server, will need to do so across some form of network connection. The introduction of the network will change some of the characteristics of the component interaction.

As described in 2.2.2, “Overview of hybrid deployment assessment criteria” on page 37, the reference configuration with a consolidated application deployed into a single server provides many qualities of service. Interaction among components and to EIS resources is efficient, reliable, secure, and you only have a single application server infrastructure to manage.

By separating the application and introducing network connections, you will compromise at least some of these qualities.

The objective should therefore be to limit any adverse affects from separating the application. One approach is to minimize the number of trips across a network connection that the application has to make in order to process a client request. The implications of separating the application for various types of resource access are presented in detail in Chapter 3, “Component interaction characteristics” on page 75.

In many cases, a remote access to a resource across a network link loses some desirable quality when compared to the equivalent local access. For example, the ability of the system to handle concurrent calls might be reduced, or you might lose the security context of the end user and have to substitute it with that of an application server, thus reducing the granularity of resource authorization checks. Where such degradations in service occur, you should seek to reduce their impact by avoiding such calls if possible.

One way to accomplish this is to keep together components that have many interactions and dependencies on each other. In this way, the majority of the calls to other application components and EIS resources are local; you only resort to a remote call when absolutely necessary, and therefore only pay any penalty which the remote access entails when you have to.

Web-EJB separation option

The model-view-controller design pattern, introduced in 1.3, “An introduction to tiers and architectures” on page 14, describes how the application function could be distributed between three design layers: model (for business logic); view (for presentation); and controller (for application navigation). These layers in turn mapped to particular types of application component, as shown in Figure 2-5.

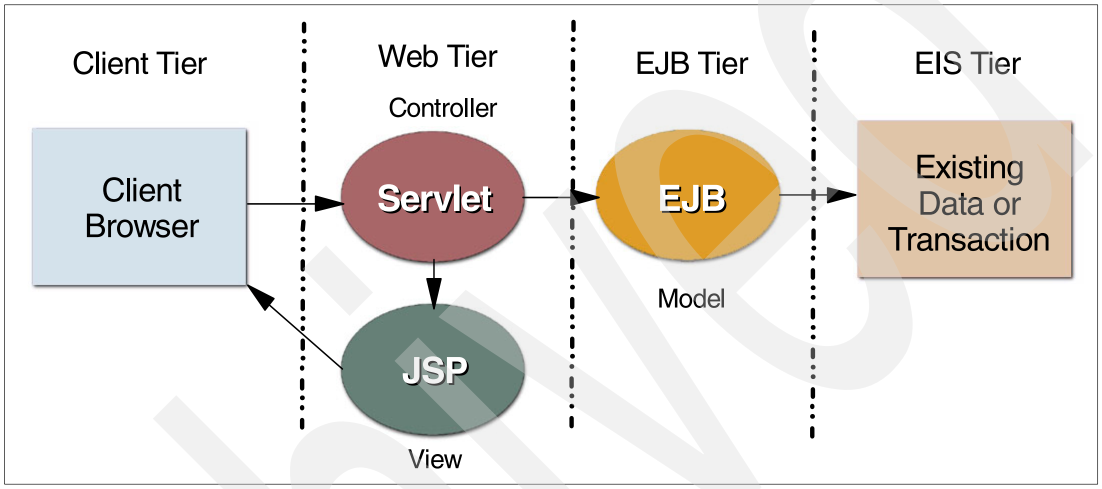


Figure 2-5 Model-view-controller mapping to J2EE components

Designing according to the model-view-controller pattern helps to keep components of similar function (and therefore, most interaction) together, and establish a clean interface between the various tiers. The business logic (the model) is logically distinct from the work flow (controller) and presentation (view) logic, and is only accessible through the use of well defined interfaces. The business logic only needs to understand and interpret the call which arrives through the architected interface; it does not need to concern itself with the nature or the location of the calling application.

You can now place wrappers around the business logic to provide various means for a client application to invoke it. Technologies could include, but are not limited to, using an RMI/IIOP call; using a JMS message; or using Web Services. These are illustrated in Figure 2-6.

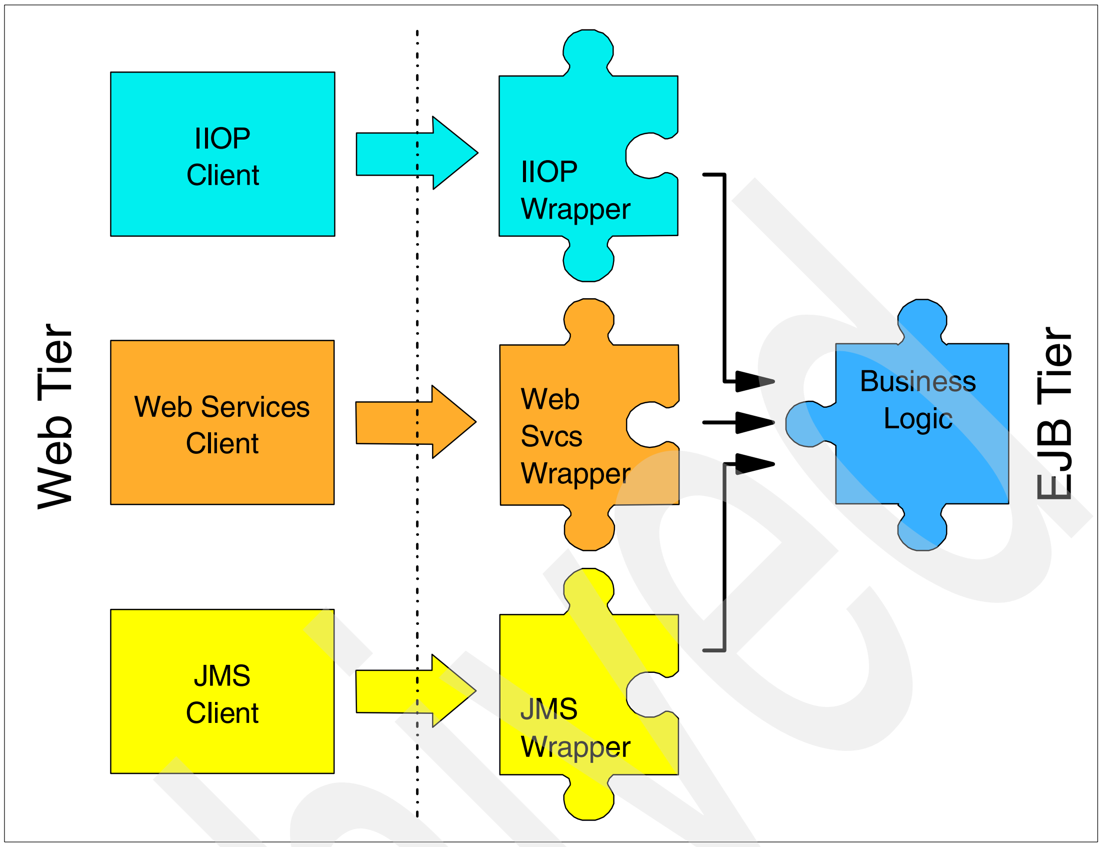


Figure 2-6 Business logic wrappers

You are able to change the technology, and therefore the characteristics, of the client access without requiring any coding changes in the business logic itself. Some technology changes introduce a physical separation between the EJB tier and the rest of the application, which could change the nature of the client access.

Designing and coding according to the model-view-controller design pattern is therefore conducive to a separation of the Web tier from the EJB tier in the physical implementation. Such a separation minimizes the remote flows by keeping together components of similar function, which will have the most interaction among themselves and to common components.

The J2EE approach to application packaging also lends itself to this split. Servlets and JSP files are both packaged in a Web archive (.war file), and deployed to the Web container. Enterprise beans are packaged in a Java archive (.jar file) and deployed to the EJB container. No extensive repackaging exercise is needed to prepare the application for deployment to this hybrid infrastructure.

This approach, which we shall call the Web-EJB separation approach, is illustrated in Figure 2-7 on page 45. In many cases, the business logic for the request will be encapsulated within a single call to a session enterprise bean. The session bean will optionally orchestrate calls to entity beans, before returning the result of its back to the Web component. There will be a single interaction to flow across the network connection.

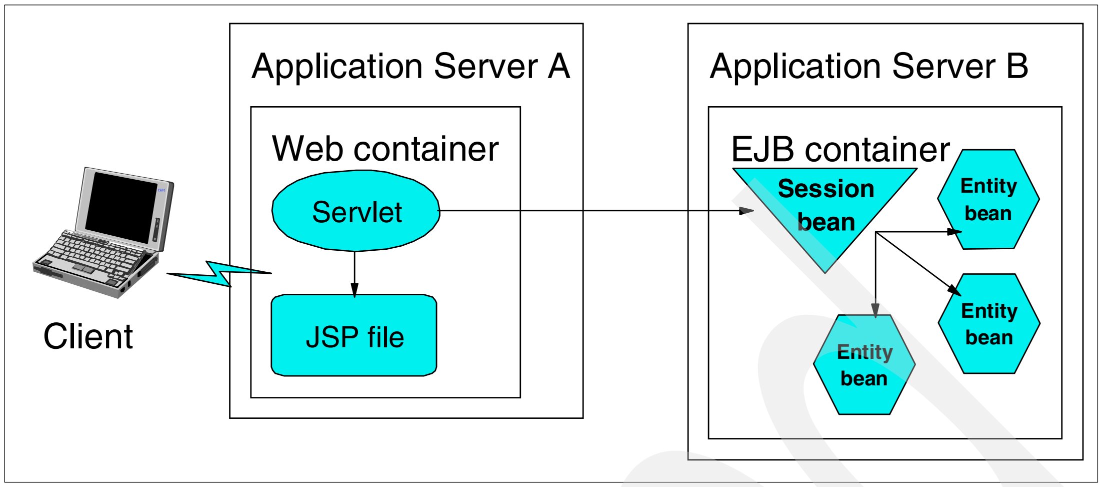


Figure 2-7 Web-EJB separation: separation between the Web container and the EJB container

EJB container separation

A second alternative, which we shall call EJB container separation, would place the split within the EJB components. In this approach, entity enterprise beans would represent the corporate data for the application and be hosted on one of the application servers. In Figure 2-8, this server would be Application Server B. The remainder of the application, including some enterprise beans, would be hosted on a separate application server. In Figure 2-8, Application Server A has a session enterprise bean in its EJB container, as well as all the Web components in its Web container. Calls to access the entity bean from the session bean would be remote.

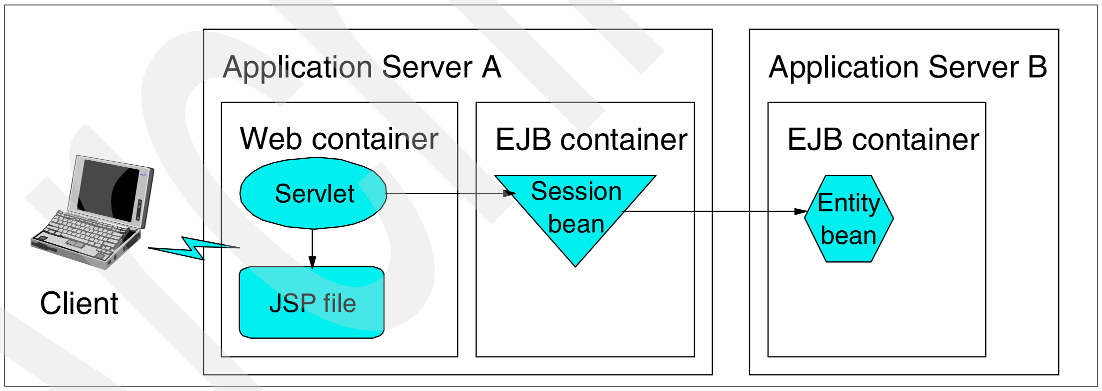


Figure 2-8 EJB container separation: separation within the EJB container

The EJB container separation could be a viable option for requests with simple interactions with the entity beans. If there are several calls to entity beans running in Application Server B, then the network overhead is carried by each of the calls, adding to the total overhead of the request.

Remote access to EIS resources

The requirement for connections to be made to remote resources across a network may not be confined to J2EE component interaction. Servlets are also capable of accessing enterprise information systems (EIS) resources, amongst them databases using the Java Database Connectivity (JDBC) API, or transaction systems such as CICS or IMS using the J2EE Connector Architecture (JCA) API. Unless the resources themselves are moved with the Web container, local access to these resources will need to be converted to remote access. Once again, the characteristics of the connection could be affected by this change.

In 2.2.2, “Overview of hybrid deployment assessment criteria” on page 37, we introduced our reference configuration, with the entire application hosted on z/OS. This is once again illustrated in Figure 2-9. For clarity, we omitted the connections from the EJB container to the resources from the diagram.

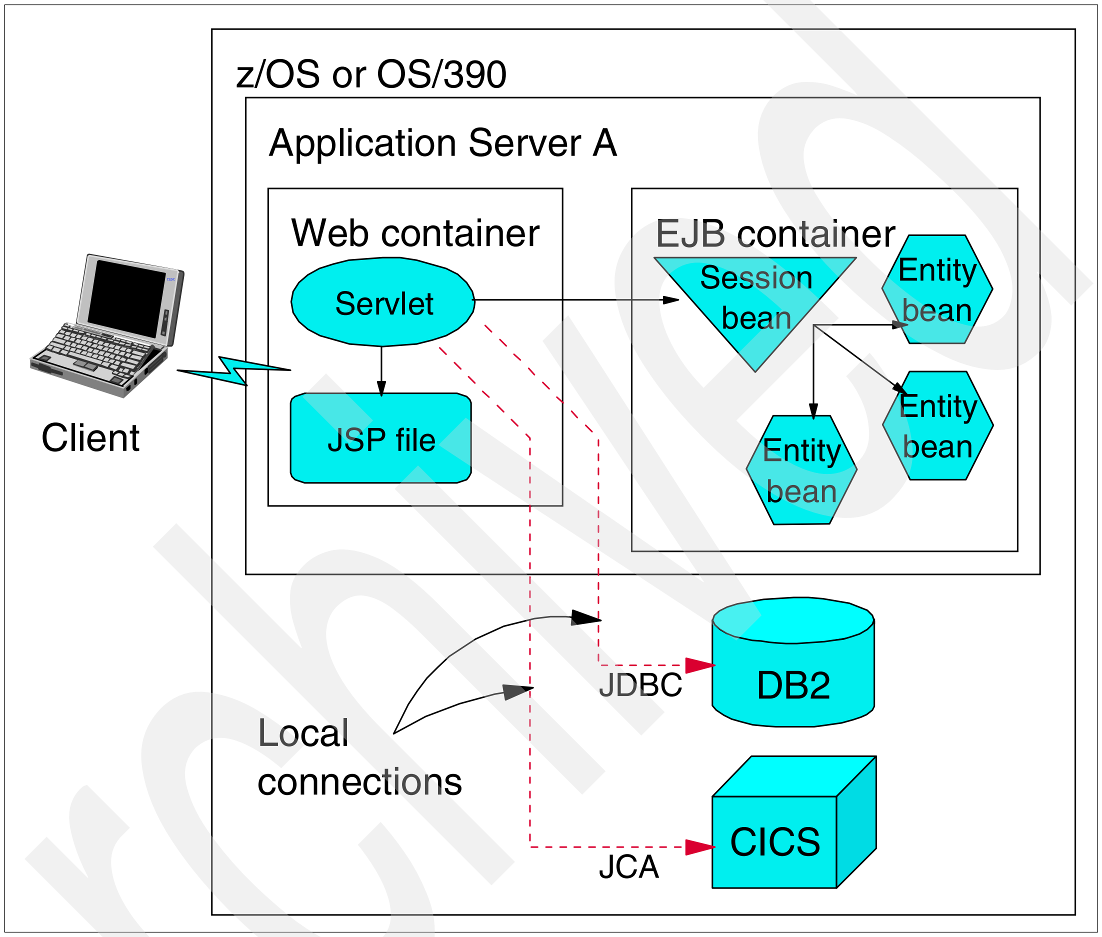


Figure 2-9 Reference configuration with local access to z/OS resources

When both client and server components are located on z/OS, we can make use of Resource Recovery Services (RRS) to provide a 2-phase commit transaction coordination service. The local connections to both DB2 and CICS take advantage of this facility.

Figure 2-10 on page 47 illustrates the same application after the Web container has been moved to another application server. In this example, the DB2 and CICS subsystems have been retained on the z/OS system, so we have had to establish remote connections to them from the Web components.

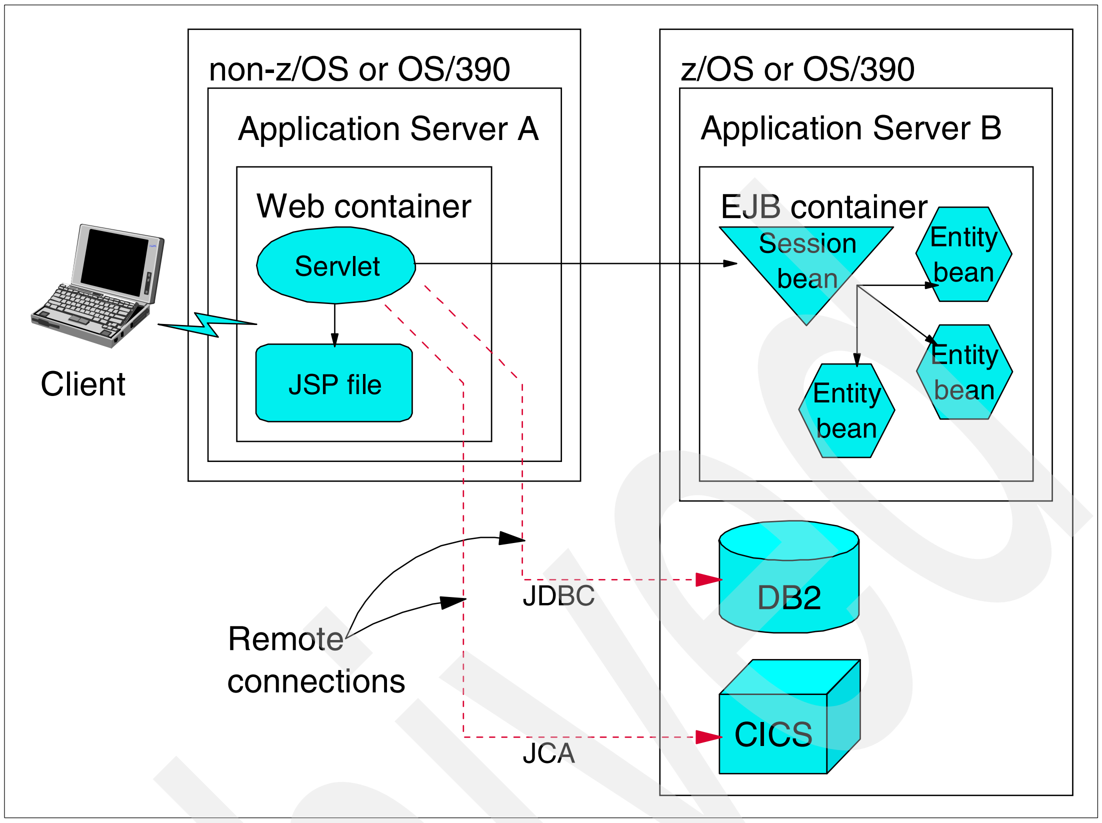


Figure 2-10 Hybrid solution with remote access to z/OS resources

As with the communication between the J2EE components, the replacing of local connections with remote connections could have implications for the characteristics of the call. These are examined in detail in Chapter 3, “Component interaction characteristics” on page 75, but the performance, security, transactional integrity, and systems management procedures could all be affected. For example, RRS is only available to transaction participants that are running in z/OS. The components running in Application Server A have to find some other means of transaction coordination with the resources that they access across remote connections. The J2EE Connector Architecture does not mandate a two-phase commit protocol, so there is no guarantee that transactional integrity across JCA local connections that existed in the reference configuration is preserved for the remote connections in the hybrid configuration.

The model-view-controller design pattern could again assist us, as it could be argued that databases and enterprise information systems (EIS) are all part of the “model”, or business logic. If the application was designed with this in mind, all EIS resource access may originate from business logic components running in the EJB container. If this is the case, there will be no direct resource access from the Web tier, thus avoiding the problem of remote resource access entirely.

2.2.4 Options for physical application separation

In 2.2.3, “Options for logical application separation” on page 42, we described how the application could logically be separated to run across two application servers. In this section, we explore some of the physical implementation options. Suggestions are proposed to illustrate how a hybrid solution might be physically deployed.

EJB container location

The mission-critical part of the application resides in the business logic. This is where we require the best qualities of service. It is where transactional integrity is vital, where the most complex processing in the application needs to be executed efficiently, and where resources and applications need to be protected from unauthorized access. The business logic may include a requirement for access to existing corporate data or transaction applications. In many cases, these resources are hosted on the z/OS platform, and relocating or reimplementing these resources is not an available option.

The runtime environment and infrastructure provided by WebSphere Application Server for z/OS and OS/390 has been designed to provide these qualities of service. In addition, resources accessed from WebSphere Application Server for z/OS and OS/390 use local connection technologies. Local connections on z/OS provide fast and secure access to legacy resources. For example, the platform makes use of Resource Recovery Services (RRS) to ensure transactional integrity between WebSphere Application Server for z/OS and OS/390 and other data and transaction subsystems. Access across a local connection is “within the box”, so we do not have to worry about the management of the network infrastructure across which a resource request from a remote application server would have to travel. Local access also presents fewer security concerns, because all the participating subsystems have been designed to use the same security infrastructure, and there are no network physical security concerns to worry about.

If the model-view-controller design pattern has been followed, the business logic will be implemented as enterprise bean components running in the EJB container. These components may require access for existing data of transaction subsystems using appropriate connectors. There is a good match between the qualities of service typically demanded by business logic and the capabilities of the WebSphere Application Server for z/OS and OS/390 and its runtime infrastructure. In a hybrid deployment, WebSphere Application Server for z/OS and OS/390 is a good choice for hosting the EJB container.

WebSphere Application Server Advanced Edition on a non-zSeries platform

One possible hybrid topology involves a WebSphere Application Server Advanced Edition running on a non-zSeries platform. The solution involves the following assumptions:

- We have adopted the Web-EJB separation approach described in “Web-EJB separation option” on page 43.
- We are using RMI over IIOP as the communications vehicle between Web components and EJB components.
- We want to exploit the traditional qualities of service offered by the z/OS infrastructure to run the application business logic.
- All access to EIS resources using the JDBC and JCA APIs is issued from enterprise bean components running in the EJB container in WebSphere Application Server for z/OS and OS/390.

Figure 2-11 on page 49 illustrates the topology.

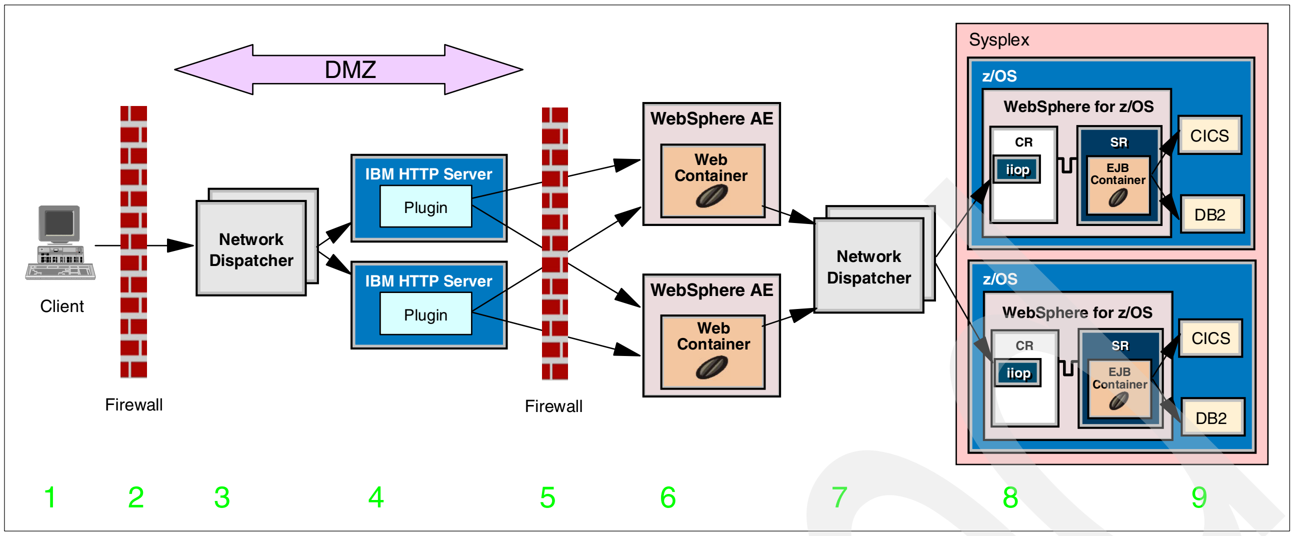


Figure 2-11 WebSphere Application Server Advanced Edition on non-z/OS topology

The various elements of the topology are:

1. The client submits the HTTP request.
2. The request passes through an outer firewall to enter the demilitarized zone (DMZ).
3. The request is intercepted by the Network Dispatcher component of WebSphere Edge Server, which routes the request to one of the IBM HTTP Servers. The Web servers will have been set up as a cluster with a network address which the client will have used to submit his request. The Network Dispatcher has been configured with a standby server to provide high availability.
4. The Web servers have been configured with the WebSphere Web server plug-in to route the requests to the Web container for processing. These requests travel over HTTP or HTTPS. The plug-in performs workload balancing among the application servers based on a round-robin or a random algorithm, while observing any session affinity requirements.
5. The request passes through the inner DMZ firewall into the secure network, where it is picked up and processed by the Web container running in WebSphere Application Server Advanced Edition.
6. The Web component requires access to an EJB component, which it accomplishes using IIOP. The IIOP request is delivered to a second Network Dispatcher.
7. The Network Dispatcher works with the z/OS Workload Manager to route the request to an instance of the WebSphere Application Server for z/OS and OS/390 server.
8. The application server services the IIOP request.
9. EIS resources such as DB2 and CICS are accessed by the enterprise beans using appropriate connection technologies.

For more information on the concepts referred to in this section, refer to:

- *IBM WebSphere V4.0 Advanced Edition: Scalability and Availability,* SG24-6192.
- *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

WebSphere Application Server Advanced Edition on Linux on a zSeries platform

There is a version of WebSphere Application Server Advanced Edition which runs on Linux for zSeries. Under certain circumstances, hosting the Web container of a hybrid solution on an instance of WebSphere Application Server Advanced Edition for Linux for zSeries could be an appropriate solution.

Linux for zSeries benefits from the qualities offered by the hardware platform, which is the most reliable hardware platform available. Possessing features such as memory chip sparing and processor sparing, it offers a mean-time-to-failure measured in decades rather than months. The hardware infrastructure may even be upgraded or repaired nondisruptively.

Virtualization

Linux operating system images may be run on a zSeries processor in several modes:

- Native mode

  Linux is run natively on the zSeries processor. This is not a very efficient or effective use of zSeries processor resources.

- In a logical partition (LPAR)

  A zSeries processor may be divided into 15 logical partitions, each of which behaves as an independent operating system image and is allocated dedicated or shared resources.

- As a z/VM guest

  Linux can run as a guest operating system in a VM virtual machine. In this mode, hundreds or literally thousands of Linux images may be configured.

The third option could be an appropriate solution for server consolidation. Application server images that would otherwise be run on separate physical servers are virtualized on a single zSeries server, as illustrated in Figure 2-12 on page 51.

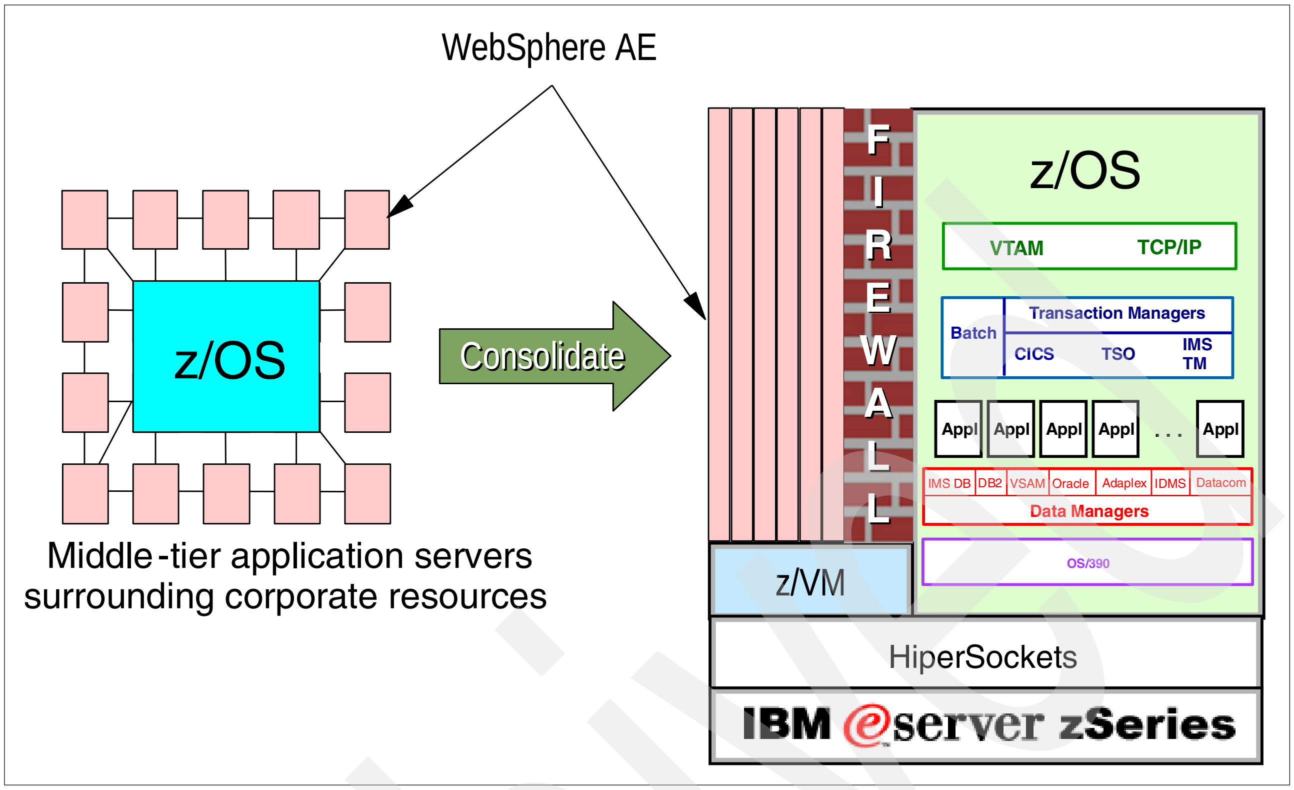


Figure 2-12 WebSphere Application Server Advanced Edition server consolidation on Linux for zSeries

There are a number of reasons why this might seem an attractive proposition:

- Resource utilization

  A zSeries processor benefits from an Intelligent Resource Director (IRD), which means that processing resources such as CPU, memory, and I/O capability are dynamically adjusted to service the workload based on application priorities. Server images that on average are relatively under-utilized can be consolidated to optimize usage of the processing capabilities. Resources will be made available to handle workload peaks when the demand arises.

- Central point of control

  Instead of having to administer a large number of physical servers, all the “virtual” servers are located on one physical infrastructure. This could realize cost savings due to reduced physical floor space and support personnel. Linux may also play multiple roles in a network topology, so elements such as firewalls may be hosted on the same physical box as the application servers.

- Flexibility

  Linux images may be created and destroyed much more easily and cheaply than it is to manage physical servers running non-z/OS operating system images. This may be beneficial in an environment where there is a volatile demand for server images, such as might be the case with a development organization that requires different server configurations for the various test phases of a project.

- Skills

  Once the z/VM infrastructure has been set up, no z/OS-specific skills are required to manage the WebSphere AE images running in Linux for zSeries. The skills required are the same as for Linux running on any other platform.

- Horizontal scaling

  Linux for zSeries benefits from the physical horizontal scaling capabilities of the zSeries hardware, which is capable of creating a very large processor configuration. In addition, there is a logical horizontal capability provided by the ability to create more Linux images on demand.

- Hipersockets

  When communicating with WebSphere Application Server for z/OS and OS/390, the IP flows across Hipersockets, a network connection that is internal to the zSeries server. Hipersockets provides simplified network management, with no additional network hardware required. Network security is improved because the flows never leave the zSeries box, and because a Hipersockets network requires fewer moving parts or wires to trip over, the connection is more reliable. The data is being transferred across memory between participants that are physically located close together, so the performance of Hipersockets communication is also very good.

- License costs

  License costs could be reduced due to the “sub-CPU granularity” of resource usage. Instead of an application server spending a large part of its time idle on a dedicated server, it is one of any number of servers sharing common resources.

Important: Note that WebSphere Application Server Advanced Edition for Linux for zSeries is not an alternate configuration for WebSphere Application Server for z/OS and OS/390; they are different products. WebSphere AE for Linux on zSeries benefits from the hardware attributes of the zSeries server, but it has not been integrated with z/OS features such as Workload Manager, Resource Recovery Services, and the security infrastructure. WebSphere AE for Linux for zSeries does not possess the highest qualities of service provided by WebSphere Application Server for z/OS and OS/390.

Virtualization workloads

Not all application servers and workloads are appropriate for Linux for zSeries virtualization. The “sweet spot” for virtualization is a server that possesses the following attributes:

- Lower qualities of service demands

  WebSphere AE for Linux for zSeries has inferior qualities of service to WebSphere Application Server for z/OS and OS/390, because it has not been integrated with z/OS subsystem features. For example, it does not exploit RRS, and so is not capable of 2-phase commit interaction with CICS.

- Low average utilization

  Virtualization consolidates multiple servers to share processing resources. The benefits will be most evident for servers with a low average utilization. Servers with high utilization will consume larger quantities of processing resources, reducing the potential for resource sharing.

  Note: One of the possible motivations for a hybrid solution was to save zSeries resources by offloading the Web component processing. If this was the case, presumably the Web components consumed a significant amount of processing resources, so Linux for zSeries may not be an appropriate solution.

- Spikey workloads

  Application servers on dedicated server boxes need to be configured for capacity. They have to be capable of handling the maximum workload which, irrespective of what the average workload is. In a Linux for zSeries environment, when the peak workload arrives, more processing resources may be made available to accommodate the increased demand. Thus, the expense of dedicated servers standing idle waiting for a workload peak is avoided. This solution is therefore appropriate for workloads with seasonal peaks.

- Uncoordinated workloads

  Virtualization is not a good solution for running multiple clones of the same application server. This is because when the workload peaks, it will be for all the clones at the same time, so just when we most need it, we lose the benefit of dynamically reallocating resources to where they are required. It is more appropriate for a collection of disparate application servers, running different workloads that peak at different times.

- Test servers

  Servers used for application development and testing are typically low utilization and have a volatile usage pattern. They are therefore a good candidate for virtualization on Linux for zSeries.

For more information on Linux for zSeries and virtualization solutions, refer to the following publications:

- *Linux for S/390,* SG24-4987.
- *Linux on IBM zSeries and S/390: ISP/ASP Solutions,* SG24-6299.

Hybrid deployment involving Linux for zSeries

A possible configuration for a hybrid solution would involve the EJB components running in WebSphere Application Server for z/OS and OS/390, and the Web components running in WebSphere Application Server Advanced Edition for Linux for zSeries. This is illustrated in Figure 2-13.

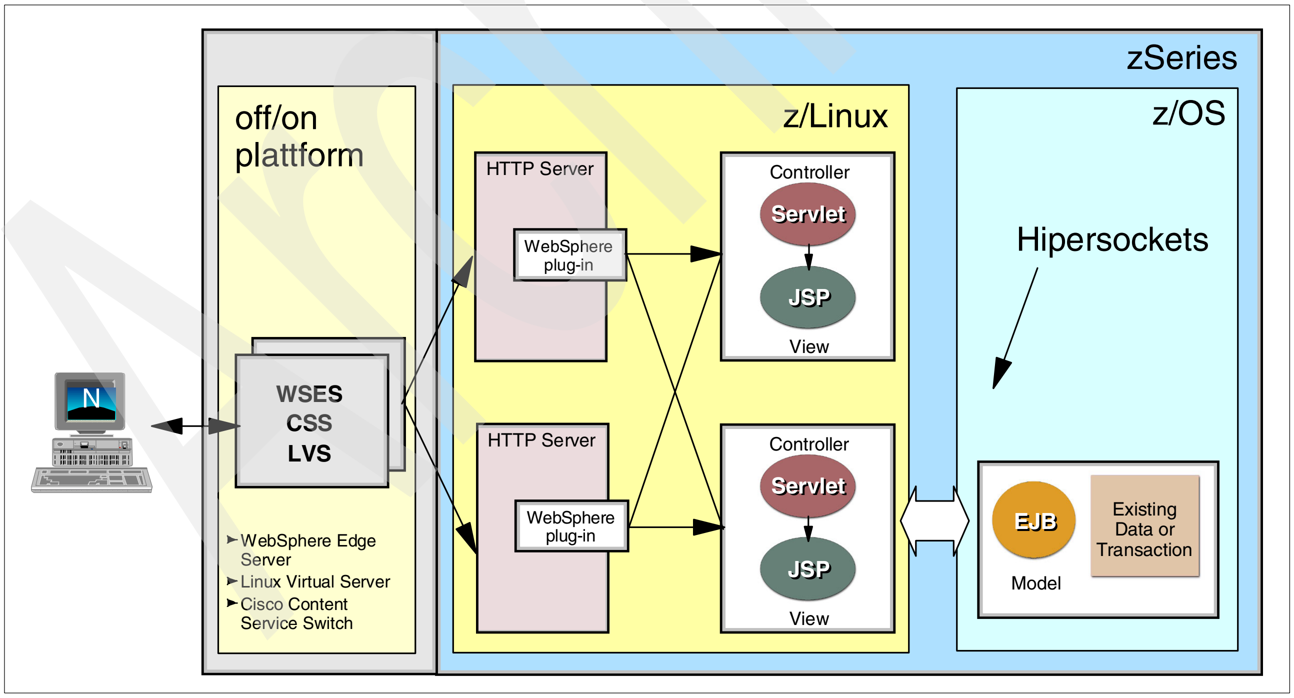


Figure 2-13 Hybrid deployment involving WebSphere AE on Linux for zSeries

With this configuration, the incoming client requests may be distributed amongst Web Servers running in Linux for zSeries using workload balancing technologies that are either on- or off-platform. The WebSphere HTTP plug-in balances work between the application servers running Web components, and requests to the EJB components are submitted across Hipersockets connections.

2.2.5 Options for J2EE inter-component communication

When separating the J2EE application into two parts, there are a number of mechanisms that may be used to provide intercomponent communication.

RMI/IIOP

Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP) is the method by which enterprise bean methods are invoked. It is used by both standalone client programs and Web components when enterprise bean functionality is required.

RMI APIs allow developers to build distributed applications in the Java programming language. They enable an object running in one Java Virtual Machine to access another object running in a different Java Virtual Machine.

IIOP is a protocol used for communication between CORBA object request brokers. An object request broker is a library that enables CORBA objects to locate and communicate with one another.

RMI/IIOP is an implementation of the RMI API over IIOP that allows developers to write remote interfaces in the Java programming language. It is required to be supported by an application server in order to achieve compliance with the J2EE 1.2 specifications. When development tools generate enterprise bean access code, RMI/IIOP is the technique used. The protocol supports global transactions, even if the client and server participants are remotely located.

For a detailed investigation of the characteristics of an RMI/IIOP connection and the implications of accessing an enterprise bean from a remote client, refer to 3.2, “RMI/IIOP access to remote enterprise beans” on page 76.

Java Messaging Service (JMS)

One possible factor for considering a hybrid deployment solution is a re-architecting of the application. This could be due to a number of reasons, but one of them might be a requirement to make the business logic available to new client applications whilst retaining the Web interface. One possible solution would be: the client application sends its request for business logic execution in an IBM WebSphere MQ message.

IBM WebSphere MQ overview

IBM WebSphere MQ (WMQ) provides an asynchronous messaging infrastructure that is capable of integrating applications spanning more than 25 different platforms. It provides messaging functions for both servers and clients, ensuring once-only delivery of messages.

It is because of its ubiquity that it is a popular choice for connecting applications running on different platforms. The nature of the connections over which the messages flow depends on the WMQ components involved.

A WMQ server running on z/OS may be accessed from another server across a peer-to-peer connection. In this case, the messages are transferred asynchronously, so a client application will write a message to the sending server and will have control returned as soon as the operation is complete. The message will be held by the sending server until the receiving server is ready. In addition, a WMQ server on z/OS may also receive messages directly from a client, but in this case the WMQ server must be available for the client to connect to in order for the message to be written.

JMS Listener

J2EE applications access messages using Java Messaging Service (JMS). JMS provides a generic API for access to messaging services, one of which is WMQ.

In order for enterprise beans to process WMQ messages, a session bean must be instantiated. To do this with no directly connected Web client application, we must make use of a JMS Listener. Application servers that support the EJB 2.0 specification will be able to make use of Message Driven Beans (MDBs). MDBs provide an infrastructure that listens for incoming messages and initiate an appropriate session bean to execute the business logic.

WebSphere Application Server V4.01 for z/OS and OS/390 only supports the EJB 1.1 specification, so an alternative technique must be used.

The JMS Listener is a long-running task, typically an auto-started servlet, that waits for messages to arrive. A problem with running the servlet in the WebSphere Application Server V4.01 for z/OS and OS/390 Web container is that the servlet cannot create client threads to access managed objects. The workaround is to move the servlet to run in the HTTP Server plug-in. This solution is described in full in the *Overcoming a Problem Running Simulated Message Driven Beans,* WP100301 whitepaper, which is available from the TechDocs library at:

```
http://www-1.ibm.com/support/techdocs/atsmastr.nsf/Web/Techdocs
```

Although this solution is possible, and may be considered strategic for future exploitation of MDBs when support is available, it does involve not only reinstating Web component processing on the z/OS platform, but also the introduction of the IBM HTTP Server in order to run the servlet in its WebSphere plug-in.

Note: At the time of writing, WebSphere Application Server V5 for z/OS and OS/390 was not available. Today, WebSphere Application Server V5 for z/OS and OS/390 enables dynamic application interaction through native, high performance Java Messaging Service (JMS), J2EE 1.3 Message Beans, and container-managed messaging. Therefore, all the JMS Listener problems discussed have been solved by the product implementation.

Performance and availability

To ensure performance and availability, a number of WMQ servers running on z/OS would have access to a shared queue. A queue is a repository for storing messages, and a shared queue is one that makes use of sysplex facilities to make itself available to a number of WMQ servers running in different z/OS images.

Each WebSphere Application Server for z/OS and OS/390 server would talk to a WMQ server in its own operating system image. The message listeners would issue a *get* for a message against the shared queue, and wait for one to arrive. The message *gets* would be serviced on a first come, first served basis, so the first message to arrive would be presented to the first listener that issued a get; the second message to the second listener; and so on. Failed servers will not be targeted with messages, because they will not have a get request outstanding. This first come, first served processing will also ensure that the work is distributed among the application servers.

Client application connectivity

The client applications will be connected to the z/OS WMQ servers using MQ channels. The client could be a Web application running in WebSphere AE, or it could be a rich client. The clients could connect directly to the z/OS WMQ servers, or they could connect to a WMQ server running locally on its own platform. Either way, the connection to WMQ on z/OS will be either LU6.2 or TCP/IP.

If the connection is TCP/IP, then the possibility of the connection flowing through a firewall has to be considered. The firewall will need to enable access for the relevant port numbers. There is a WMQ SupportPac™, MS81: WebSphere MQ Internet Pass-Thru (MQIPT), which can be used to simplify the passage of WMQ channel protocols through firewalls, by wrapping the flows as HTTP requests. This SupportPac may be downloaded from:

```
http://www-3.ibm.com/software/integration/support/supportpacs/individual/ms81.html
```

When connections are established to WMQ servers, their placement may be subject to workload balancing using technologies such as Sysplex Distributor. However, the connection will be established with a single instance of the WMQ server, and it persists until it is closed. Once a client server has established a connection, all its messages are transmitted to the same WMQ server on z/OS. However, this would not pose too much of a problem, because once the message is on the shared queue, it would be processed by whichever server was next in line with a get for a message.

Development

Redesigning a J2EE application to provide a JMS-based interface between the Web components and the EJB components is a significant undertaking. It would likely have to be justified by a requirement to expose the business logic to other client applications in addition to the Web application.

The Web application could be designed to perform a synchronous operation with the EJB components, even though it is operating across an asynchronous transport. The Web component would put a message on a queue, either directly to the WMQ server running on z/OS or to a local WMQ, which would then forward the request on to WMQ on z/OS. It would then wait for a reply message to be returned once processing had been completed, and construct some appropriate response to be sent to the end user. Such a design needs to take account of the fact that the processing of the message could be delayed indefinitely.

The WMQ servers will look after the message to ensure it is not lost, but because messaging is an asynchronous operation, the message could get held up due to the unavailability of some aspect of the infrastructure, such as WMQ servers or WebSphere application servers. Under such circumstances, the message will eventually get processed when the infrastructure is fully operational, and a response message sent back to the client application. The problem is that the client application may still be awaiting the response, or could have timed out and gone away. The application design needs to accommodate this possibility, as follows:

- Place a limited shelf-life on the original request message, so that after a specified period of time the message is no longer “valid” and will not be processed.
- Adopt a “fire and forget” approach, in which WMQ is trusted to ensure that the request message will not be lost, and the client application does not wait for a response message but continues processing in the knowledge that the message will be delivered. The application has to include some form of compensation processing should errors occur while the message is being processed.

Strategic aspects

The J2EE implementation of MDBs is available with application servers that support the EJB 2.0 specification. WebSphere Application Server V5 supports J2EE 1.3, which includes this specification, providing support that has been architected into the product.

Web Services

An alternative approach to loosely coupling the Web and EJB components is Web Services. Web Services is one aspect of a service-oriented architecture (SOA) approach to application development, the goal of which is to provide support for the connection and sharing of data and resources in a flexible and standardized manner.

Service-oriented architecture

Each component in a service-oriented architecture can play one (or more) of three roles: service provider, broker, and requestor, which perform the operations shown in Figure 2-14.

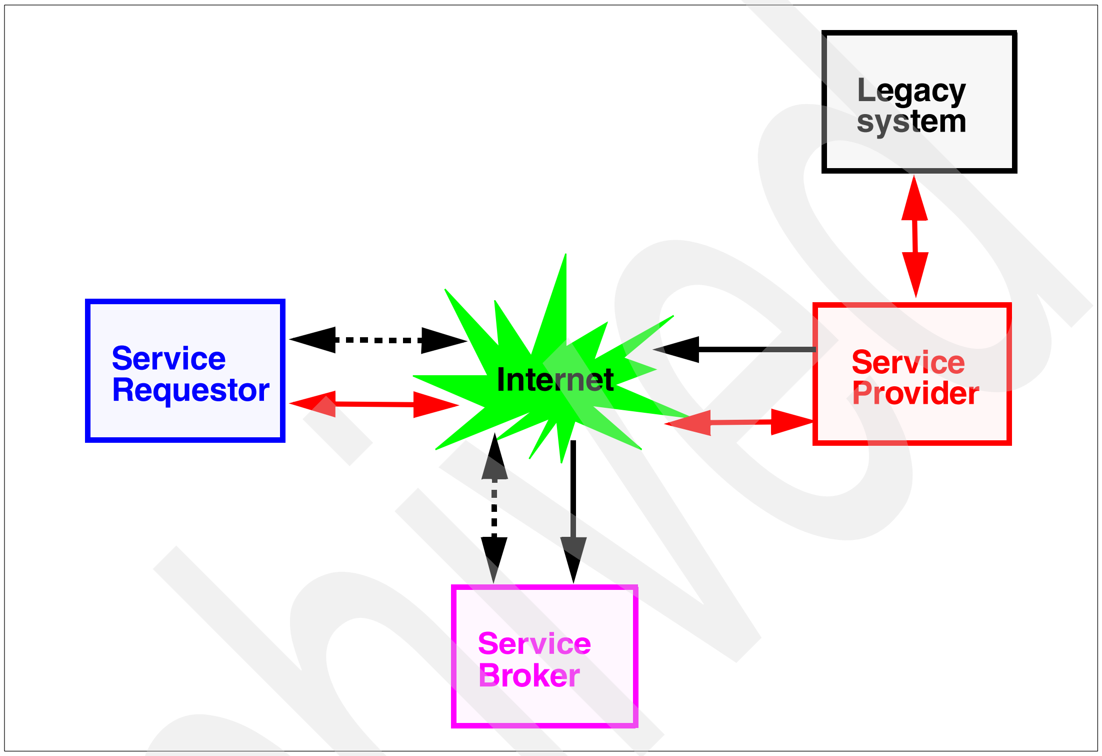


Figure 2-14 Web services roles and operations

- The service provider creates a Web service and possibly publishes its interface and access information to the service registry. Each provider must decide which services to expose, how to make trade-offs between security and easy availability, how to price the services (or, if they are free, how to exploit them for other uses). The provider also has to decide what category the service should be listed in for a given broker service and what sort of trading partner agreements are required to use the service.
- The service broker (also known as service registry) is responsible for making the Web service interface and implementation access information available to any potential service requestor. The implementers of a broker have to decide about the scope of the broker. Public brokers are available all over the Internet, while private brokers are only accessible to a limited audience, for example users of a company-wide intranet.

  Furthermore, the width and breadth of the offered information has to be decided. Some brokers will specialize in breadth of listings. Others will offer high levels of trust in the listed services. Some will cover a broad landscape of services and others will focus within a given industry. Brokers will also arise that simply catalog other brokers. Depending on the business model, a broker may attempt to maximize look-up requests, number of listings, or accuracy of the listings.
- The service requestor locates entries in the broker registry using various find operations, and then binds to the service provider in order to invoke one of its Web services. One important issue for users of services is the degree to which services are statically chosen by designers compared to those dynamically chosen at runtime. Even if most initial usage

is largely static, any dynamic choice opens up the issues of how to choose the best service provider and how to assess quality of service. Another issue is how the user of services can assess the risk of exposure to failures of service suppliers.

Web Services is an implementation of a service-oriented architecture designed to provide access to application services using standard protocols and tools that are independent of the service's implementation. When a Web service has been published, the service provider may never have had any previous interaction with the service requester. However, it is also possible to make a service available internally within an organization, and make use of it as an in-house application. In this case, the service requestors may not need the service broker, because they could have been provided with details of how to access the service by some other means, such as e-mail.

Access to Web services

The service-oriented architecture has flexibility built into it to enable services to be invoked by any standard communications infrastructure. As the concept evolves, a broader range of options will appear, but initially the most common access mechanism is through the use of Simple Object Access Protocol (SOAP) messages. SOAP is a network-, transport-, and programming language-neutral protocol that allows a client to call a remote service. The message format is Extensible Markup Language (XML), a generic language that can be used to describe any kind of content in a structured way, separated from its presentation to a specific device. HTTP provides the runtime transport.

The WebSphere Studio Application Developer Integration Edition V4.1.1 tool provides support for developing applications that follow a service-oriented architecture approach. They also provide wizards for generating most of the documents and code required to produce a Web service. The Web service may be made available either as a session enterprise bean or as a SOAP service (or both).

Use of Web services for a hybrid solution

If an application is to be exposed as a Web service, then it is possible that one of the client applications for accessing the service is a Web application with a browser as the user interface. In this instance, it is possible that Web components could invoke enterprise bean logic as a Web service. However, this mechanism carries a large performance overhead compared to RMI/IIOP. The development tools provide wizards to wrapper an enterprise bean as a Web service, so it is more likely that the RMIRI/IIOP access and the Web services access will coexist to provide access to common business logic from different client applications.

2.3 Evaluation criteria for remote component and EIS access

As part of assessing the feasibility of a hybrid deployment, consideration needs to be given to the effect that the separation will have on the application when request execution needs to cross the divide between the two (or more) parts of the application. When access to components and enterprise information systems (EIS) resources changes from a local connection to a remote connection, the characteristics of the connection could change.

Chapter 3, "Component interaction characteristics" on page 75 examines the implications of a local to remote conversion for some specific connection examples. In this section, we discuss in general terms the criteria for comparing the differences between local and remote access. The evaluation criteria presented may be applied to any type of connection that has not been included in our collection of examples.

A hybrid solution will require some component access, and possibly some EIS access, to be converted from local to remote. Assessing each type of access against these criteria will assist with determining whether a hybrid solution is appropriate. It may also assist with determining the most appropriate place to insert the separation in a hybrid deployment. Different points of separation may require different remote connections to be established. Following an evaluation of the remote connections against the criteria, different solutions may incur different degrees of service degradation or compromise. This information will influence where the application separation is implemented.

2.3.1 Performance

Separating the application increases the overall instruction path length for all requests that require processing by components in both parts of the application. Local requests are fast because the transport is through a program call. It avoids the overhead of the network and data transformations, simplifies the marshalling of requests, and is able to use optimized RACF facilities for security. From remote clients, more heavyweight security mechanisms such as SSL and Kerberos have to be employed. An assessment has to be made as to whether the cost of this increased overhead is worth the intended benefits.

Capacity

One of the possible motives for a hybrid solution is to maximize the benefit of WebSphere Application Server for z/OS and OS/390 processing capacity. By relocating the Web components on another platform, we free up valuable processing resources and focus them on the kind of processing that the z/OS platform has historically been used for: mission-critical business logic. A consideration might be: what are the relative amounts of resource consumption between the Web container components and the EJB container components?

If the bulk of the application processing is spent in the Web components, then some benefit may be obtained from performing this activity on a cheaper platform that does not require such high qualities of service. This benefit must of course be weighed against the overhead of remote component and/or EIS access that has been introduced.

There are a number of performance monitoring products which may be used to investigate the CPU consumption of various application components. These include:

- WebSphere Studio Application Monitor
- IBM Tivoli® Monitoring for WebSphere Application Server on z/OS
- IBM Resource Measurement Facility (RMF)
- Other vendors' monitoring products

For more information on these products and their capabilities, refer to *Monitoring WebSphere Application Peformance on z/OS,* SG24-6825.

Bear in mind that performance monitoring and capacity planning will be more complex with a hybrid deployment. Although performance monitoring tools for WebSphere application servers on all platforms exist, resource consumption statistics now originate from multiple sources for the same application. Performance metrics may need to be merged or correlated to provide a complete picture. It could also be difficult to relate changes in workload volume to changes in the WebSphere Application Server for z/OS and OS/390 application server performance, particularly if the EJB container is configured as an EJB "repository" for a number of hybrid applications, each with Web components deployed to different servers, as shown in Figure 2-15.

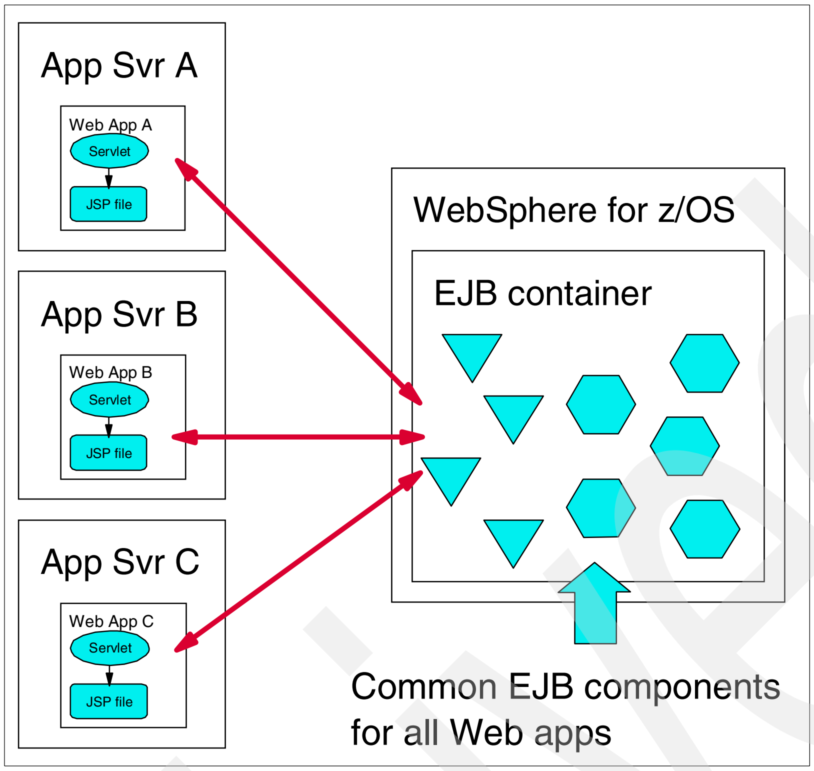


Figure 2-15 Common EJB components for multiple Web applications

Scalability

With a consolidated application running in WebSphere Application Server for z/OS and OS/390, we can take advantage of the scalability features of z/OS. As the demands of the incoming workload volume vary, z/OS is capable of starting and stopping application server instances. This provides an effective horizontal scaling solution. As the workload increases, more replica servers are made available to process the client requests.

This is illustrated in Figure 2-16 on page 61. Work is directed to the Queue Manager Address Space, which is a Control Region in WebSphere Application Server for z/OS and OS/390 terms. The Control Region works with the z/OS Workload Manager to queue the work. Any number of Queue Server Address Spaces (or Server Regions) pull work off the queues and process it. Workload Manager adjusts the number of Server Regions available to cope with the incoming workload.

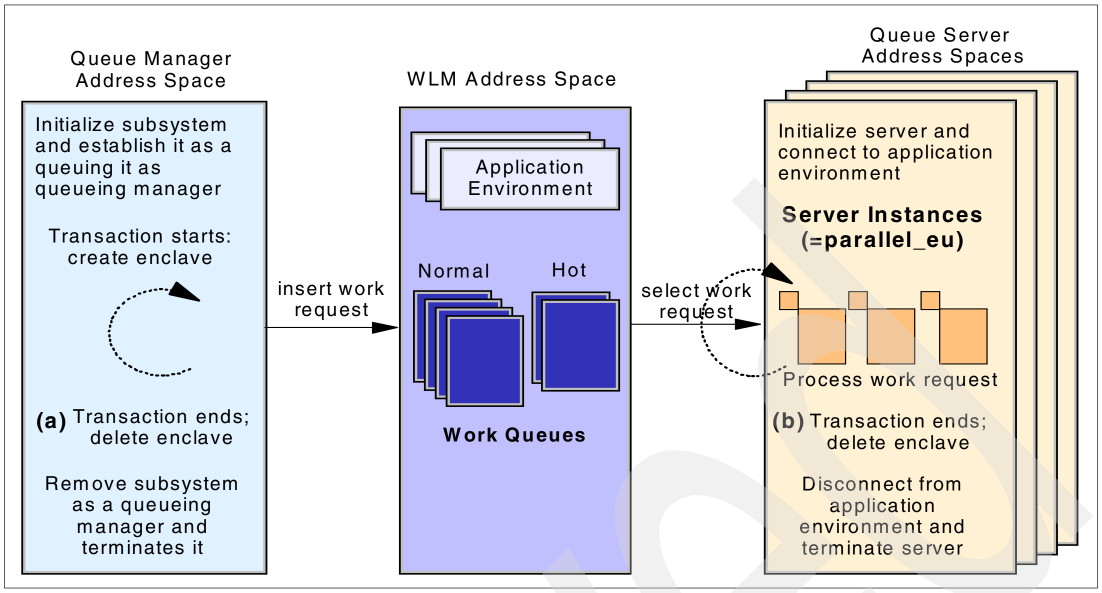

Queue Manager Address Space
WLM Address Space
Queue Server Address Spaces
Initialize subsystem and establish it as a queuing it as queueing manager
Transaction starts: create enclave
(a) Transaction ends; delete enclave
Remove subsystem as a queueing manager and terminates it
Application Environment
Normal
Hot
Work Queues
insert work request
select work request
Initialize server and connect to application environment
Server Instances (=parallel_eu)
Process work request
(b) Transaction ends; delete enclave
Disconnect from application environment and terminate server

Figure 2-16 Horizontal scalability for WebSphere Application Server for z/OS and OS/390

With a consolidated application deployed to a single server, both the Web components and EJB components reside in a WebSphere Application Server for z/OS and OS/390 server. If the z/OS Workload Manager decides that more server regions need to be spawned in order to handle the incoming workload, then additional containers for both the Web components and the EJB components are created.

In a hybrid solution, not all of the application resides in WebSphere Application Server for z/OS and OS/390. Part of it has been installed in another application server running outside of the z/OS environment. Although the WebSphere Application Server for z/OS and OS/390 components will benefit from the scalability afforded by the z/OS Workload Manager, the off-platform components and the intervening network infrastructure do not. We have more layers in a hybrid solution for which we need to figure out a scalability solution.

Assuming that in a hybrid solution the Web container is located on a non-z/OS platform, consider whether the Web container and the network infrastructure between the application servers can scale. WLM can only dynamically control the number of available servers running EJB components. The infrastructure for the Web container and network connection may have to be configured for sufficient capacity to handle any anticipated workload. This may not be an efficient use of resources if the workload volumes can vary by significant amounts, because during quiet periods the server resources will remain idle.

Workload distribution

An application is deployed to a WebSphere Application Server for z/OS and OS/390 J2EE server. The J2EE server is comprised of a number of server instances, where each server instance may run on a different operating system image in the sysplex. The application servers running on several physical systems are collectively known as a host cluster, and are viewed by the outside world as a single system with a single Daemon IP name. This configuration assures application availability, but to make the best use of available resources, some means of distributing work across the running servers is required.

There are a number of technologies available for distributing work that arrives over an IP connection around the sysplex.

- Round robin DNS

  This solution makes use of a *Domain Name Server* (DNS) to resolve the host name attached to the client request to one of a number of alternate IP addresses. This provides a primitive load balancing solution, but takes no account of servers that may be offline.

- Connection optimization

  Connection optimization also makes use of a DNS, but this time it runs on z/OS. It dynamically resolves the host name of requests to a specific server, and does so by taking advice from the z/OS Workload Manager as to the best place to route the request based on current CPU, memory, and I/O activity.

- IBM WebSphere Edge Server Network Dispatcher

  Network Dispatcher is a router with the host name associated with its address. It takes requests and forwards them on to an appropriate server. It is capable of taking advice from the z/OS Workload Manager regarding the placement of requests. It also supports content-based routing, so it can handle any HTTP session affinity requirements that require the request to be directed to a specific server.

- Cisco Multi-Node Load Balancer (MNLB) and Cisco Content Services Switch (CSS)

  Cisco's Multi-Node Load Balancing (MNLB) solution is also a router that is capable of routing requests according to advice provided by the Workload Manager.

- Sysplex distributor

  Sysplex distributor is a technology for routing work into a sysplex. It also works with Workload Manager to determine the target server. It may be used in conjunction with other solutions, such as WebSphere Edge Server or Cisco's load balancers.

- WebSphere HTTP plug-in

  If the workload consists of HTTP(S) requests, then they may be directed to a Web server running the WebSphere Application Server Advanced Edition Web server plug-in. The plug-in can then distribute requests to the application server instances on a round-robin basis, or more intelligently with sysplex distributor assistance. It also handles HTTP session affinities.

Note: A more lengthy description of these technologies and their capabilities is provided in *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

For a consolidated application that was installed in its entirety into a WebSphere Application Server for z/OS and OS/390 J2EE server, it is the client's HTTP(S) requests which will need to be distributed amongst the sysplex for processing. The above solutions are capable of providing this function, although the first two DNS solutions are not strategic. For a hybrid solution, the client's HTTP(S) request will already have been intercepted by the application server running the Web components. Depending on how the hybrid solution has been architected, requests for EJB container function or EIS access could arrive over a variety of protocols, for example IIOP, or a JMS message, or as a SOAP request over HTTP(S), or some other protocol. Some may have more flexible workload distribution capabilities than others, so each connection and protocol will need to be evaluated on its own merit.

Workload distribution inhibitors

The ability to effectively distribute work may be affected by the manner in which the application was designed. It could contain coding practices that require workload distribution algorithms to be bypassed. For example, to look up and instantiate an enterprise bean is an expensive operation, so a common practice is to cache the object reference (or handle) once these operations have been completed. The object reference may then be reused on subsequent requests to save the cost of repeating the lookup and instantiation.

This tactic improves the performance of the application, but it means that every access to the subject of the cached reference will be directed to the same server. The object reference includes information about the specific server in which the object was originally located, so all requests are directed there, bypassing any workload distribution intervention.

Workload management

As well as providing routing advice for directing work around the sysplex, the z/OS Workload Manager is responsible for managing workload performance by optimizing the use of resources required to complete client requests. It assigns processing resources such as CPU, memory, and I/O to address spaces processing the workload in an attempt to satisfy the performance goals that have been set.

Address spaces themselves may have performance goals, but work requests entering the z/OS system may also be assigned a workload manager enclave. An enclave determines the performance service class of the request, and the level of service and priority that the request should receive. Figure 2-17 shows two work requests that have been assigned different enclaves with different service classes, which results in their running in different server regions.

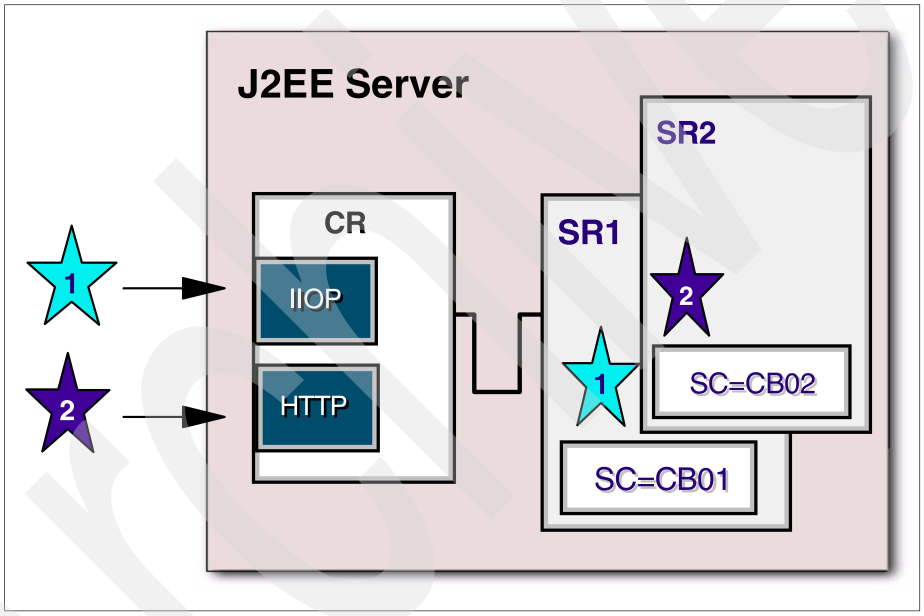


Figure 2-17 WebSphere Application Server for z/OS and OS/390 workload classification

There are some considerations regarding the allocation and use of enclaves with hybrid solution work requests that arrive from a remote Web container. We first have to determine how it is possible to assign an enclave when the request arrives at the z/OS platform. For client HTTP(S) requests, there are three options:

1. The service class may be assigned based on the user ID under which the work is running.
2. The service class may be assigned based on the server name.
3. The service class may be based on the URI of the request, either in the Web server or in the HTTP Handler.

In a hybrid solution, it is more likely that an IIOP request rather than an HTTP(S) request will arrive at the z/OS platform. For remote IIOP requests, we lose option 3 from the above list. So by adopting a hybrid solution, we may lose some flexibility for managing the workload.

Another consideration is that the workload goals will now only apply to the EJB piece or the EIS access piece of the application. The Web components are located on another platform over which workload manager has no jurisdiction. The performance goals for the request have to be set for EJB processing only, which will not map so well to service level requirements that may have been agreed to with the user community.

2.3.2 Availability

When we are dealing with a hybrid application, we introduce an additional application server and intervening communication channels into the infrastructure when compared to the reference configuration. There are more components that need to be configured for fault tolerance to ensure that availability requirements are met.

Each of the additions that the hybrid solution introduces into the infrastructure should be configured such that we do not compromise availability objectives. Things to consider include:

- Ensure that there is no single point of failure. Replicate or clone infrastructure components in a configuration that enables surviving components to assume the workload of a failed component to preserve application availability. This could be a hot standby or failover configuration, when a secondary backup of the component is normally idle but ready to take over processing responsibilities if the primary copy fails; or it could be a configuration in which multiple copies of a component process work requests all the time and merely take on the additional load when a component fails.
- If a workload balancing mechanism is used to distribute requests across a set of cloned components, ensure that any component failure is recognized such that no further requests are routed in that direction. Only functioning instances of the infrastructure component should be considered by the workload balancing mechanism.
- When a failed component is recovered, check whether it is automatically reinstated into the decision-making algorithm for workload balancing, or if manual intervention is required.
- Beware of any component “affinity” considerations that need to be accounted for. After an initial client request has been serviced, subsequent requests need to be routed to the same instance of the component. Such a condition might be imposed by the infrastructure, such as a connection to a CICS server over which an extended unit of work is in place; or it might be imposed by application design, such as a cached lookup to an enterprise bean. In these cases, following a failure and recovery it may not be possible to immediately resume a balanced distribution of requests across the component instances.

2.3.3 Security

Within the confines of a sysplex environment, everything runs under a RACF (or equivalent product) identity. All resource authorizations are performed using a RACF identity, either explicitly (such as access to DB2 data) or implicitly (such as J2EE EJBROLE checking). A work request running on z/OS needs to be assigned a RACF identity.

When the entire application is hosted on the z/OS platform, there are a variety of techniques for authenticating the end user:

- Basic authentication
- Client certificates or mutual authentication
- Form-based login
- Off-platform authentication (Trust Association Interceptor or Custom User Registry)

More options are described in *z/OS WebSphere and J2EE Security Handbook,* SG24-6846.

All these methods will result in a RACF identity being established for the end user. Once we have a verified identity, it may travel around with the request so that it may be used for authorization checks. In the reference configuration described in “Reference configuration” on page 37, the request never leaves the confines of the WebSphere Application Server for z/OS and OS/390 application server, so the authenticated user identity is available for the lifetime of the request in the sysplex (unless configured not to do so[4]). In addition, we can propagate the authenticated user’s identity on calls to the EIS resources.

In a hybrid solution, the user authentication would have taken place in the Web container of a WebSphere Application Server Advanced Edition server. z/OS will not trust the remote platform by default, and will in most cases not just accept a propagated user identity to run a request with. For IIOP requests, currently the only means of authentication with a WebSphere Application Server for z/OS and OS/390 EJB container for a non-z/OS client request is a client certificate. (The EJB container could assign a default user identity to the request, but these typically have been configured with very little processing authority.) The client certificate may be mapped to a RACF user identity, but this represents the identity of the remote WebSphere AE server, not the end user. Authorization checks now have to be set up to permit access for the remote server, rather than individual users.

Other solutions for the communications channel between Web components and EJB components in a hybrid deployment (JMS, Web Services) will likely suffer from a similar loss of granularity for authorization checks.Today, a good method for userid propagation would be a HTTP protocol implementation.

When accessing z/OS EIS resources from outside of the sysplex, we again lose the ability to propagate the identity of the authenticated user. Typically, these requests carry with them a user ID and password that are authenticated against RACF by the resource subsystem. The user identity once more represents the remote server or application rather than a specific user, leading to less granular resource authorization controls than those of the reference configuration.

Password protection issues for remote clients

Another aspect of remote client authentication for z/OS EIS access is the introduction of a password in order to be able to run work. Connection technologies usually flow a user identity and password with their requests. (In the z/OS-only environment of the reference configuration, only the user ID needs to be flowed; no password is required because the user has already been authenticated against RACF.) This existence of the password presents a possible security exposure.

The password may be hard-coded in a program, or stored in a properties file, or set up as part of the application server or connector configuration. Wherever it is stored, it will need to be protected in the remote client environment from unauthorized use.

2.3.4 Transaction integrity

In transactional applications, the integrity of resource updates is vital. When a client submits a request that involves updates to corporate resources, we must be certain whether or not the updates were successfully applied. In addition, we need to ensure that updates that form part of the same unit of work are either applied or not applied in their entirety. When a business transaction involves several update operations, they must all be committed or rolled back together. Failure to do so could result in the data being left in an inconsistent state, a condition which could have serious implications for the business operations.

[4] It is possible for the original user context to be “lost” if the application deployment descriptor specifies a *RunAs* mode of anything other than “Caller”.

When a transaction involves multiple updates, the unit of work may be qualified as one of two types: and extended unit of work, or a distributed unit of work. In both cases, update requests to the resource manager would need to be held in a pending state until instructions were received to either commit or roll back. The means or communication with the resource manager, as well as the connector technology and the capabilities of the resource manager itself, could all have a bearing on whether such transactional behavior is supported. When contemplating a hybrid solution, we need to assess whether or not the communications infrastructure for component interaction and EIS access will support the required levels of transactional integrity.

Extended unit of work

An extended unit of work involves multiple updates to the same resource manager. All the updates comprise the same unit of work, which means that they must all be committed or all rolled back. In Java terms, this situation is referred to as a *local transaction*.

The remote resource manager might be the EJB container running in a WebSphere Application Server for z/OS and OS/390 server. If the user request only involves a single update to an entity bean, then the update may be committed upon return from the call. If the user request involves updates to several entity beans, then an extended unit of work is required. The resource manager defers committing any updates until it receives an instruction to commit or roll back from the client part of the application (referred to as *syncpoint time*).

All the updates for the transaction are owned by a single resource manager. At syncpoint time the client may simply ask the resource manager to commit the updates, and the resource manager decides whether it is capable of committing them or needs to roll them back. Transaction integrity is maintained whatever decision the resource manager makes, because all the updates for the transaction are acted on in this one operation.

In summary, to support extended units of work, the resource manager needs to be able to defer committing updates until syncpoint time, but does not necessarily have to support a two-phase commit protocol.

Distributed unit of work

A distributed unit of work involves updates to multiple resource managers in order to complete a user transaction. As with an extended unit of work, all the requests must be committed or rolled back together. In Java terms, this is a *global transaction*.

The resource manager involved in the unit of work could include a WebSphere Application Server for z/OS and OS/390 EJB container, or database subsystems, or any subsystem accessed through J2EE Connector Architecture (JCA) connectors. It could also include the client subsystem itself—if an EJB container applies updates to an entity bean and then uses a JCA connector to run an update in CICS, we have a distributed unit of work, even though the EJB container is the client to the CICS call.

A distributed unit of work requires its participant resource managers to defer committing updates until syncpoint time. It also requires a transaction coordinator to ensure that all participant resource managers take the same syncpoint action (either commit or roll back). To accomplish this, the transaction coordinator must make use of a two-phase commit protocol to ensure data integrity.

1. Phase one asks all the participant resource managers if they are ready to commit. Each resource manager replies yes or no.
2. Phase two tells the resource managers what action to take based on the responses from phase one. If any resource manager is unable to commit, then a rollback instruction is issued. Otherwise, a commit instruction is issued.

If the application involves distributed units of work, then the participant resource managers need to be able to defer updates until syncpoint time, and support a two-phase commit syncpoint.

Note: WebSphere Application Server Enterprise V5 provides Last Participant Support. With this you can coordinate the use of a single one-phase commit capable resource with any number of two-phase capable resources in the same global transaction.

2.3.5 Infrastructure criteria

In the reference configuration described in “Reference configuration” on page 37, the application is pretty much self-contained. There will be some network infrastructure to direct the client request into the z/OS system, but once it has arrived, all intra-application communication is over secure, local connections until the response is sent back to the client.

When creating a hybrid solution, more elements need to be added to the infrastructure. Principally these elements are the extra application server for the Web components, and the network channels to permit communication with the EJB components in WebSphere Application Server for z/OS and OS/390. Additional elements may be required if access is required to EIS hosted on the z/OS system, or if communication with the EJB components is accomplished using Java Messaging Services (JMS), Web Services, or some other non-IIOP protocol.

When evaluating a hybrid solution, there are a number of considerations related to the additional infrastructure that need to be introduced.

Total cost of ownership

Adding elements into the infrastructure will increase the complexity and total cost of the solution. Additional costs incurred could include, but are not limited to, the following:

- License costs for the additional application servers, and associated maintenance contracts
- License costs for connector technologies to EIS resources
- License costs for additional systems management infrastructure, such as monitoring tools
- Hardware costs for additional server platforms
- Additional network infrastructure costs
- Additional skills required to support the infrastructure now that it spans multiple platforms
- Increased application development and testing costs

Note that a possible motivation for a hybrid solution could be to make more effective use of z/OS processing resources, the benefits of which should be offset against any additional costs.

Firewall considerations

In a hybrid solution, we are moving elements of the application off the z/OS platform. Some installations place a firewall in front of their sysplex through which all IP traffic must flow. We have to consider where to place the WebSphere Application Server Advanced Edition server which is hosting the Web components.integration and security considerations

If the infrastructure is configured with the firewall between the Web components and the EJB components, the following considerations apply:

- We must have some means of permitting requests to the WebSphere Application Server for z/OS and OS/390 and to z/OS EIS to reach their destination. This means that for connections that use an IP protocol, the firewall must be configured to let them through. For example, we may have to permit IIOP requests.
- There is a problem when routing IIOP requests through a firewall that operates using Network Address Translation (NAT). A request is originally submitted to the public IP address of the firewall, which then converts the destination address to that of the target server. The problem is that when a JNDI lookup request for an EJB component is issued by the client, the returned IOR contains the IP address of the target server. This address is useless to the client because it cannot access the server directly; what it needs is the public firewall address. The ideal solution would be for the firewall to perform the appropriate translation within the IOR as it flows back to the client, but currently there are no firewalls with this capability. A workaround is to configure the servers to use an unqualified symbolic host name instead of the IP address.
- If any services on the z/OS system need to be accessed using an IP protocol from the WebSphere Application Server Advanced Edition server, they need to have a predetermined port number. Some port numbers are generated dynamically, with the client being advised of the actual number when it needs to use it.

  An example of this is the port numbers used to service IIOP calls by the WebSphere Application Server for z/OS and OS/390 name server. In this case, the port number may be locked down using the IIOP Firewall Port and the SSL Firewall Port properties of the server instance. However, there may be other instances, possibly involving EIS resources, where the ability to lock down the port number is not available. In these cases, the firewall will need to be reconfigured to the generated port number every time it changes.

Given the inherent problems with driving IIOP requests through a firewall, it is worth considering placing the WebSphere AE server together with the z/OS system behind all the firewalls.

2.3.6 Development and deployment criteria

A hybrid solution requires the application to be packaged into two or more deployable artifacts, each of which is installed into a different application server. This makes the processes of application development, testing, packaging, and deployment more complex than they would be for a consolidated application targeted for a single application server. This section identifies some of these implications.

Application development

This section addresses issues that arise with the application design and development. An assessment of the feasibility of a hybrid solution should consider the amount of additional complexity which the solution will add to the development process.

Lookup namespace

When developing a hybrid deployment, the fact that we need communication between components in separate servers affects the application code. Consider the example of a hybrid deployment where EJB components are accessed from Web components using the RMI over IIOP. The Web components will need to perform a JNDI lookup in order to determine the location of the target enterprise bean.

A lookup is issued against an InitialContext object, which is used to identify the namespace to be used. In a single consolidated application, InitialContext is created using a default set of properties, which equates to the namespace of the same application server that the Web component is running in. When access is required to a component located in a remote application server, the lookup needs to be directed to the namespace of the remote application server (the one hosting the EJB components). In this case, the InitialContext is instantiated with the javax.naming.Context.PROVIDER_URL property to point to the correct namespace.

In order to support the migration of the application through its test environments, each of which may have the enterprise beans located in a different server, the PROVIDER_URL property should not be hard-coded into the program. One possible solution would be to specify it as an environment property, the value of which may be set appropriately at application assembly time.

com.ibm.CORBA.iiop.noLocalCopies

The com.ibm.CORBA.iiop.noLocalCopies JVM property determines whether objects passed to enterprise beans are passed by reference or passed by value. This support is not part of the J2EE 1.1 specs. It is within the J2EE 1.3 specification and therefore supported by WebSphere Application Server V5 for OS/390 and z/OS and implemented with the "local" interface support.

Note: APAR PQ57189, Service Level W401019 is required to provide this support. Refer to *Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836 for more information.

In a hybrid deployment, the ability to pass by reference (noLocalCopies=true) is lost, because the JVM boundary is crossed. Pass by reference enables the calling object to see changes made by the called object to the object passed as an argument. Therefore, the removal of pass by reference could change the behavior of the calling object.

In addition, there may be occasions when the client program is expecting to handle a particular exception, and the removal of noLocalCopies changes the exception that will be raised for a particular error condition. For example, in WebSphere Application Server V4.01 for z/OS and OS/390, the default configuration (noLocalCopies disabled) will return java.rmi.RemoteException for non-user exceptions. However, with noLocalCopies=true, specific exceptions may be returned to the client without having them wrapped in the java.rmi.RemoteException. If the program originally ran as a consolidated application with noLocalCopies=true, and it contained code to catch the specific exceptions, it may need to be amended once it has been converted to a hybrid application to catch the java.rmi.RemoteExceptions that may replace the original exceptions.

Code shared by an application's artifacts

An application that is being converted from a single to a hybrid deployment may make use of common utility classes which are shared between the Web and EJB components. In the hybrid solution, each application will need to be packaged with its own copy of the common code.

If the shared code has some dependency on the topology of the application, then consideration should be given creating separate copies of the common code. For example, there could be a common class to perform an EJB lookup that is used by both Web and EJB components. In a consolidated application, both sets of components perform their lookup in the same namespace. In a hybrid application, the Web components will need to look up in a remote namespace (that of the EJB container's application server), but any EJB components that look up other enterprise beans will look up in their local (default) namespace. The hybrid utility class needs to provide a different behavior to the Web and EJB components.

Modern J2EE applications are developed using tools that provide an integrated test environment in which code may be run. IBM's WebSphere Studio Application Developer V4 set of products provide such a facility. It is conceivable that early unit testing of a hybrid application will be performed using such an environment. This poses a problem for testing applications that use common classes with a topology dependency, such as a lookup utility. The tool's test environment only provides a single server, so the different behavior for the common class required by the Web and EJB components needs to be accommodated somehow.

Possible solutions involve adjusting the classloader options for the server so that the Web components locate their own copy of the class (packaged in the Web application), and the EJB components locate their own (different) copy of the class (packaged in the EJB application); or creating a separately named version of the class for each of the application packages to use.

Application assembly

Assessment of a hybrid deployment should include a review of the complexity of the application assembly, and the tools required for it. The IBM WebSphere Studio Application Developer V4 is gradually removing the need for a separate Application Assembly tool (though the tools will continue to be provided for those who do not have the benefit of the Studio tools). However, at the time of writing, the Application Assembly Tool for z/OS and OS/390 is still required to manipulate z/OS-specific parameter settings, namely:

- SyncToOSThread
- Connection Management Policy

If use is made of these parameters for the application components deployed to WebSphere Application Server for z/OS and OS/390, then separate tools are required to assemble each part of the application.

Note: WebSphere Application Server V5 for z/OS and OS/390 eliminates the need for z/OS-specific tooling.

Application deployment

The processes and tools required for the deployment of a hybrid application should be considered.

Deployment coordination

Application deployment in an integrated single server configuration deploys the whole application in one process. We do not have to worry about incompatibilities between the Web components and the EJB components.

With a hybrid deployment, the Web components and EJB components are deployed separately. When application updates are being implemented, the two sides may get out of step. If the EJB components are deployed first, workload distribution mechanisms would typically detect that the EJB application was back online, and start routing work to it. The original Web components will attempt to access the new EJB components.

If the enterprise bean remote interfaces have not changed, such that the Web and EJB components are still compatible, this may be an acceptable state of affairs; it depends on the nature of the application. However, if they are incompatible, the application could encounter numerous failures. Worse still, if the incompatibilities are more subtle and exist at the level of application functionality, corruptions and inconsistencies in the corporate data could be introduced.

The only way to ensure that this does not occur is to take the application offline. This could compromise availability requirements for the application.

Deployment tools

WebSphere Application Server Advanced Edition V4 and WebSphere Application Server V4.01 for z/OS and OS/390 use different deployment tools. Manual deployments are performed using the Administration Client for WebSphere AE, and using the Systems Management End User Interface for WebSphere on z/OS. The functions they perform are almost the same, but they are completely incompatible.

Similarly, automation tools with WebSphere Version 4 of the products are incompatible. WebSphere Application Server for z/OS and OS/390 used REXX and the Systems Management API; WebSphere AE uses XMLConfig and WSCP (the WebSphere Control Program).

With a hybrid deployment, we have to consider the extra complexity that is introduced into the application deployment process, whether deploying manually or using automated tools.

HTTP session data

In many Web applications, users dynamically collect data as they move through the site based on a series of selections on pages they visit. Where the user goes next, and what the application displays as the user's next page (or next choice) may depend on what the user has chosen previously from the site. For example, if the user clicks the checkout button on our site, the next page must contain the user's shopping selections.

In order to do this, a Web application needs a mechanism to hold the user's state information over a period of time. However, HTTP alone does not recognize or maintain a user's state. HTTP treats each user request as a discrete, independent interaction.

The Java servlet specification provides a mechanism for servlet applications to maintain a user’s state information. This mechanism, known as a session, addresses some of the problems of more traditional strategies such as a pure cookie solution. It allows a Web application developer to maintain all user state information at the host, while passing minimal information back to the user via cookies.

The accessibility of HTTP session data is not a problem in a hybrid solution. Session data is a resource that is associated with the Web container; the APIs to access session data are only available to Web components. So when the Web container is moved, the resource for storing the session data moves with it.

Session data may be maintained in-memory or persisted to a database. When an application is split into a hybrid solution, both types may be migrated with the Web container to another platform.

From a performance perspective it is expensive to persist HTTP session data to DB2. z/OS’s Parallel Sysplex implementation of DB2 makes it possible to share DB2 data between multiple operating systems. This takes a lot of pain away when one needs to implement a highly available infrastructure for HTTP session data, but it also adds some more pathlength for data sharing. We recommend that this kind of HTTP session implementation backs the sessions within DB2, but still keeps them in memory. This can be done by coding `session.persistenceversion=2` in the webcontainer.conf file.

DataSource references

Although accessibility to session data should not be a problem when converting a consolidated application to a hybrid application, care should be taken with the content of session data. For example, it is possible for a servlet to perform a lookup on a DataSource and store the returned handle for subsequent reuse in the session data. Where the Web container and the EJB container are co-resident in the same application server, the DataSource handle may be extracted from the session data by the servlet and passed to a session EJB, which may then use it for its own purposes. This would save the enterprise bean the cost of performing a lookup for the EIS DataSource. This is possible because the same DataSource is available to both the Web container and the EJB container.

When the Web and EJB containers have been separated, their separate application servers each have their own DataSource defined. The handle passed from the servlet to the session bean (which was provided by the remote application server to the Web component) will be meaningless in the EJB container environment. The session bean will need to look up its own DataSource.

2.3.7 Systems management

Systems management involves a broad range of activities. Many of these will be rendered more complicated by the conversion of an application to a hybrid deployment. When evaluating a hybrid solution, the cost of this added complexity should be assessed.

Performance monitoring

Performance monitoring is required to ensure that service levels are being maintained, and to detect possible problems with the infrastructure, such as a server going offline. If the application is deployed to a single server, an overall picture of the application performance, such as request response times and resource consumption, may be easily obtained. A hybrid deployment requires the consolidation of performance data from each of the application server platforms.

Capacity planning

As with performance monitoring, with a hybrid deployment, capacity planning data will need to be collected from all the relevant application platforms. An additional capacity planning concern is the communication channels between the application servers and EIS resources.

Failure alerts

The application needs to be monitored for failures, and alerts raised so that automatic or manual intervention may be taken to address the problem. With the application running on a single platform, errors from all components can be reported in the same place. However, with a hybrid deployment, errors may occur in either part of the application, so both platforms have to be monitored for error alerts.

In addition, once an error alert has been raised, problem determination will be more complex, with multiple logs and traces needing to be consolidated in order to obtain the end-to-end diagnostics.

Backup coordination

With a hybrid application being deployed over multiple application servers, the backing up and restoring of application code and server configurations needs to be coordinated. Failure to do so could result in an incompatibility between the application's components or infrastructure.

Server maintenance

When applying application server maintenance in a hybrid deployment, we have to ensure that the servers remain compatible and that all corequisite updates are applied. This might compromise application availability requirements, because the servers may have to be taken offline in order to avoid exposing the application on incompatible servers.

Operations

A hybrid deployment will introduce another platform, which may require additional skills to support it.

2.3.8 Strategic considerations

When evaluating the options for a hybrid solution, attention should be paid to current strategic trends in the Web application server industry. Some solutions may be better positioned than others to take advantage of emerging technologies. Conversely, technologies with a declining popularity or importance should be avoided.

As an example, consider the connection technologies to EIS resources. IBM originally developed the Common Connector Framework (CCF) to provide a generic programming model for access to EIS resources. More recently, the J2EE Connector Architecture (JCA) has evolved to provide the same function. JCA is based on the CCF, and although at first glance there seem to be many similarities between the two, the programming models are incompatible. The strategic direction, which modern application servers and development tools support, is the JCA.

3

Component interaction characteristics

When an application is deployed into separate application servers, communication between some of the components and EIS resources could change form local to remote. In this chapter we investigate the implications of this change by examining the characteristics of different types of connection within the application.

We first describe the connection technologies to be investigated in 3.1, “Connection types” on page 76. Each of these connection types is then subjected to a detailed study of the changes that will be introduced by a conversion from a consolidated application to a hybrid deployment. The changes have been categorized into the various qualities of service which were described in 2.3, “Evaluation criteria for remote component and EIS access” on page 58. A summary table is provided in 3.1.1, “Cross-reference table” on page 76 to provide a quick reference to each of the evaluation criteria for each of the connection types.

3.1 Connection types

When you have an application that has been deployed across two separate application servers, a number of communication channels may exist between the two servers, or between the platforms on which the servers have been installed. This chapter examines the characteristics of some of these types of connections, but those discussed should not be considered to be a definitive list. They are described here for illustrative purposes. Other types of connections may need to be set up, and they should each be given careful consideration from the perspective of each of the characteristics described in 2.3, “Evaluation criteria for remote component and EIS access” on page 58.

The types of connection which will be considered are:

- RMI/IIOP access in 3.2, “RMI/IIOP access to remote enterprise beans” on page 76.
- JDBC access in 3.3, “JDBC access to DB2” on page 82.
- JCA access to CICS in 3.4, “JCA access to CICS” on page 90.

3.1.1 Cross-reference table

Table 3-1 provides a reference of connection characteristics for each connection type.

Table 3-1 Cross-reference table

Connection	RMI/IIOP	JDBC	JCA
Performance	3.2.1, “Performance” on page 77	3.3.2, “Performance” on page 83	3.4.2, “Performance” on page 92
Availability	3.2.2, “Availability” on page 78	3.3.3, “Availability” on page 84	3.4.3, “Availability” on page 93
Security	3.2.3, “Security” on page 78	3.3.4, “Security” on page 84	3.4.4, “Security” on page 94
Transaction integrity	3.2.4, “Transaction integrity” on page 79	3.3.5, “Transaction integrity” on page 87	3.4.5, “Transaction integrity” on page 95
Infrastructure	3.2.5, “Infrastructure” on page 79	3.3.6, “Infrastructure” on page 87	3.4.6, “Infrastructure” on page 96
Development and deployment	3.2.6, “Development and deployment” on page 80	3.3.7, “Development and deployment” on page 87	3.4.7, “Development and deployment” on page 96
Systems management	3.2.7, “Systems management” on page 81	3.3.8, “Systems management” on page 90	3.4.8, “Systems management” on page 99
Strategic considerations	3.2.8, “Strategic considerations” on page 81	3.3.9, “Strategic considerations” on page 90	3.4.9, “Strategic considerations” on page 99

3.2 RMI/IIOP access to remote enterprise beans

This section outlines the considerations for a “default” separation of the Web container from the EJB container, using Remote Method Invocation (RMI) over Internet Inter-ORB Protocol (IIOP) as the communications medium between Web and EJB components. This is the means by which methods on enterprise beans are invoked.

3.2.1 Performance

RMI/IIOP is used for all access to enterprise beans, but optimizations are available to reduce the overhead of the call when the enterprise bean and its client are located in the same JVM. WebSphere Application Server for z/OS and OS/390 automatically optimizes calls from clients to enterprise beans to local program calls, but there are also explicit controls. One example is the com.ibm.CORBA.iiop.noLocalCopies JVM property, which if set to "true" (or actually any non-null value) causes arguments on the call to be passed by reference rather than by value.

Another example is introduced by the EJB 2.0 specification with the concept of a Local interface to a bean (in addition to the conventional Home and Remote interfaces). This optimizes the call to the enterprise bean by circumventing some of the inter-ORB processing, which is not required because the components involved are running in the same ORB.

In a hybrid deployment, components could be running in separate application servers, which means they are also in separate JVMs. The capability for any of the RMI/IIOP call optimizations has been removed.

Workload Manager

In a hybrid deployment, there is a loss of flexibility in the way that the z/OS Workload Manager can manage the WebSphere Application Server for z/OS and OS/390 workload. The reason for this is as follows.

The way by which work requests are managed in a z/OS environment from a performance perspective is determined by the *enclave* which is associated with the request. An enclave represents a workload manager service class, and when associated with a request it determines the priority of the server to which the request should be dispatched.

WebSphere Application Server for z/OS and OS/390 has a set of rules for associating an enclave with a work request. One such rule applies to HTTP and HTTPS requests that arrive at the J2EE server. These rules are defined in the environment file current.env, with the following relevant parameters:

- BBOC_HTTPALL_TCLASS_FILE=file name - identifies a file containing rules for classifying HTTP and HTTPS requests.
- BBOC_HTTP_TRANSACTION_CLASS=transaction class - defines the default class for HTTP requests.
- BBOC_HTTP_SSL_TRANSACTION_CLASS=transaction class - defines the default class for HTTPS requests.

The format of an entry in the BBOC_HTTPALL_TCLASS_FILE file is:

```
TranClassMap <host>:<port> <uritemplate> <tclass>
```

where:

<host>	The host name of the request header
<port>	The port number to which the request was submitted
<uritemplate>	The URI of the request
<tclass>	The workload manager transaction class used for the enclave

From this we can see that we have a lot of flexibility for classifying HTTP and HTTPS requests that arrive at the system.

When the request arrives in the EJB container, it will not be subject to these rules, because it will travel over IIOP rather than HTTP or HTTPS. This limits our ability to assign a workload manager service class to the request; the only tools we have left are the classification rules which we can define in the workload manager ISPF configuration panels. These classification rules may only classify work based on the J2EE server or the user identity on the request. So by submitting the request over IIOP we have lost some granularity for classifying work requests.

3.2.2 Availability

With an integrated application deployed to a single application server, the consequences of a server failure are simplified, because all J2EE application components reside in the same server. The WebSphere Application Server for z/OS and OS/390 infrastructure and workload manager cooperate to maintain an available service. Servers are started when required, and may be distributed across z/OS images in the sysplex. When a server fails, work requests are routed to surviving servers. When the failed server is recovered, it is reinstated into the routing algorithm.

When the application is separated, and application server failure will only render part of the application unavailable, we have to think about the fate of any inter-component relationships which may have been set up. For example, a Web component may have performed a lookup on an enterprise bean. The lookup will resolve to a specific instance of the bean running in a specific server. We have the possibility of a failure in the EJB container application server, which will result in an error being returned to the Web component when it attempts to use its bean.

If the application remains consolidated in a single WebSphere Application Server for z/OS and OS/390 server, then both the Web and EJB containers will have failed. Workload manager will route the request to an alternate server, where it may run successfully if session data has been configured to tolerate a server failure.

This is why it is a good idea to minimize the number of remote calls between the application servers. Not only do we minimize the calls that carry the extra performance overhead of being remote instead of local, but we also minimize the number of opportunities for such errors to occur.

3.2.3 Security

J2EE authorization checks are performed against enterprise bean methods according to a specification in the deployment descriptor. The subject of the authorization check is an EJBRole, which is a logical representation of the capabilities of the user. In the z/OS environment, the way to associate a user with an EJBRole is to grant the user identity access to the RACF EJBROLE or GEJBROLE profile.

In a consolidated solution, the user will have authenticated in the WebSphere Application Server for z/OS and OS/390 Web container, and a RACF user identity will have been associated with the user request. The associated user identity is carried with the request when an enterprise bean method is called. The enterprise bean method may be run under one of three user identities, depending on the specification of the RunAs parameter (which is also in the application deployment description):

- RunAs(Caller) - the identity of the authenticated user.
- RunAs(Server) - the identity associated with the application server in which the request is running.
- RunAs(Role) - the identity associated with an EJBROLE RACF profile.

RunAs(Caller) is the default setting, and while this option is selected, all downstream processing within the EJB container will run under the identity of the authenticated user. Additionally, when the deployment descriptor indicates that EJBRole authorization checks should take place, it is the authenticated user identity which needs to have been granted access to an appropriate RACF EJBROLE/GEJBROLE profile.

In a hybrid deployment, the user will have been authenticated in the Web container of the WebSphere Application Server Advanced Edition running on a non-z/OS platform. The authentication would have been against a security respository that is external to the sysplex. Unlike the integrated WebSphere Application Server for z/OS and OS/390 solution, it is impossible to propagate the user identity because the request flows from the Web container to the EJB container. The EJB container will need to authenticate the remote Web container, and with WebSphere Application Server V4.01 for z/OS and OS/390 the only means to accomplish this is by using a client certificate. The implication of this is that the request will run under the identity that has been associated with the remote application server certificate; the request is no longer running under the authenticated user identity.

The loss of the authenticated user identity on the request means that authorization checks are less granular. Resource access has to be granted to the identity of the Web container's application server. All invocations of EJB methods from the Web container will run under the same identity, so the loss of granularity of authorization checks extends to enterprise bean methods. It is the common user identity of the Web container which needs to be granted access to EJBROLE profile in order to permit execution of a bean method.

3.2.4 Transaction integrity

RMI/IIOP supports transactional integrity, so remote calls to enterprise beans are able to participate in distributed units of work.

3.2.5 Infrastructure

By creating a hybrid solution, a network connection has to be established over which the RMI/IIOP requests may flow. This process is more complex if a firewall is situated between the J2EE components.

Some firewall environments use Network Address Translation (NAT). NAT receives packets for one IP address and translates the headers of the packet so it can be sent to a second IP address. RMI/IIOP contains IP addresses embedded in the body of the IP packet, which are not translated, making the packet useless. Where this problem exists, some form of IIOP tunnelling must be implemented that permits the IIOP requests to flow over an HTTP transport.

IIOP tunnelling involves:

1. Wrapping the IIOP packet inside an HTTP request.
2. Sending the HTTP request to a tunnelling servlet on WebSphere Application Server for z/OS and OS/390.
3. Unwrapping the IIOP packet from the HTTP request, and forwarding the IIOP request to the target EJB server.
4. The servlet waits for the EJB component's reply and sends the reply to the client.

There are some trade-offs in using this feature. The obvious one is that we have reinstated Web container functionality in the WebSphere Application Server for z/OS and OS/390 server. Another is performance. The HTTP protocol is not a guaranteed persistent connection, as is the case for a TCP/IP (socket) connection. Therefore, a new HTTP connection is established for each request. This results in slower performance.

Using HTTP requests to send data between the plug-in and the application server through the firewall requires opening only one port. Opening firewall ports to permit the IIOP protocol is less desirable, as they are often more complex to set up, and the protocol switching overhead can impact performance.

3.2.6 Development and deployment

The tool we shall consider is WebSphere Studio Application Developer V4 (WSAD V4). WSAD V4 supports end-to-end development, testing, and deployment of e-business applications. The new WebSphere Studio products are designed from the ground up to meet the requirements for all new types of applications. These requirements include open standards, Java, XML, Web services, testing, varying levels of integration with other components and ISV products, pluggability, expandability, role-based development, increased usability for all users, enhanced team support, as well as increased speed to market. It provides integrated development tools for all e-business development roles, from Web developers to Java developers to business analysts to architects to enterprise programmers.

Application development

WSAD V4 provides facilities to develop and test all aspects of a J2EE application, including Web components and EJB components. WebSphere Studio Application Developer Integration Edition V4.1.1 adds the capability to develop and test applications that make use of EIS connectors. They both include a built-in test environment to which the application may be deployed and unit tested.

Enterprise beans are located by means of a lookup issued against a JNDI namespace. By default, the initial context for the lookup is in the namespace of the application server in which the component is running. When our Web component needs to access an enterprise bean in another application server, we need to steer it to the namespace in which the bean has been registered. This is done by passing a property table (java.util.Properties class) into the constructor for the InitialContext. The property javax.naming.Context.PROVIDER_URL is set to a string that identifies the location of the JNDI namespace we should be searching.

To avoid having to make program changes as the application is migrated through its test environments, we recommend that the value for this property is not hard-coded, but is retrieved from a variable which may be set at application assembly time, such as an environment variable.

Application assembly

When J2EE applications are developed, they are created in an Enterprise Application Project. Within the Enterprise Application Project, other projects may be created to contain the J2EE components. For example, an EJB project stores EJB components such as session enterprise beans and entity enterprise beans, and a Web Project stores Web components such as servlets and JSP files. When an application is being developed for a single server deployment and is ready for testing, the Enterprise Application Project will contain all of the application component projects, and the whole thing will be published to the built-in test environment. Unit testing may be performed. Then, when the application is ready, the application is packaged into an enterprise archive (.ear) file ready to be deployed to the next stage of testing.

For a hybrid deployment, we need separate packages. For the Web components, we have a choice of creating a Web archive (.war) file or an enterprise archive (.ear) file. When only one .war file is involved, there is not much to choose between the two options. However, when there are multiple .war files involved, these would require multiple installs, whereas they could all be packaged into a single .ear file and installed in one operation.

In WSAD V4, we can create additional Enterprise Application projects to reflect the separation of the application. We do not have to dispense with the original Enterprise Application project that contained the complete application; they can coexist. This is shown in Figure 3-1.

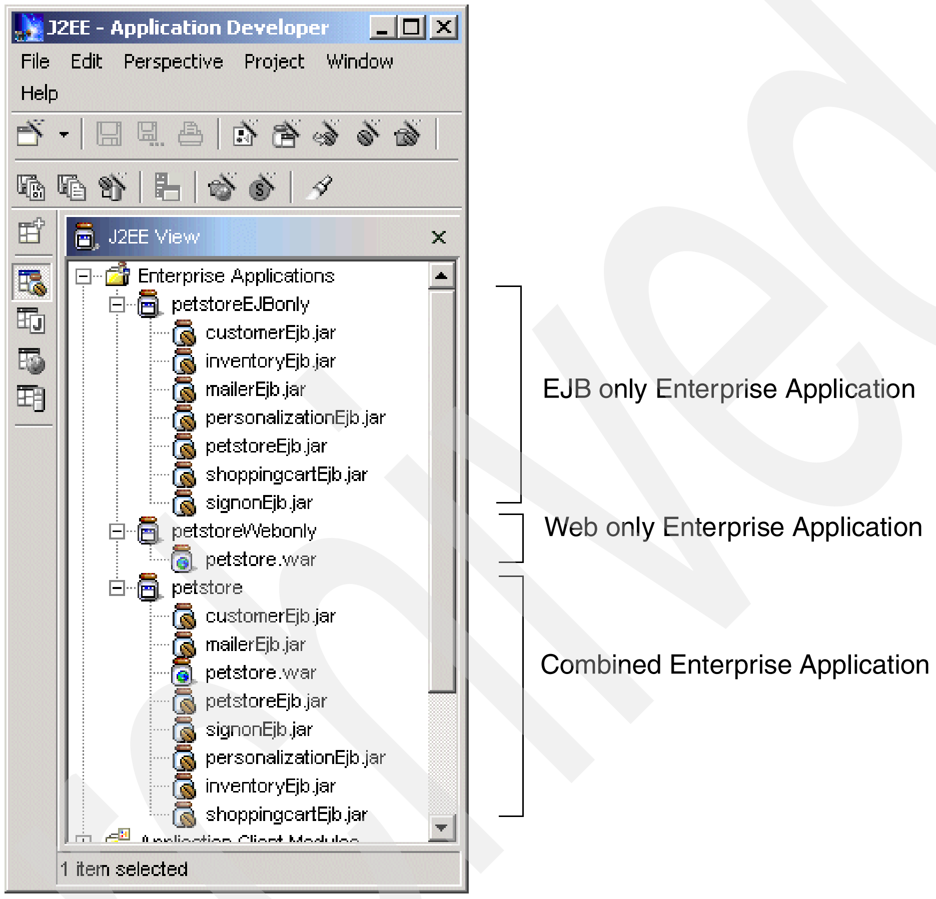


Figure 3-1 Combined and separated enterprise applications

3.2.7 Systems management

A hybrid deployment requires a more controlled process of application deployment. When deploying a single consolidated application, the Web components, the EJB components, and all the stubs and ties required to enable the enterprise bean access are deployed at the same time. A hybrid solution consists of two applications, the deployment of which needs to be coordinated in order to avoid incompatibilities between the application parts.

3.2.8 Strategic considerations

Application development tooling and infrastructure are geared up towards producing applications to be deployed to a single (cloned) server. Several compromises and additional development activities need to be introduced for a hybrid application, particularly if the RMI/IIOP calls are required to flow through a firewall. An alternative means if inter-component communication that does not present so many infrastructure problems might be considered.

RMI/IIOP itself is a well-established protocol, and forms part of the J2EE specification. In fact, the more loosely coupled solutions, such as JMS or Web Services, will still be using RMI/IIOP under the covers when the enterprise bean logic is invoked.

3.3 JDBC access to DB2

Database resources are accessed using the Java Database Connectivity (JDBC) API. Web components are capable of issuing JDBC and SQLJ calls to the same databases that the EJB components use. If the Web container and the EJB container are separated, then the database will be remote from at least one of them. (It may of course already be remote from both of them if it is hosted on a separate server.)

For the purpose of discussion, we shall make the following assumptions:

- In the original configuration, the Web container and the EJB container resided in the same application server, and the DB2 database ran on the same machine.
- In the new configuration, the Web container has been moved to a remotely located application server.
- The database remains in its original location, on the same machine as the EJB container.

The solutions that were considered for accessing the DB2 database from a Web container component were:

- Move or copy the database to a location which the Web container could access locally. This solution would work and give you the best performance, but is really only applicable if only the Web container requires access to this data. If the EJB components need access to the same data, then all we have succeeded in doing is to move the problem from the Web container to the EJB container. This consideration applies not only to the present day, but future requirements for enterprise bean access to this data need to be anticipated.
- Use DB2 Connect to provide a Distributed Relational Data Architecture (DRDA®) connection to DB2. The database remains in its original location, and a remote connection to it is established from the Web container. This is the recommended approach.
- Make use of a federated database. A federated database is a multi-database configuration that provides access to data resources which reside on multiple different platforms. All the data may be viewed as though it were local. A federated database uses DB2 Connect under the covers, and so offers no real advantages over option Ê.
- Use the new Universal JDBC Driver that comes with DB2 UDB Version 8, which is a Type 4 JDBC driver. A Type 4 driver is pure Java and requires no other client product software to be installed in order to establish a JDBC connection to a database. At the time of writing, this driver does not support the Java Transaction API (JTA), which means that it does not possess two-phase commit capability to participate in distributed units of work. For this reason, the Universal JDBC Driver was not considered a viable solution at this time.

The option that we selected to study, and implement in our laboratory exercises, was to use DB2 Connect.

3.3.1 DB2 Connect

Three editions of DB2 Connect are available:

- DB2 Connection Personal Edition

DB2 Connect Personal Edition makes host data available directly to a workstation. It is well suited for:

 – Typical two-tier design applications
 – Environments where native TCP/IP support is available, and an intermediate server is not desirable due to lack of administration skills
 – Situations where very large result sets are expected by an application

- DB2 Connect Enterprise Edition

  Connects LAN-based workstations and their desktop applications to mainframe and minicomputer host databases. It is well suited for:

  – Environments where the host system does not support native TCP/IP
  – Applications that need support for Java Applets
  – Web applications (for example, Microsoft IIS, Net.Data®, WebSphere, and NetDynamics)
  – TP Monitor applications (for example, CICS, Encina®, Microsoft, Transaction Server, and MQ Series)
  – Multitier applications

- DB2 Connect Unlimited Edition

  This edition is functionally equivalent to DB2 Connect Enterprise Edition, but with a different licensing agreement.

DB2 Connect Enterprise Edition is the version recommended for use with WebSphere Application Server Advanced Edition. For more information on DB2 Connect, refer to one of the Quick Beginnings guides such as *IBM DB2 Connect Enterprise Edition for UNIX V7 Quick Beginnings,* GC09-2952.

3.3.2 Performance

Applications that issue JDBC calls on WebSphere Application Server for z/OS and OS/390 communicate with a local instance of a DB2 server running in the same z/OS image. Scalability is accomplished by cloning application servers across z/OS images, and configuring DB2 in data sharing mode. The z/OS workload manager takes care of distribution of the work across the application servers.

For JDBC calls issued from a remote application server to a DB2 server running on z/OS, the entry point of the work into the sysplex is different. It arrives directly into DB2 via DB2 Connect. DB2 Connect Enterprise Edition supports integration with the Parallel Sysplex facilities workload balancing and fault tolerance functions. It accomplishes this by connecting at the level of the DB2 data sharing group.

Whenever a connect request is completed, DB2 for OS/390 sends the network addresses of all of the participants in the data sharing group to the DB2 connect server. The list is either a list of SNA LU names or a list of IP addresses depending on whether the initial connect was for a TCP/IP node or an APPC node. Each entry in the list also includes priority information based on advice received from the z/OS workload manager. Whenever further connect requests are received, DB2 Connect tries each of its cached network addresses in turn, in descending order of priority. In this way, connections may be distributed across the DB2 instances according to recommendations from the workload manager.

3.3.3 Availability

On z/OS, if the DB2 server in the same image as the application server fails, then the application server itself fails. The z/OS workload manager will route work away from the failed server to the surviving images.

The DB2 Connect Enterprise Edition support for the Parallel Sysplex feature described in 3.3.2, “Performance” on page 83 also provides availability services. When determining which DB2 for OS/390 instance in the data sharing group to target for a connect request, DB2 Connect examines its cached list of prioritized addresses. If a server is not available, then it tries the next one on the list. It continues this process until a successful connection is established. Connect errors will only be returned to the client application if all possible connections are unavailable.

Once a failed server has been recovered, it is automatically reincorporated into the connection decision process.

3.3.4 Security

When WebSphere Application Server for z/OS and OS/390 applications issues JDBC calls to DB2 from a servlet, all access is performed under one of two identities:

- The identity of the user ID which was provided on the getConnection() call. The code to accomplish this is illustrated in the code fragment in Example 3-1. For this identity to be recognized, the resource reference must specify a resource authentication of “Container”.
- Under all other circumstances, the request is issued under the identity of the application server region.

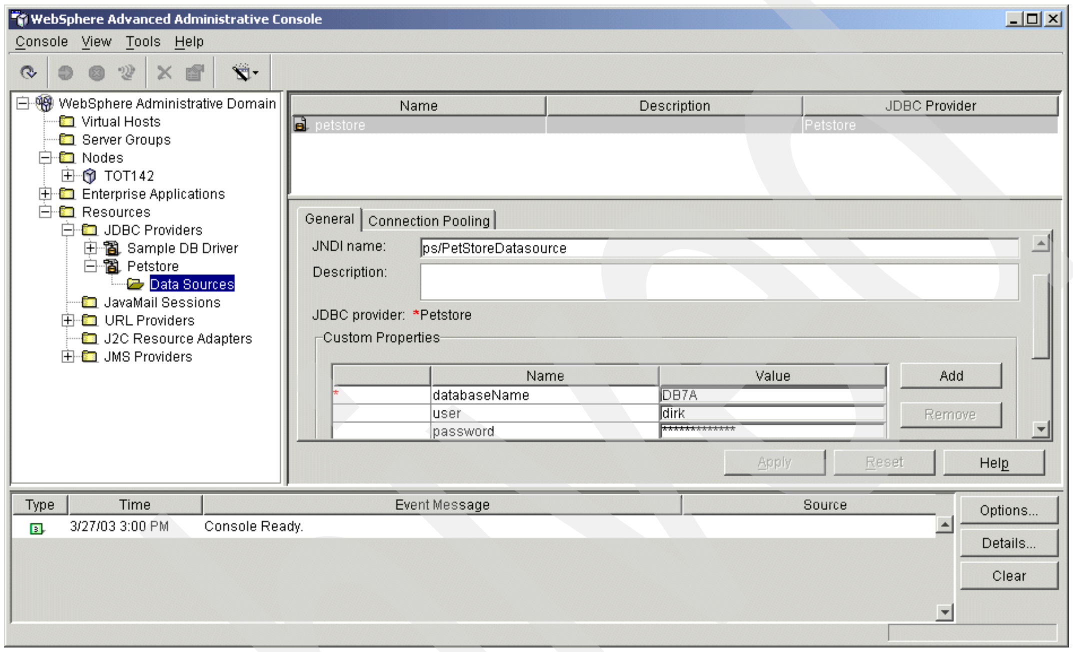
Example 3-1 Code and deployment descriptor requirements for application authentication on a DataSource

```
Code fragment:

String userid = “thomas”;
String password = “matthew”;

ctx = new IntialContext();
DataSource ds = (DataSource)ctx.lookup(“java:comp/env/jdbc/DB390”);
Connection conn = ds.getConnection(userid, password);

Deployment Descriptor web.xml snippet:

<resource-ref id=”ResourceRef_1”>
   <description>no description</description>
   <re-ref-name>jdbc/DB390</ref-ref-name>
   <res-type>javax.sql.DataSource</res-type>
   <res-auth>SERVLET</res-auth>
</resource-ref>
```

The second option is recommended; it avoids the problems of managing and securing the user ID and password details in the application code. With this option, the request runs under the identity of the WebSphere Application Server for z/OS and OS/390 J2EE server region. This identity could have been assigned to the region as a result of the STARTED class in RACF. This facility enables a user ID to be associated to a z/OS started procedure. If this is the only use of this particular user ID, there may be no password associated with it. Association with a started task is the only way that anything can run under this identity.

When the DB2 on OS/390 system is being accessed remotely using DB2 Connect, a user ID and password must be provided. Again, we have choices for the location of their specification:

1. As with WebSphere Application Server for z/OS and OS/390, a user ID and password can be supplied by the application code on the getConnection() call.
2. The user ID and password may be supplied on the DataSource definition, as shown in Figure 3-2.



Name	Description	JDBC Provider
petstore		Petstore

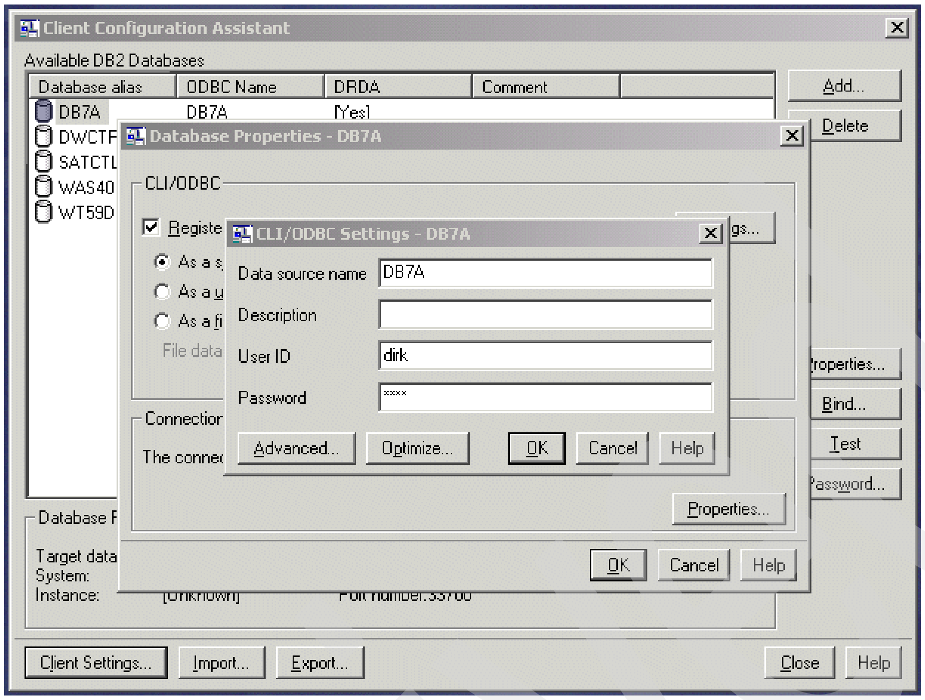
JNDI name: ps/PetStoreDatasource
Description:
JDBC provider: *Petstore
Custom Properties

	Name	Value
*	databaseName	DB7A
	user	dirk
	password	*************

Type	Time	Event Message	Source
	3/27/03 3:00 PM	Console Ready.	

Figure 3-2 User credentials defined on a DataSource

3. The user ID and password may be supplied on the DB2 Connect DataSource definition, as shown in Figure 3-3 on page 86.



Database alias	ODBC Name	DRDA	Comment
DB7A	DB7A	[Yes]	
DWCTF			
SATCTL			
WAS40			
WT59D			

Data source name	DB7A
Description	
User ID	dirk
Password	****

Figure 3-3 User credentials on a DB2 alias

Any of these three sources of user credentials may be employed when the resource reference in the Web application deployment descriptor specified application authentication. The order of preference for which credentials are actually used is as shown here: the credentials specified on getConnection(user, password); if getConnection() is coded with no user credentials, then the DataSource credentials are used; if the DataSource credentials are not specified, then the DB2 Connect alias credentials are used.

However, all three options require both a user identifier and a password to be defined, so that they can be flowed on the request and authenticated by DB2 for OS/390. If the access privileges to DB2 data are to remain unchanged, the user identity to be flowed must be that of the WebSphere Application Server for z/OS and OS/390 application server region. But we also need to flow an accompanying password, which could mean assigning a password to the server region user profile where it previously did not have one.

The application server user identity could potentially have access to significant amounts of production data. Relaxing its security by assigning it a password and storing it in a J2EE application, or a WebSphere Application Server Advanced Edition configuration, or a DB2 Connect configuration, is not recommended.

Without a password, the only way to run work under the server identity is to associate it with a started procedure. The ability to start procedures is usually well controlled, and the association of a runtime identity needs to be configured by security administrators, so the use of this user identity is well controlled and secure. With a password, any party who acquires the user ID and password credentials could submit work into the system, so we have an increased security exposure. In addition, we have increased systems management concerns to protect the password from unauthorized access and deal with any expired password and revoked user concerns.

The approach we recommend is to define a separate user and password for the JDBC connection to use, and authorize the appropriate DB2 resources for access by this new userid.

3.3.5 Transaction integrity

The JDBC driver installed into WebSphere Application Server for z/OS and OS/390 supports global transactions. These are transactions that are coordinated using a two-phase commit protocol, and are defined within servlets using the Java Transaction API (JTA). By using this API the servlet is able to demarcate its transaction boundaries, and at syncpoint time coordinate its updates to all resource managers that support an XAResource interface. DB2 supports this interface, so it can participate in global transactions.

When the Web components are moved to another platform with DB2 Connect introduced into the configuration, the servlet and JDBC driver capabilities remain the same; they both support the JTA. DB2 Connect also support JTA, so we retain the capability to participate in global transactions.

3.3.6 Infrastructure

DB2 Connect requires a connection between the remote client (the Web components running in a WebSphere Application Server Advanced Edition server) and the DB2 subsystem. DB2 Connect uses Distributed Relational Data Architecture (DRDA) to access distributed data across a network.

For versions of DB2, 5.1 or higher, the DRDA may be flowed across an SNA connection or a TCP/IP connection.

DB2 Connect will require licensing on the WebSphere Application Server Advanced Edition platform.

3.3.7 Development and deployment

Database connections

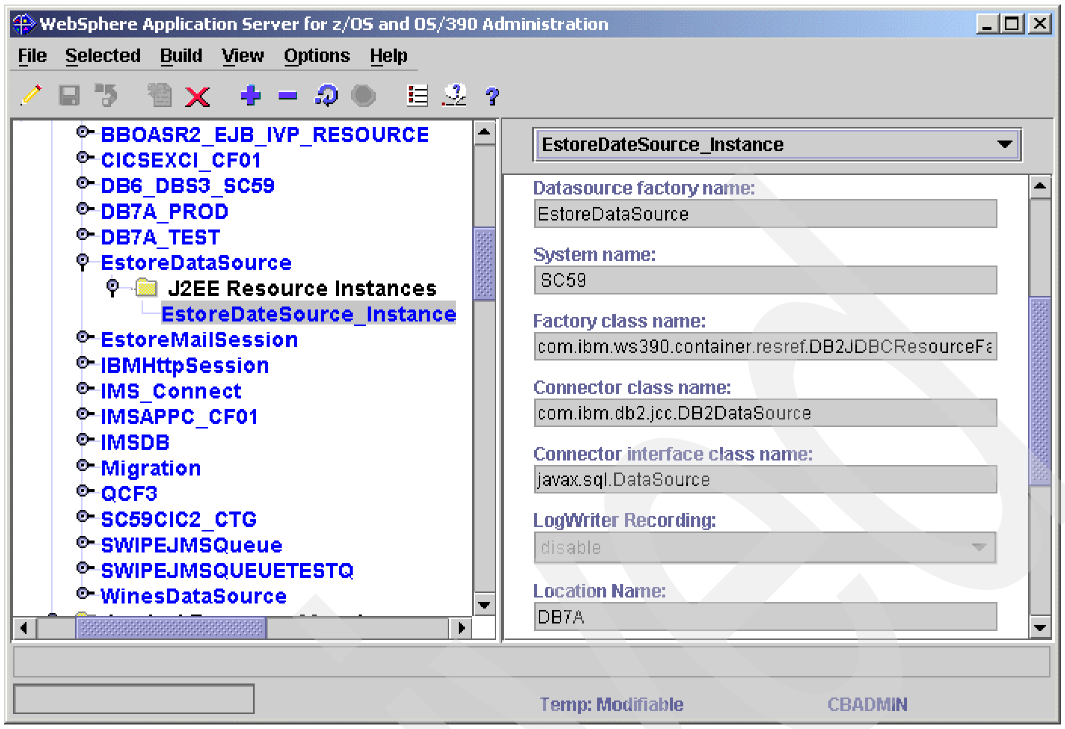
Connections to the database are provided by a JDBC DataSource. The DataSource is obtained as the result of a JNDI lookup. Example 3-2 illustrates a code snippet to accomplish this.

Example 3-2 Obtaining a connection from a DataSource

```
InitialContext ic = new InitialContext();
datasource = (DataSource) ic.lookup("java:comp/env/jdbc/EstoreDataSource");
Connection dbConnection = datasource.getConnection();
```

The lookup string is prefixed with “java:comp/env”, which means that there is a resource reference in the deployment descriptor for the application. The resource reference is resolved at application deployment time to bind the reference used in the application code to a real database. The location of the database is specified in a DataSource configuration. A DataSource may be defined on WebSphere Application Server for z/OS and OS/390, WebSphere Application Server Advanced Edition, and WebSphere Studio Application Developer V4. In each case, the definition resolves to the appropriate database for the application server environment. There will be no changes required to the application code when porting the application from the test environment built into WebSphere Studio Application Developer V4 through to production implementation on WebSphere Application Server for z/OS and OS/390.

In WebSphere Application Server for z/OS and OS/390, the DataSource points to the DB2 database on the z/OS image. Some of the DataSource attributes are shown in Figure 3-4 on page 88. On WebSphere Application Server for z/OS and OS/390, the identity of the target database is determined by the DataSource Location Name field.



WebSphere Application Server for z/OS and OS/390 Administration

File Selected Build View Options Help

- BBOASR2_EJB_IVP_RESOURCE
- CICSEXCI_CF01
- DB6_DBS3_SC59
- DB7A_PROD
- DB7A_TEST
- EstoreDataSource
  - J2EE Resource Instances
    - EstoreDateSource_Instance
- EstoreMailSession
- IBMHttpSession
- IMS_Connect
- IMSAPPC_CF01
- IMSDB
- Migration
- QCF3
- SC59CIC2_CTG
- SWIPEJMSQueue
- SWIPEJMSQUEUETESTQ
- WinesDataSource

EstoreDateSource_Instance

Datasource factory name: EstoreDataSource

System name: SC59

Factory class name: com.ibm.ws390.container.resref.DB2JDBCResourceFa

Connector class name: com.ibm.db2.jcc.DB2DataSource

Connector interface class name: javax.sql.DataSource

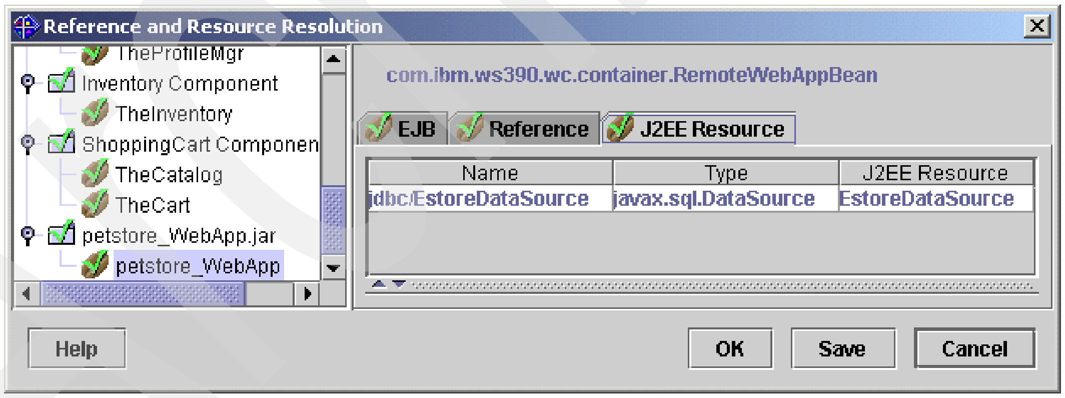
LogWriter Recording: disable

Location Name: DB7A

Temp: Modifiable CBADMIN

Figure 3-4 DataSource on z/OS attributes

The resource reference for /jdbc/DB390DataSource is resolved to this DataSource during application installation. This is illustrated in Figure 3-5.



Reference and Resource Resolution

- TheProfileMgr
- Inventory Component
  - TheInventory
- ShoppingCart Componen
  - TheCatalog
  - TheCart
- petstore_WebApp.jar
  - petstore_WebApp

com.ibm.ws390.wc.container.RemoteWebAppBean

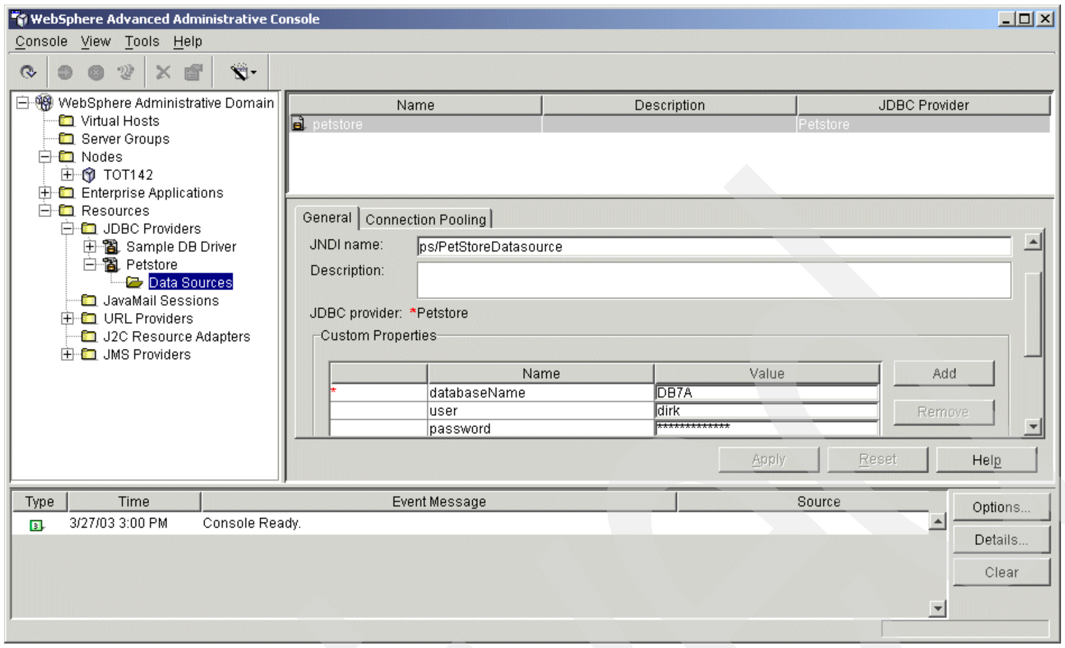
EJB | Reference | J2EE Resource

Name	Type	J2EE Resource
jdbc/EstoreDataSource	javax.sql.DataSource	EstoreDataSource

Help OK Save Cancel

Figure 3-5 Resource reference for a DataSource on z/OS

When the Web component is moved to a WebSphere Application Server Advanced Edition server, a similar DataSource needs to be defined for the resource reference to be resolved to, as shown in Figure 3-6 on page 89.



Name	Description	JDBC Provider
petstore		Petstore

JNDI name: ps/PetStoreDatasource

Description:

JDBC provider: *Petstore

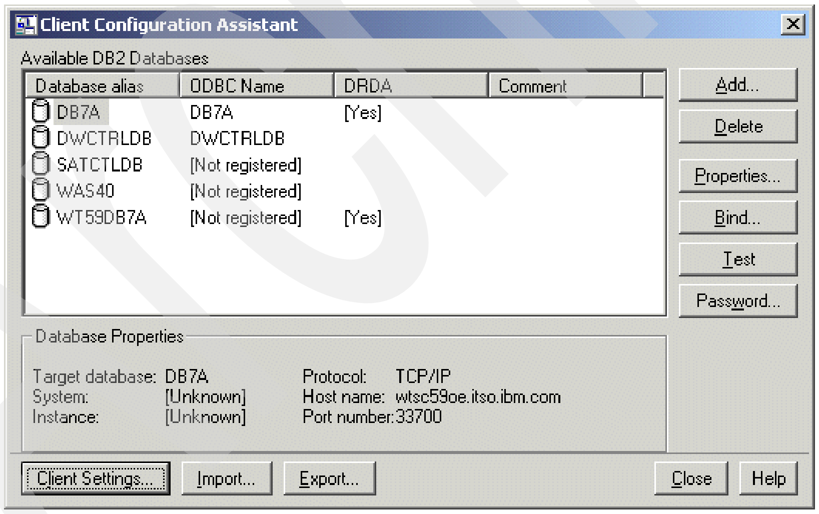
Custom Properties

	Name	Value
*	databaseName	DB7A
	user	dirk
	password	*************

Type	Time	Event Message	Source
	3/27/03 3:00 PM	Console Ready.	

Figure 3-6 DataSource on WebSphere Application Server Advanced Edition

In this case, the databaseName parameter refers to a database alias entry which DB2 Connect is able to map to the original DB2 database on z/OS. In this case, the alias name is the same as the name of the target database. This mapping is shown in the DB2 Client Configuration Wizard screen capture of Figure 3-7.



Available DB2 Databases

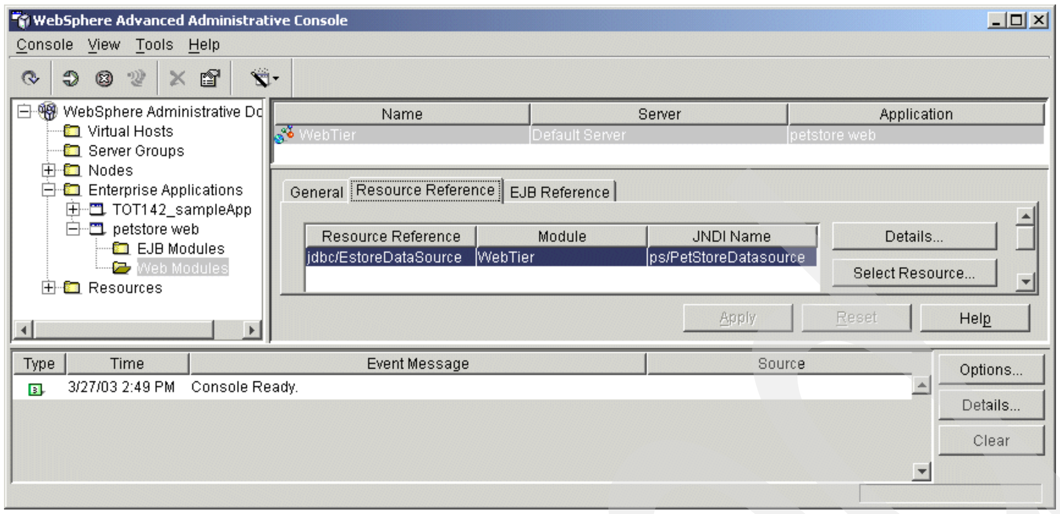
Database alias	ODBC Name	DRDA	Comment
DB7A	DB7A	[Yes]	
DWCTRLDB	DWCTRLDB		
SATCTLDB	[Not registered]		
WAS40	[Not registered]		
WT59DB7A	[Not registered]	[Yes]	

Database Properties

Target database:	DB7A	Protocol:	TCP/IP
System:	[Unknown]	Host name:	wtsc59oe.itso.ibm.com
Instance:	[Unknown]	Port number:	33700

Figure 3-7 DB2 Connect alias definition

During application installation, the resource reference is resolved to the DataSource; see Figure 3-8 on page 90.



Name	Server	Application
WebTier	Default Server	petstore web

Resource Reference	Module	JNDI Name
jdbc/EstoreDataSource	WebTier	ps/PetStoreDatasource

Type	Time	Event Message	Source
	3/27/03 2:49 PM	Console Ready.	

Figure 3-8 Resource reference for a DataSource on WebSphere Application Server Advanced Edition

The difference here is that when the Web component is moved to a WebSphere Application Server Advanced Edition server, a similar DataSource needs to be defined for the resource reference to be resolved to.

3.3.8 Systems management

DB2 Connect Enterprise Edition provides a database system monitor to monitor database activity and performance. It may be used for:

- Activity monitoring, to observe the status of database connections and table access activity. It is also possible to track the progress of a query or application using information.
- Problem determination, to collect data for diagnosis of problems caused by application or system performance.
- Performance monitoring, to analyze the performance of individual applications or SQL queries.
- System configuration, to evaluate and tune the effectiveness of the database infrastructure.

For further information, refer to the *System Monitor Guide and Reference* in the DB2 online information.

3.3.9 Strategic considerations

It is possible that in the future, the DB2 Connect functionality will be replaced by that of a Type 4 JDBC driver. Type 4 drivers are Java-only drivers. The change will not necessitate any application program changes. The driver will need to be configured into the application server DataSource.

3.4 JCA access to CICS

It has been said that with Java, you can “write once, run anywhere”. It has also been said that with COBOL, you “write once, run forever”. The world’s IT installations are full of existing applications and data which need to be integrated into the latest application developments and initiatives. Such resources are referred to as *enterprise information systems* (EIS). They are existing corporate resources which the J2EE application may need to access. They may comprise databases, Enterprise Resource Planning (ERP) systems, or transaction processing systems. Rewriting such business logic to implement its function in a J2EE application would usually be too expensive and time-consuming an exercise to undertake. We need to have access to these EIS resources with minimum disruption to them.

In many cases, these resources will already be located on z/OS. Where this is the case, they will be accessed using local call technologies. Local calls are generally efficient, secure, and maintain transactional integrity. If a part of the application that accesses EIS resources is moved away from the z/OS environment, then some of these qualities of service may be compromised.

The J2EE Connector Architecture (JCA) defines a standard API for accessing a variety of EIS resources. The technology that provides the physical connection to the resource is usually provided by the resource vendor. The characteristics of the connection are therefore determined by the capabilities which the vendor builds into the connector product. Not all of the qualities of service are mandated by the JCA specifications; for example, it is not a requirement for a JCA connector to support global transactions (two-phase commit).

The number and range of EIS connectors will continue to grow, and the JCA specification will evolve. The connection described in this book should be considered to be a point-in-time assessment. When assessing the viability of other EIS connectors, they should be subject to a similar review of the various aspects of connection characteristics, as listed in 2.3, "Evaluation criteria for remote component and EIS access" on page 58.

The first consideration for accessing an EIS resource is to determine whether it is possible at all from a remote location. Does any form of technology or connector exist to provide the link from a J2EE application running in a remote container to the EIS resource?

An example of such a resource might be a shared log to which audit records are recorded. It is a conceivable requirement for both Web components and EJB components to record audit data to the same shared log. If the Web components are moved away, then the ability to retain the shared log poses a problem.

If no means of connection to the resource exists, then the options are:

- To develop some form of connector

  This could be a costly and difficult task, particularly if qualities of service such as transactional integrity and security need to be maintained. However, it may be an option for simple applications.

- To relocate or replicate the resource

  If the only components to access the resource are in the remote application server, an option might be to physically move or copy the resource so that it is co-located with those components. Copying resources that are subject to real time updates could pose a problem if both copies need to be kept synchronized. There may also be additional systems management procedures that need to be introduced; for instance, using the example of a shared log, the log files recorded in the separate environments would need to be merged once they had been closed.

In general, though, there will be a robust and reliable means of access from the resource vendor.

3.4.1 CICS Transaction Gateway

The connection we investigated was that required to access CICS. Java client connectivity to CICS servers has been evolving for a number of years now. It originally provided a set of base APIs, which evolved into the Common Connector Framework (CCF). The CCF provided a common approach to coding connections to multiple EIS resources such as CICS and IMS. The CCF principles were then adopted to form a large part of the JCA. This brings us to today, when the JCA is part of the J2EE 1.3 specification, but many applications coded to the CCF conventions still exist.

JCA is, however, the strategic direction. It is the means by which the application server is able to provide us with a managed connection to take care of connection pooling, transaction integrity, and security. For this reason, this chapter only considers the JCA connector, and does not discuss CCF application implications.

In order to submit JCA requests to CICS, two options are available.

1. Make use of a CICS JCA connector. The CICS connector we consider is the CICS Transaction Gateway (CICS TG). At the time of writing, the latest version is IBM CICS Transaction Gateway V5.0. This connector may be installed on Windows, AIX, Linux, Solaris, HP-UX, and z/OS itself. It conforms to the Java 2 Enterprise Edition (J2EE) Connector Architecture, and provides a means for J2EE applications to issue requests to run CICS programs.
2. IBM CICS Transaction Server for z/OS V2 (CICS TS V2) is an EJB server capable of running EJB 1.1 enterprise beans. As such, existing COBOL/CICS programs may be "wrappered" by a session enterprise bean, which may be invoked using normal EJB client techniques.

The installation and migration of a CICS server to CICS TS V2 is a significant undertaking compared with the installation and configuration of a CICS TG. We shall therefore focus on the CICS TG as our mechanism for integrating a J2EE application with heritage CICS applications.

For more information on the CICS TG, refer to:

- *Java Connectors for CICS Featuring the J2EE Connector Architecture,* SG24-6401.
- *CICS Transaction Gateway V5 The WebSphere Connector for CICS,* SG24-6133.
- `http://www-3.ibm.com/software/ts/cics/library/cicstsforzos22.htmlh`

3.4.2 Performance

The CICS TG for z/OS communicates to CICS servers running on z/OS using the External Call Interface (EXCI), a protocol available to non-CICS clients to issue calls to CICS programs within an MVS image. EXCI takes advantage of a platform's Resource Recovery Services (RRS) to implement a two-phase commit capability between the client application (in this case the CICS TG) and the CICS servers. The use of RRS imposes a restriction that the client and the server applications must both be running on the same operating system image. In other words, the WebSphere Application Server for z/OS and OS/390, the CICS TG, and the CICS server must all be in the same image. There is no capability to route work to CICS regions located in other images in the sysplex while keeping the transactional context.

The typical solution to this problem is to use the local CICS region as a gateway. Once it receives a request, it can dynamically route the request to any other CICS region to which it can communicate. CICS is capable of taking z/OS workload manager advice into account when making the routing decision. This solves the workload problem, but leaves the local

CICS region as a single point of failure. This problem is addressed in 3.4.3, "Availability" on page 93.

Another common solution for cross-system CICS communication would be CICS listener based. Using EXCI over XCF as a sysplex-enabled protocol, it allows WebSphere Application Server to communicate to any CICS Transaction Server living within one Parallel Sysplex. Therefore, one needs to implement an CICS listener (CICS TS having remote definitions for transactions and programs) that connects to the target CICS transaction servers using the LU 6.2 (TCP62 or APPC) protocol. No transactional context would be supported in this setup.

The requests between a client and a CICS server flow across EXCI pipes. CICS interregion communication limits the number of pipes on an EXCI connection to 100. If more pipes are required, then the only solution is to replicate the client address space, which in this case is the application server.

The distributed version of CICS TG contains a workload manager function. (This is not to be confused with the workload manager that is used to schedule resources allocated to work on the z/OS platform.) The workload balancing algorithms available are:

- Round-robin - All the CICS regions available are dispatched work in turn.
- Biasing algorithm - "Weights" are assigned to regions such that some are more favored than others when requests are dispatched.

The regions available for selection to the CICS TG workload manager are not constrained to those within a single MVS image. They may be dispersed throughout the sysplex.

3.4.3 Availability

Client applications have a number of options for connecting to the CICS TG. The CICS TG on z/OS may be run as a separate daemon process that is configured to receive requests over TCP/IP, SSL, HTTP, or HTTPS connections. However, none of these are supported when the client application is WebSphere Application Server V4.01 for z/OS and OS/390.

Here we use a special network configuration of "local:", which means that the CICS TG code is run within the application server; there is no requirement for a separate daemon process to be started. In order to provide failover and scalability, from the perspective of the CICS TG all we have to do is run multiple application servers. By doing so, we implicitly run multiple copies of the CICS TG itself. The architecture of WebSphere Application Server for z/OS and OS/390 inherently provides this for us.

The CICS TG for z/OS, running within the WebSphere Application Server for z/OS and OS/390, needs its partner CICS server running in the same MVS image. This presents a single point of failure. CICS does provide an exit for EXCI clients called DFHXCURM. This exit may be coded to distribute the connections amongst two (or more) target CICS servers. It is only invoked when the connection is first established; once created, the connection to the CICS server that was selected persists. The exit is able to detect when a request is unable to reach its CICS server, and take action to mark that server as unreachable so that subsequent requests bypass the failed server.

However, some algorithm needs to be coded into the exit such that it resumes sending work to the server once it has been restarted. One approach would be to resume sending requests to the server after a fixed period of time has elapsed to allow for server recovery. This addresses the issue of the CICS region being a single point of failure.

Attention: Note that this is not a solution for workload balancing, because the connections between the CICS TG and the CICS server (the EXCI pipes) are persistent; once established, they remain until the EXCI connection is closed. Following a CICS server failure, all the pipes will be established to the surviving CICS. After the recovery time has expired, the DFHXCURM exit is only called during Allocate_Pipe processing when a new pipe is being created, and there is no non-disruptive way to redistribute the pipes across two CICS servers.

As with WebSphere Application Server for z/OS and OS/390, WebSphere AE accesses the CICS TG in “local:” mode. So the CICS TG inherits its availability characteristics from the application server. Like the z/OS solution, we need to provide failover by running multiple versions of the application server.

The CICS TG for distributed platforms workload manager is capable of distributing to a number of target CICS servers according to a biasing algorithm. Each CICS server has a weight assigned to it, which is used to bias the amount of work sent to it. When a request fails to reach the target CICS server, the server is removed from consideration by the algorithm. It is reinstated after a period of time determined by the Region timeout(s) parameter, which may be set using the CICS TG configuration tool.

3.4.4 Security

The security characteristics of CICS TG will change as a result of moving the application from WebSphere Application Server for z/OS and OS/390 to another application server. In particular, the means of determining the user context under which the CICS transaction is executed will change.

A property of the connection between the CICS TG and CICS is Attachsec, which is defined on the CICS Connection® resource definition. For a connection to CICS TG for z/OS, the options for this parameter value are Local and Identify. Neither of these settings requires a password to be flowed with the request. CICS makes the assumption that the user ID flowed on the request has already been authenticated.

When a CICS TG running on another platform submits requests, the only valid value for Attachsec on the CICS Connection is *Verify*, which means that both a user ID and password must be flowed on the request. No assumption about the authenticity of the user ID is made, so CICS needs both values to check its validity.

The Application Assembler determines whether the application or the runtime container has responsibility for supplying the authentication credentials to the EIS system. This information is recorded in the <res-auth> element for the resource reference in the deployment descriptor. The value for the <res-auth> element may be set to “Application” if the application code is to provide authorization credentials, or “Container” if the container is to provide authorization credentials.

Note: Depending on the level of specification and tool support, when specifying <res-auth> for servlets, the options may be “Servlet” for application authentication and “Container” for container authentication. “Servlet” is equivalent to “Application”.

One possible scenario is a J2EE application running in WebSphere Application Server for z/OS and OS/390 that submits requests to CICS using a resource reference to identify the DataSource that defines the connection to CICS. The resource reference could specify a <res-auth> of Container, with the Attachsec parameter on the CICS Connection definition set to Identify. If requests to CICS are being submitted from an enterprise bean, then the *RunAs* setting for the bean could be set to *Caller*. This configuration will result in the CICS transaction running under the user identify of the user who was authenticated in the application server.

If this application is migrated away from the WebSphere Application Server for z/OS and OS/390, then the requirement to run the CICS transaction under the user ID of the application user becomes more difficult to implement. The user ID is no longer flowed automatically on the request, but must be provided with a valid password. The only way that this can be accomplished is by setting <res-auth> to Application and programmatically providing the user ID and password. This will require changes to the program code, and will introduce extra security concerns to protect the user ID and password values that need to be flowed on the requests to CICS.

3.4.5 Transaction integrity

J2EE applications running in WebSphere Application Server for z/OS and OS/390 are capable of issuing calls through a local CICS TG to a CICS server with full transactional integrity. This is because the connection to CICS exploits Resource Recovery Services (RRS), a feature exclusive to the z/OS platform. RRS uses a two-phase commit mechanism to ensure that updates applied to multiple resource managers within the same transaction are either all committed together or all backed out together. In our case, updates submitted to CICS through the CICS TG will be coordinated with other resources updated by the J2EE application running in WebSphere Application Server for z/OS and OS/390.

RRS is not available to a CICS TG running on any other platform than z/OS, so calls issued from a WebSphere Application Server Advanced Edition J2EE application cannot exploit its function. Neither is there a two-phase commit protocol used across the connection to CICS. Figure 3-9 shows the commit capabilities for requests to CICS from distributed and mainframe versions of the CICS TG.

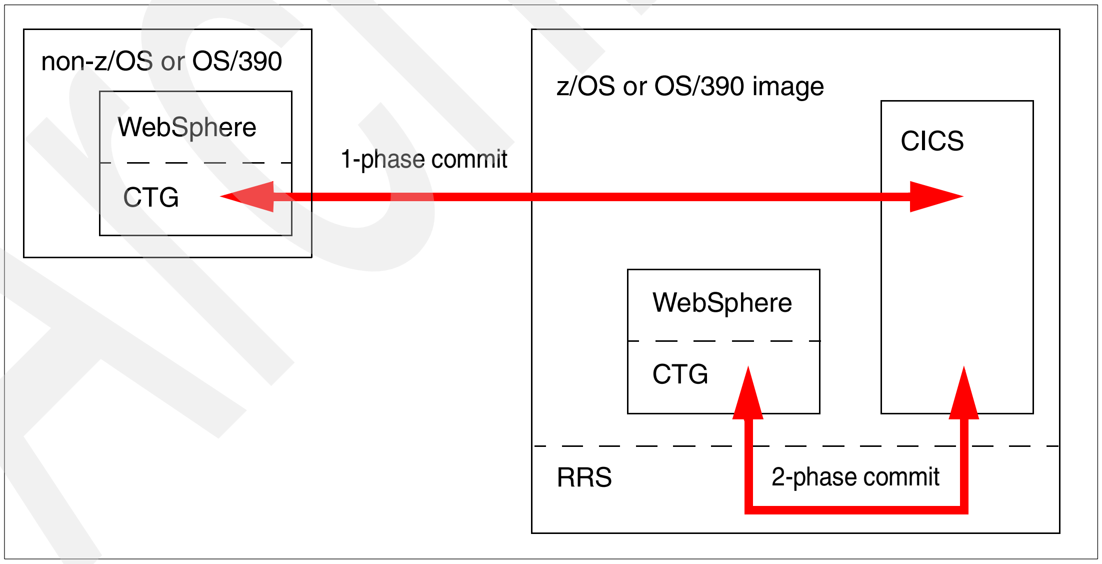


Figure 3-9 CICS Transaction Gateway commit capabilities

Note: WebSphere Application Server Enterprise V5 provides Last Participant Support. With this you can coordinate the use of a single one-phase commit capable resource with any number of two-phase capable resources in the same global transaction. IBM CICS Transaction Gateway V5.0 is the resource that can take advantage of this function.

What this means is that a J2EE application hosted entirely in WebSphere Application Server for z/OS and OS/390 will have full integrity between update requests submitted to CICS and updates to other resource managers (providing that they also are two-phase commit capable). If the part of the application that issues the calls to CICS is moved to another platform, then we lose this guarantee of data consistency.

3.4.6 Infrastructure

A network connection will need to be established between the non-z/OS platform and the CICS. Prior to CICS TS V2, the options available for the connection were SNA or TCP62 (an SNA connection configured over a TCP/IP link). With CICS TS V2, the connection may be native TCP/IP.

If TCP62 is selected as the link technology, then the firewall needs to permit UDP packets on the TCP62 port (397).

3.4.7 Development and deployment

The recommended way of accessing EIS systems is to wrapper the calls in a stateless session enterprise bean, rather than by issuing the call directly from a servlet. If the application has been designed according to the MVC programming model, then the calls will have been coded into the “model” part of the application, which will have been implemented in the EJB container. The user authentication and unit of work issues that are raised when migrating the CICS TG calls to another platform provide compelling arguments for retaining the EJB container in WebSphere Application Server for z/OS and OS/390.

- Requests to CICS may be issued with full transaction integrity.
- The mechanisms for setting the user ID under which the CICS transaction will run are more robust and secure.

However, there are many applications that issue EIS calls directly from Web components such as servlets, so this section addresses issues with those components.

Application architecture

EIS resources would normally form part of the “model” of a model-view-controller design pattern. They represent business data and processes. A possible solution would be to code the EIS calls directly into the servlet, but this is starting to introduce business logic into the “controller” piece of the application. A best practice is to encapsulate any calls to CICS programs in a stateless session bean, rather than coding the call directly into a servlet. This improves the prospects for reuse of the CICS interaction because session beans may be invoked from a wider variety of client applications than servlets, which may only be driven by a client connected over HTTP. Enterprise beans also have more flexibility in the way that their runtime characteristics may be specified at application assembly and deployment time.

Note: One such benefit is the attribute to determine which identity a bean method runs under, which is the RunAs attribute. The servlet 2.3 specification permits RunAs support for servlets that is similar to that for enterprise beans.

If the CICS calls are encapsulated in a stateless session bean, then the client for the connector will reside in the EJB container rather than the Web container. By following the model-view-controller design pattern, there should be no issue with regard to establishing a CICS connection from a client servlet running in the remote Web container.

Tools

WebSphere Studio Application Developer Integration Edition V4.1.1 contains tools for building and testing J2EE applications that submit JCA requests to CICS. It provides wizards that enable the developer to take a COBOL copybook of the CICS COMMAREA storage layouts and generate all the code and supplementary files required to create a CICS service. The CICS service may then be deployed as a session enterprise bean, which will of course live in the EJB container. When these development wizards have been used as designed, there should be no issue with trying to establish a connection between a remote Web component and a CICS server, because the connections will all originate in the EJB container.

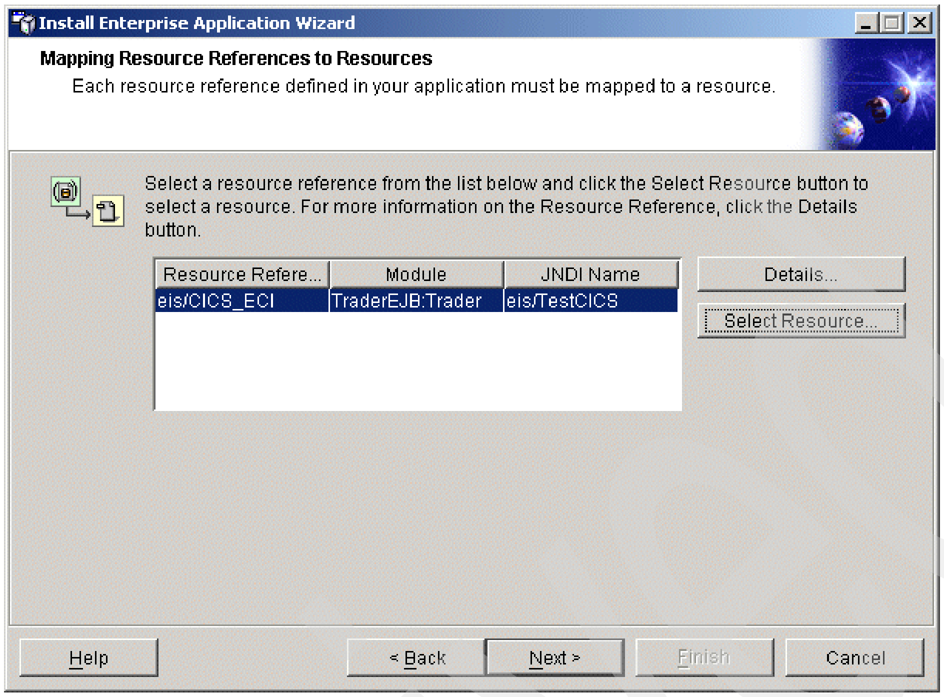
Note: WebSphere Application Server for z/OS and OS/390 requires APAR PQ65206 PTF UQ90051 to provide runtime support for CICS and IMS J2EE connector applications developed using WebSphere Studio Application Developer Integration Edition V4.1.1.

JCA connections

J2EE application components such as servlets and enterprise beans need a connection factory object in order to be able to obtain a connection to an EIS. The J2EE Connector Architecture Specification V1.0 suggests that applications should use JNDI lookups to access their connection factory instance.

The applications use a resource reference to identify the particular connection factory they wish to access. The resource reference is stored as part of the deployment descriptor. When the application is installed into the application server, the resource reference is resolved by the deployer to a real JNDI name, which is able to make the appropriate connection factory instance available.

This level of indirection means that there are no coding changes required to move the application through its various test systems. At each stage of testing, the resource reference is resolved to the appropriate back-end resource. Figure 3-10 on page 98 shows the resource reference “eis/CICS_ECI” being bound during application deployment to WebSphere Application Server Advanced Edition V4.


Install Enterprise Application Wizard

Mapping Resource References to Resources

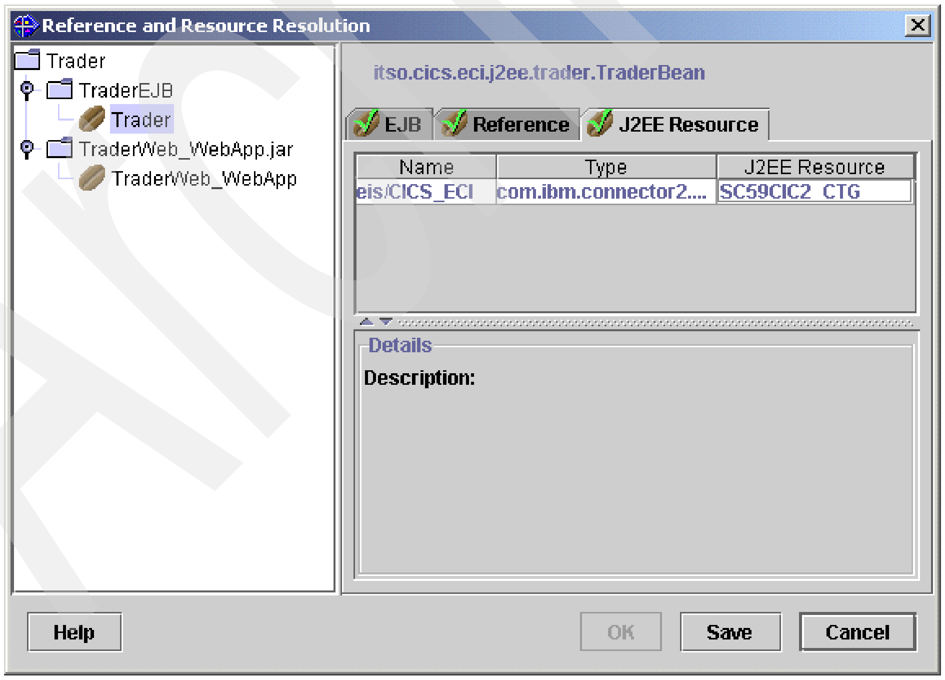
Each resource reference defined in your application must be mapped to a resource.

Select a resource reference from the list below and click the Select Resource button to select a resource. For more information on the Resource Reference, click the Details button.

Resource Refere...	Module	JNDI Name
eis/CICS_ECI	TraderEJB:Trader	eis/TestCICS

Figure 3-10 Resolving a resource reference on WebSphere Application Server Advanced Edition

Figure 3-11 shows the same resource reference in the same application being bound during deployment to WebSphere Application Server V4.01 for z/OS and OS/390.


Reference and Resource Resolution

itso.cics.eci.j2ee.trader.TraderBean

Name	Type	J2EE Resource
eis/CICS_ECI	com.ibm.connector2....	SC59CIC2_CTG

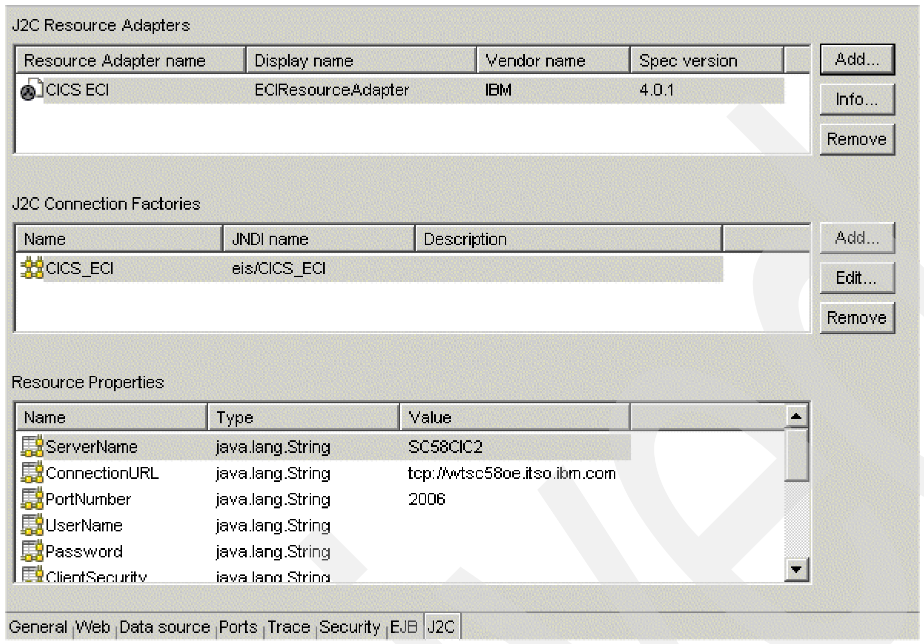
Details

Description:

Figure 3-11 Resolving a resource reference on WebSphere Application Server for z/OS and OS/390

This capability to bind a resource reference to a real resource at deployment time is extended to WebSphere Studio Application Developer Integration Edition V4.1.1, so unit testing of the application in the development tool is possible.

Figure 3-12 shows a resource reference being bound in the test environment of WebSphere Studio Application Developer Integration Edition V4.1.1.



J2C Resource Adapters

Resource Adapter name	Display name	Vendor name	Spec version
CICS ECI	ECIResourceAdapter	IBM	4.0.1

J2C Connection Factories

Name	JNDI name	Description
CICS_ECI	eis/CICS_ECI	

Resource Properties

Name	Type	Value
ServerName	java.lang.String	SC58CIC2
ConnectionURL	java.lang.String	tcp://wtsc58oe.itso.ibm.com
PortNumber	java.lang.String	2006
UserName	java.lang.String	
Password	java.lang.String	
ClientSecurity	java.lang.String	

Figure 3-12 Resolving a resource reference in WebSphere Studio Application Developer Integration Edition V4.1.1

3.4.8 Systems management

The CICS TG element of the infrastructure is required whether the application is deployed to WebSphere Application Server for z/OS and OS/390 as a single integrated application or to several application servers as a hybrid solution. There is therefore not much difference in the systems management overhead between the two configurations that has not already been incurred by the addition of the remote WebSphere AE server. From the perspective of the CICS TG, many of the systems management tasks are common to both scenarios.

3.4.9 Strategic considerations

The CICS TG APIs used by application program clients to CICS have been evolving. Recent versions of CICS TG provide support for the J2EE Connector Architecture (JCA). Programs should be developed according to this API so that they can take advantage of the managed connections which modern application servers that support the JCA provide. Utilizing managed connections enables qualities of service such as transaction integrity, connection pooling, and security to be taken care of by the application server so that the developer does not have to worry about them.

In addition, the JCA APIs provide a common programming model for accessing any backend resource, so only one set of skills needs to be obtained. The JCA is a strategic initiative for the industry, so modern development tools and their built-in wizards generate code to this specification.

4

Static Web component optimization

This chapter describes how static and selected dynamic content can be served outside the WebSphere Application Server for z/OS and OS/390. It discusses various possible scenarios and describe how to implement them.

4.1 Overview

Almost all Web applications have mixed static and dynamic content. Before the use of dynamic content in applications, when the Web was made mostly of static content consisting of HTML, graphical images in binary formats, and other non-dynamic elements embedded into the HTML structure. Web servers were in charge to serve this static content.

Today, as the application servers take more and more work to manage business logic, it seems a good idea to continue using the Web servers for the work they do best: static page serving, file caching, and proxying requests.

When serving this content plain out of an WebSphere Application Server it will always be delivered by the Web container. The Web container will always use a Java program (SimpleFileServlet) to access the content. The Web components will not be cached by default. If you have an application that is rich in static content, you would finally end up with a lot of servlet activity within your Web container. To avoid this overhead of static content processing in the WebSphere Application Server, you need to architect your infrastructure in such a way that other specialized products and functions can give the Web container a helping hand.

Important: If your main motivation for a hybrid infrastructure is to relieve the J2EE server on z/OS from some Web container work, static file offloading and dynacaching could be a solution for you. It is easy to implement, does not confront you with the complexity of a splitted J2EE application, and in most cases gives a significant relief for your WebSphere Application Server. Nevertheless, you can combine the offloading techniques described here with hybrid deployment approaches. Therefore, we will discuss static content offloading solutions for hybrid and integrated infrastructures.

Application component interaction

Components discussed here are either totally static HTML-based or JSP-generated semi-dynamic content.

HTML pages usually consist of HTTP code and embedded (or referenced) binary objects (gif, jpg, wav, mp3 files). These pages are often dynamically composed by a JSP function within the Web application running in the Web container. Dynamically generated HTML files also have the possibility of adding (referencing) any objects to their structure. That means that the ensemble, wherever it is, has to maintain its logical structure.

If the binary objects are not created dynamically by the application, they are usually stored inside the .war file of the application's .ear file as components of the application and are managed, as Web content, by the Web container.

Web components interact using the HTTP protocol and HREF statements. HTML forms can also be used to trigger dynamic content creation of information from the client/customer. The information must be sent to the application server to be processed, stored or manipulated by the Web container. Forms use the HTTP GET or POST method.

Architectural building blocks and configurations

A Web server as a front end for WebSphere Application Server is most likely used for the following functionality:

- Workload distribution, using specialized plug-ins
- Security front-end to interact with registries and the ability to forward private security headers to the WebSphere Application Server

- HTTP session handling by maintaining the session state when distributing content (session affinity)
- Serving static content, caching static content
- Caching dynamic content (dynacaching)
- Proxy or reverse proxy configuration

To architect some valid Web content offloading configurations that support integrated and hybrid infrastructures, we decided to work only with the most common infrastructure elements.

This chapter describes four different configurations, though there are many more ways to architect an appropriate solution.

We kept our application server on the z/OS environment with dynacaching activated (see 4.2, “Dynamic fragment caching concepts” on page 105) and we used the IBM HTTP Server in two platforms, z/OS and distributed.

For the architectures that use distributed components one can use almost any platform of choice. To avoid additional boxes (and the problems related to this) you can run your Web server on a Linux for zSeries image. To follow our configuration guidelines more easily, we document the configuration steps based on the settings in a Windows operating system:

- 4.3, “Configuration 1: Local IBM HTTP Server for static file handling” on page 109
- 4.4, “Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in” on page 115
- 4.5, “Configuration 3: Remote reverse proxy caching server” on page 122
- 4.6, “Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in” on page 126.

The generic scenarios are depicted in Figure 4-1 on page 104 and Figure 4-2 on page 105.

Interesting variations develop according to the placement of the Web server—that is, on z/OS or on a distributed platform.

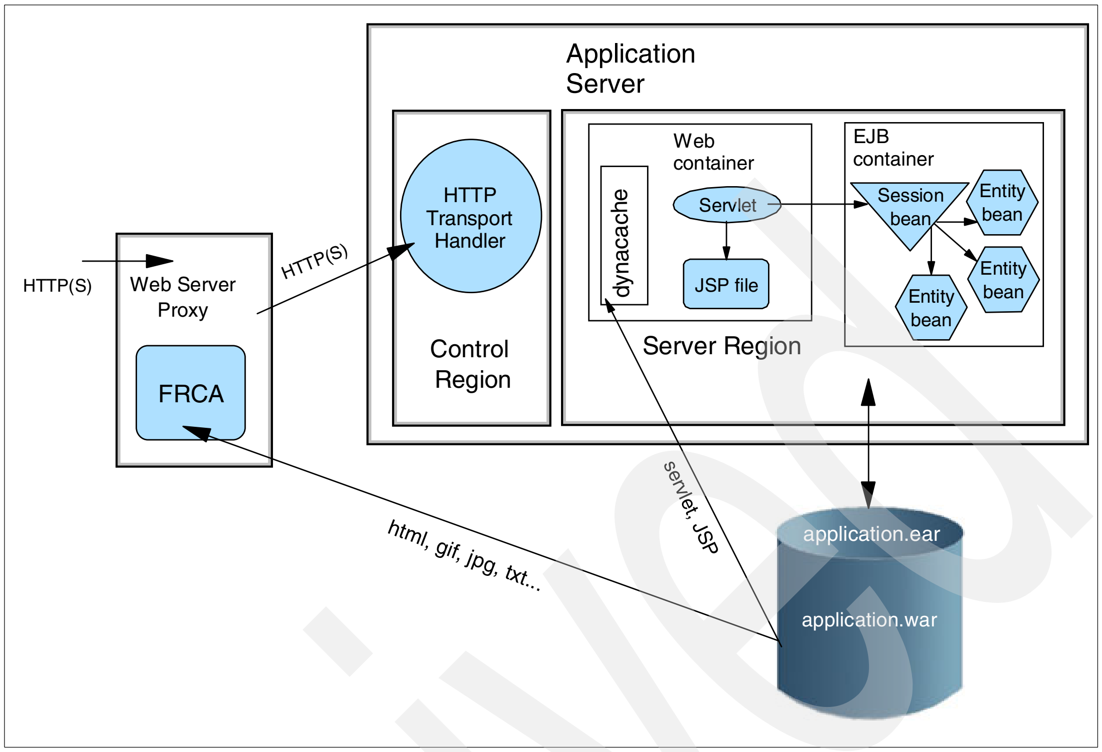


Figure 4-1 Generic caching scenario (configurations 1 and 3)

Deciding which of these configurations you should use depends on the overall infrastructure architecture you are going to plug into. If, for example, the decision in your enterprise has been made that all Web Servers have to be run on Linux, then logically you do not need to think about the z/OS Web server scenario. Other major influencers on the scenario selection are security, availability, and HTTP session aspects.

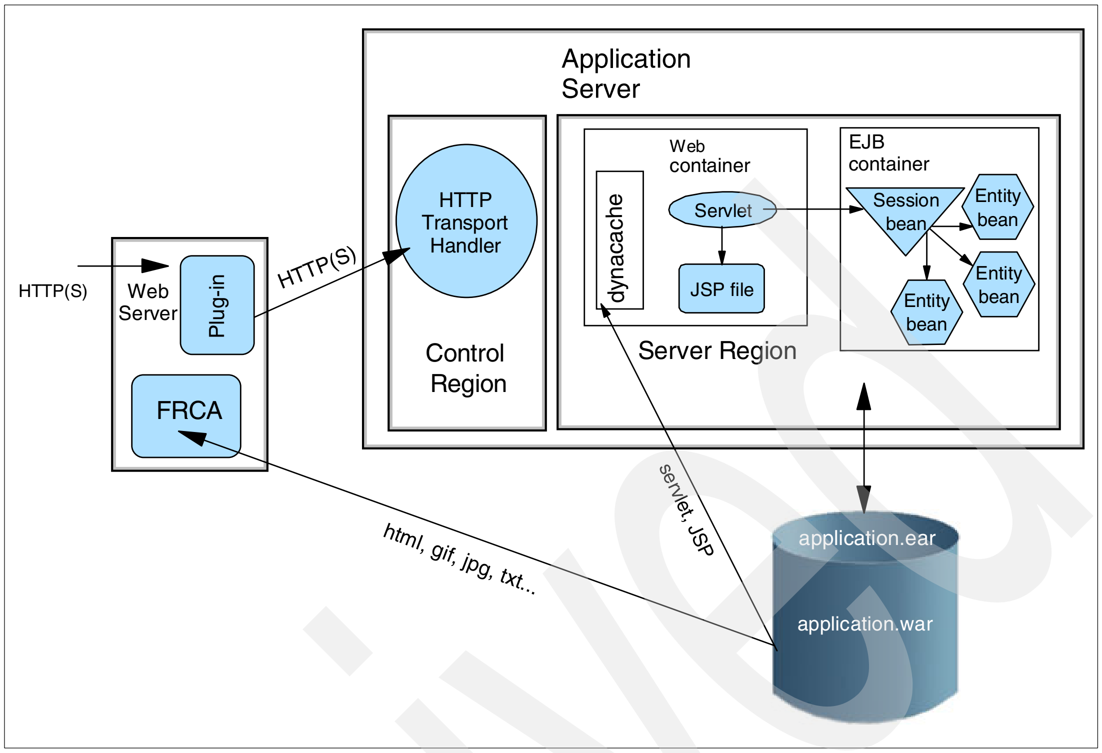


Figure 4-2 Generic WebSphere plug-in scenario (configurations 2 and 4)

4.2 Dynamic fragment caching concepts

The idea of dynamic fragment caching (also servlet caching or dynacaching) is to store the output of servlets and JSP procession within the server region's JVM. For all subsequent calls of the same servlet or JSP, the server region checks whether the response can be delivered from the JVM's cache instead of running the servlet or JSP again to recreate the result.

Depending on the type of application running in the WebSphere Application Server, dynamic fragment caching can improve the performance and response time of the application significantly. It also caches some other servlet products such as:

- Content types
- Character encodings
- Setting cookies
- Including and forwarding to other servlets

Dynacache gives better performance when the J2EE application has a suitable design for caching. A good way for dynamic caching would be to have page designs that are built from several "subpages". These fragments can be reused along the application presentation layer.

Important: Dynamic fragment caches live in each server region, and no attempt to synchronize caches among server regions is made. This means that each server region has its own dynacache and the same content can use up JVM heap sizes in different server regions, as shown in Figure 4-3. It makes sense to direct dynamic fragment cached requests always to the same server region to save some memory within the JVM heap.

The technique to use is an HTTP session for the JSP or servlet and session affinity support in a front-end workload distributor or Web server running the WebSphere HTTP plug-in. The control region distributing the requests to the individual server regions supports session affinity by default. Note that JSPs automatically enable HTTP sessions, so you should not manually turn this feature off. If you are not exploiting session data, you can improve performance by disabling session data in your JSPs (session=”FALSE").

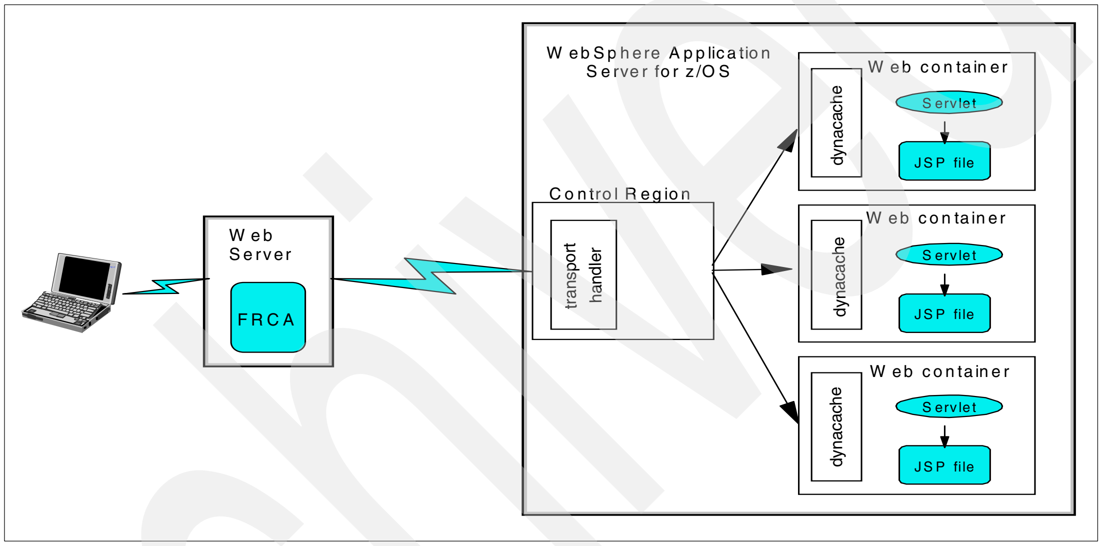


Figure 4-3 Dynamic fragment caching. Different server regions have different dynamic caches

This Dynamic Fragment Caching can be set in combination with an external cache server that can increase the offload (see Figure 4-4).

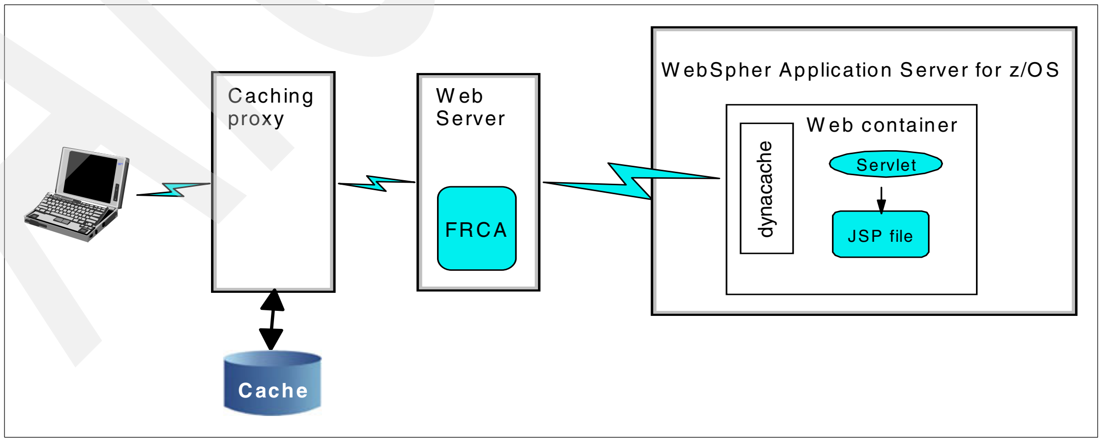


Figure 4-4 Dynamic Fragment caching with external caching proxy

When the external caching proxy is enabled, eligible pages are stored in the external cache and are served from there instead of the application server. A caching proxy has the ability to store and manage cached data on an external device. Be aware that if the caching ability of an Web server itself is used, the cached data would stay in the Web servers address space or in the address space of the supporting TCP/IP stack.

4.2.1 Configuring dynamic fragment cache support

To use the benefits of dynamic caching you have to set up the necessary files: dynacache.xml and servletcache.xml (see “Dynamic fragment caching” on page 141 for setup details).

The presence of the dynacache.xml file is sufficient to enable caching. The file is read at application server startup. If you make changes to any of the files, the J2EE server has to be recycled to activate the changes.

In this file you can define the global properties you want for your cache. You can get a list of the configured properties by looking at the dynacache.dtd shipped with your application server, located in <install-root>/dtd (see *Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836, chapter 8).

It is also in the dynacache.xml file where you can define the external cache groups that will be managed by the external caching proxy. The commented part of Example 4-1 shows what a definition for external caching could be. This example is the sample file shipped with the application server for z/OS.

Example 4-1 The dynacache.sample.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cacheUnit SYSTEM "dynacache.dtd">

<cacheUnit>
  <cache size="1000" priority="1" />



<!--
<externalCacheGroups>
     <group id="afpa" >
        <member address="9081" adapterBeanName="com.ibm.servlet.dynacache.Afpa"/>
     </group>
  </externalCacheGroups>
-->
</cacheUnit>
```

The second file you need to enable Dynamic Fragment Caching is servletcache.xml (see “Dynamic fragment caching” on page 141 for setup details); this file defines which and how servlets and JSPs are to be cached: this is where you define your cache policies.

Note: Dynamic Fragment Cache policies can be defined via the servletcache.xml file or using an XMI file attached to a .war file; to use this option the Web application developer has to create the file and put it in the application. The Application Assembly Tool (AAT) for z/OS and OS/390 will preserve these definitions (390fy will also preserve them). If an entry is found in both, the XML file takes precedence.

You may use the servletcache.xml file to have custom definitions for each of your servlets or JSPs, specifying:

- The class file or the URI of the servlet to be cached
- Unique entries for different requests
- Variables to determine which pieces of information are associated with the unique ID
- Time to keep the entry in the cache
- Priority
- ...

The servletcache.dtd is available from your application server in <install-root>/dtd. To illustrate all this, see Example 4-2; it is the sample file shipped with the WebSphere Application Server.

Example 4-2 The servletcache.sample.xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE servletCache SYSTEM "servletcache.dtd">

<servletCache>
  <servlet>
      <timeout seconds="0" />
      <servletimpl class="SnoopServlet.class" />
  </servlet>

<!--

  <servlet>
      <invalidateonly/>
      <path uri="/inv.jsp" />
      <path uri="/status/inv.jsp" />
      <request>
         <parameter id="invalidate" invalidate="arg1" />
         <parameter id="invalidate" invalidate="arg2" />
      </request>
  </servlet>
  <servlet>
      <timeout seconds="0" />
      <path uri="/status/calc.jsp" />
      <request>
        <parameter id="arg1" data_id="arg1" />
        <parameter id="arg2" data_id="arg2" />
        <parameter id="invalidate" invalidate="arg1" />
        <parameter id="caching" >
           <exclude value="off"/>
        </parameter>
      </request>
  </servlet>
  <servlet>
      <timeout seconds="10" />
      <path uri="/frontpage.jsp" />
      <externalcache id="afpa" />
  </servlet>

-->
</servletCache>
```

To know whether your cache policy is working as desired, WebSphere Application Server provides a dynamic fragment caching monitor, aka servlet cache monitor. It is delivered as an

.ear file, ServletCacheMonitor.ear, that you have to deploy. For more information about this monitor, go to the WebSphere v4.x Infocenter. You can also refer to (specifically for z/OS):

- *WebSphere Application Server for z/OS and OS/390 v4.0.1: Migration,* GA22-7860
- *WebSphere Application Server for z/OS and OS/390 v4.0.1: Installation and Customization,* GA22-7834
- *Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836

You can find a description of the cache API package at:

`ibm.com/software/webservers/appserv/doc/v40/aee/wasa_common/apidocs/index.html`,

4.3 Configuration 1: Local IBM HTTP Server for static file handling

This configuration describes how to use an IBM HTTP Server for z/OS configured as a reverse proxy (see Figure 4-5 on page 110) and as a static content server. All the requests flow through this IBM HTTP Server, so it needs to decide which requests can be served directly, and which ones need to be forwarded to the Web container.

By placing the IBM HTTP Server on the same z/OS image as the Web container, you gain direct access to the static content that is deployed to the local Hierarchical File System (HFS). In addition you can exploit platform-specific static content accelerators, such as Fast Response Cache Accelerator (FRCA) in the TCP/IP stack, or the caching mechanisms that live in the Web server itself. All this can be combined with dynamic fragment caching.

It is supported to have the plug-in forward requests to the IBM HTTP Server on z/OS if you need a WebSphere HTTP plug-in off platform.

The implementation guidelines for this configuration can be found in 5.2.2, “Configuration 1: Local IBM HTTP Server for static file handling” on page 142.

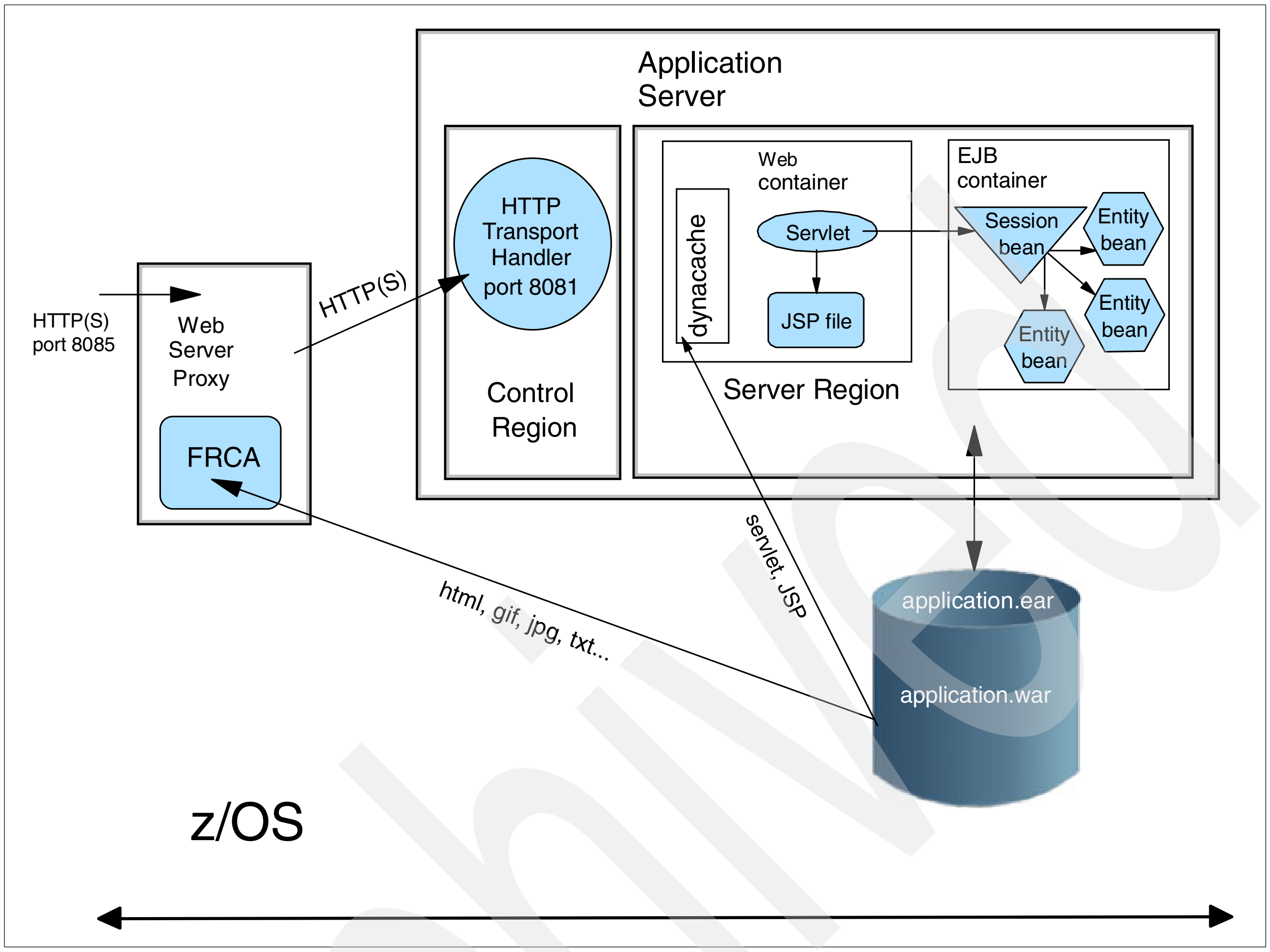


Figure 4-5 Configuration 1. IBM HTTP Server for z/OS as a local proxy

4.3.1 HTTP session considerations

If you have HTTP sessions in memory and multiple server instances, your infrastructure needs to maintain session affinity. This configuration, as it is, does not support session affinity. You should rather look at the WebSphere HTTP plug-in configurations described in 4.4, "Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in" on page 115 and 4.6, "Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in" on page 126.

4.3.2 Security considerations

You need to understand that Web container security is treated differently than Web server security. Web server security is triggered by protection setups, Web container security by deployment descriptors. Basically you have access to almost the same authentication models:

- Basic authentication
- Certificate-based authentication
- Form-based authentication
- Unauthenticated processing

All of these are valid and can be set up in different ways according to the needs of the application. For a detailed discussion about security, see *z/OS WebSphere and J2EE Security Handbook,* SG24-6846. Chapter 14 in that book provides detailed information about the flow and setup of authentication; chapter 15 has a flow chart that will help you make the decisions for your security needs.

The problem now is that you have to configure exactly the same authentication model in the Web server and in the Web container. You also need to make sure that the client is not asked for his credentials several times.

Usually the static content does not contain security-relevant information, it often consists of corporate logos and design elements. Our recommendation is to use the static file offloading capabilities of this configuration when you do not need to protect your static content. For our configuration we did not protect the static content delivered by the IBM HTTP Server.

During our tests we worked with HTTP basic authentication. When you use basic authentication, the authentication information is present in the HTTP authorization header. Basically, the authenticated USERID and PASSWORD are in there. Because IHS and WebSphere Application Server on z/OS share the same security repository, the authentication information generated by the Web server could be reused by the Web container. In this case you need to make sure that the ServerID value in the httpd.conf matches the <realm-name> in the deployment descriptor or web.xml. This technique allows you to have the static and the dynamic application elements secured using the same policy.

4.3.3 System management considerations

This configuration is transparent to the deployment process. The only thing the administrator has to do is to configure the IBM HTTP Server accordingly. The PASS directives in the httpd.conf need to point to the proper path where the application is deployed (see details in 5.2.2, “Configuration 1: Local IBM HTTP Server for static file handling” on page 142).

After setting up this configuration, double-check that the static content is now served by the IBM HTTP Server and not by the application server. You can disable static content management in the Web container as described in “Integrated scenario” on page 138, so you will immediately see an error page if the setup is not working correctly.

You can also check the Web server’s access and FRCA logs.

4.3.4 Performance considerations

Intelligent file caching itself means to deliver the content from fast and expensive memory rather than from slow and cheap physical I/O devices. Intelligent caching also means using the expensive memory resources carefully and applying good algorithms to hold memory consumption low. Generally caching improves the overall response time and allows a higher throughput rate. Be aware that if you exploit caching technologies without having the necessary resources in place, you could even suffer performance penalties.

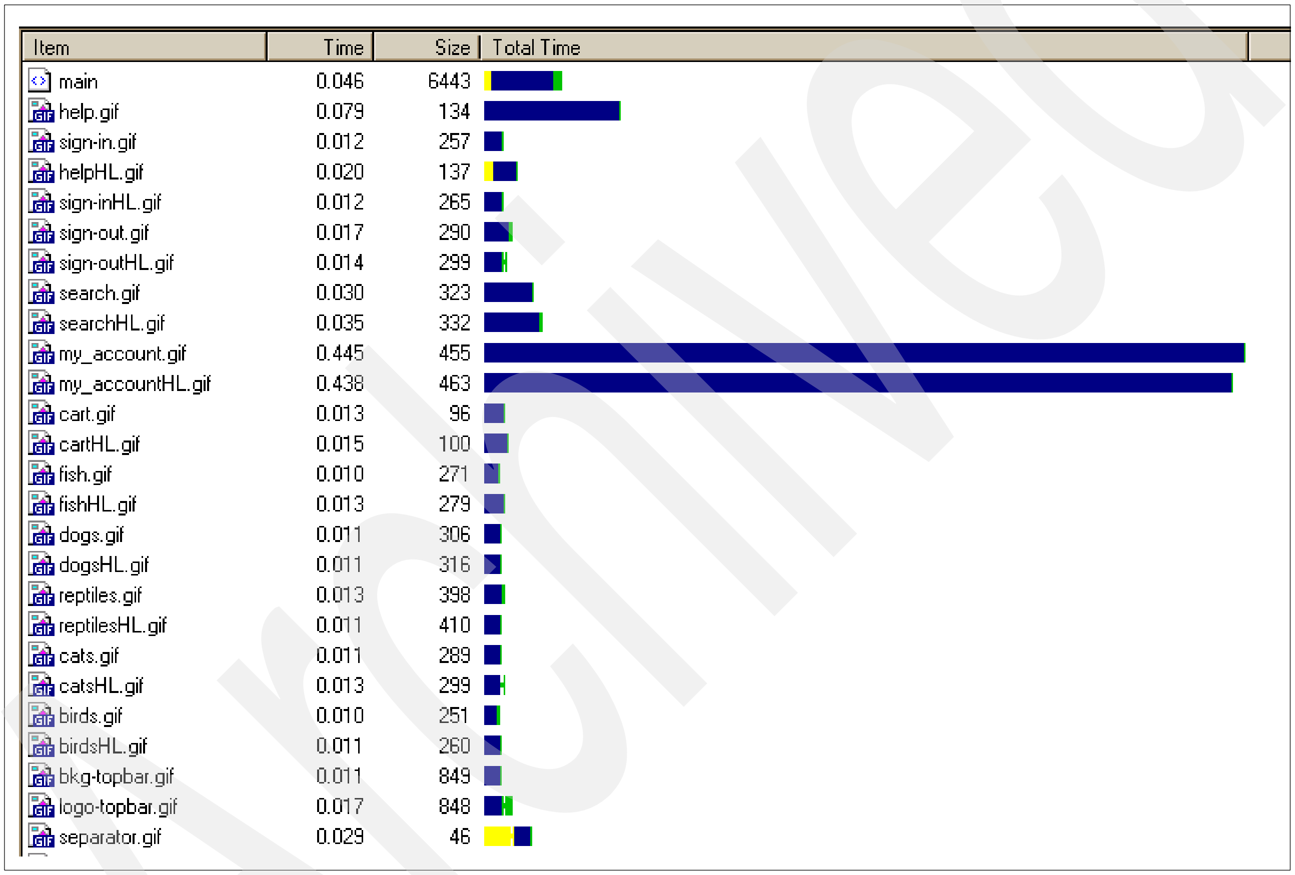
When a static document is cached in a caching hierarchy, less requests are travelling through the network and some network bandwidth can be saved. The rule of thumb is to cache as close as possible to the client. This allows the best bandwidth consumption reduction.

To illustrate this, we ran some tests that consisted of capturing all the information travelling from and to the client. The information included response times, amount and size of headers, and total serving times.

For our tests, dynamic fragment caching was enabled in the application server. We ran two performance measurements, one with and one without additional FRCA caching in the IBM HTTP Server on z/OS. As a test case we measured the welcome page of our sample application.

Reference data

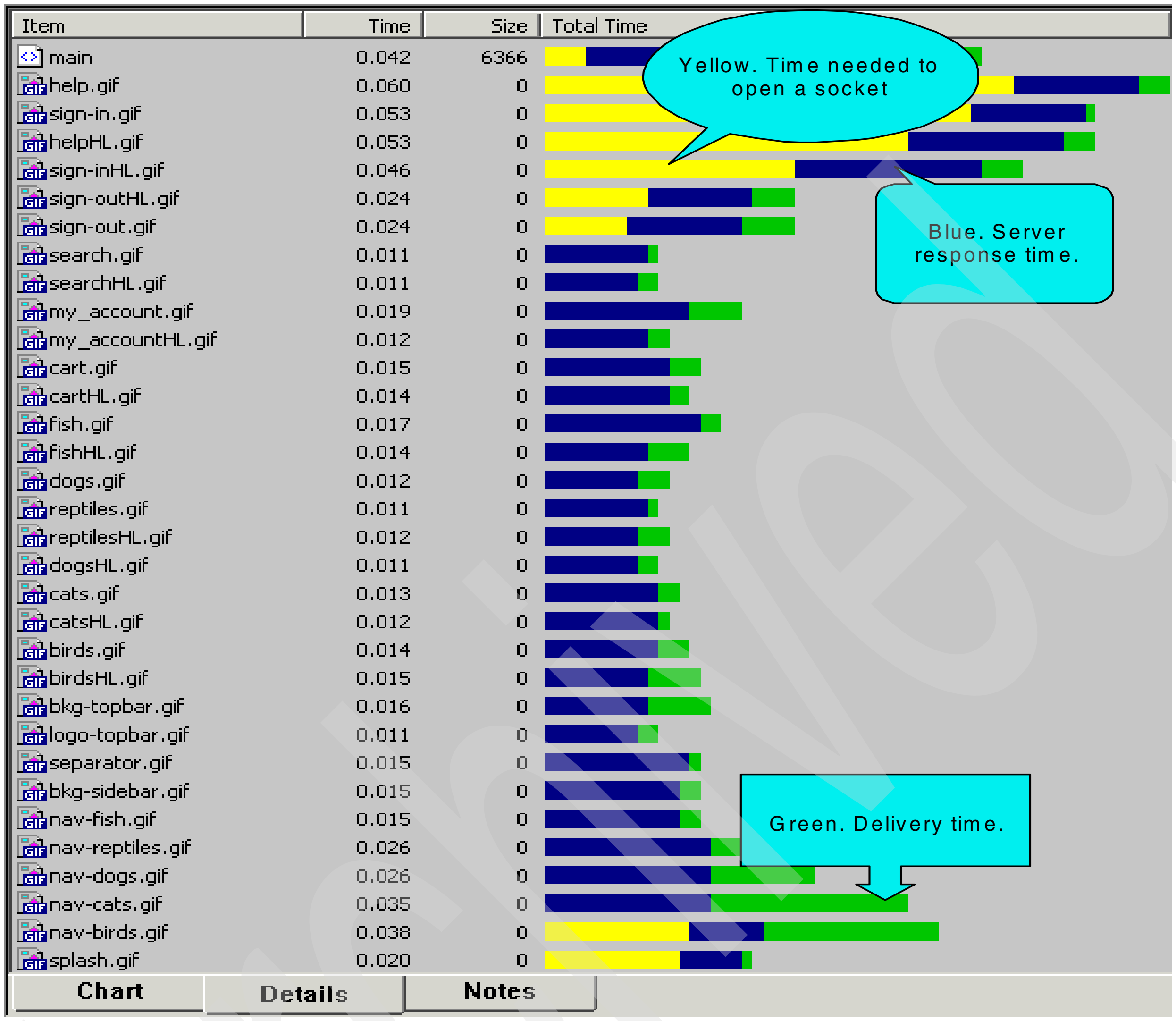
Figure Figure 4-6 shows the measurements without static offloading, serving the content directly through the Transport Handler. When you compare this data with the following measurements, keep in mind that these are no scientific performance measurements or benchmarks. We took these measurements to get a feeling of how our configurations behave. We did not make any measurements regarding the path length reduction on the z/OS side. If your target is to avoid CPU cycles, you need to look at RMF reports.



Item	Time	Size	Total Time
main	0.046	6443	
help.gif	0.079	134	
sign-in.gif	0.012	257	
helpHL.gif	0.020	137	
sign-inHL.gif	0.012	265	
sign-out.gif	0.017	290	
sign-outHL.gif	0.014	299	
search.gif	0.030	323	
searchHL.gif	0.035	332	
my_account.gif	0.445	455	
my_accountHL.gif	0.438	463	
cart.gif	0.013	96	
cartHL.gif	0.015	100	
fish.gif	0.010	271	
fishHL.gif	0.013	279	
dogs.gif	0.011	306	
dogsHL.gif	0.011	316	
reptiles.gif	0.013	398	
reptilesHL.gif	0.011	410	
cats.gif	0.011	289	
catsHL.gif	0.013	299	
birds.gif	0.010	251	
birdsHL.gif	0.011	260	
bkg-topbar.gif	0.011	849	
logo-topbar.gif	0.017	848	
separator.gif	0.029	46	

Figure 4-6 Welcome page processing of our sample application: reference data

For the proxy server with FRCA disabled, we had the following results for the main page of the sample application (see Figure 4-7 on page 113).


Total Time
Yellow. Time needed to open a socket
Blue. Server response time.
Green. Delivery time.
Chart
Details
Notes

Item	Time	Size
main	0.042	6366
help.gif	0.060	0
sign-in.gif	0.053	0
helpHL.gif	0.053	0
sign-inHL.gif	0.046	0
sign-outHL.gif	0.024	0
sign-out.gif	0.024	0
search.gif	0.011	0
searchHL.gif	0.011	0
my_account.gif	0.019	0
my_accountHL.gif	0.012	0
cart.gif	0.015	0
cartHL.gif	0.014	0
fish.gif	0.017	0
fishHL.gif	0.014	0
dogs.gif	0.012	0
reptiles.gif	0.011	0
reptilesHL.gif	0.012	0
dogsHL.gif	0.011	0
cats.gif	0.013	0
catsHL.gif	0.012	0
birds.gif	0.014	0
birdsHL.gif	0.015	0
bkg-topbar.gif	0.016	0
logo-topbar.gif	0.011	0
separator.gif	0.015	0
bkg-sidebar.gif	0.015	0
nav-fish.gif	0.015	0
nav-reptiles.gif	0.026	0
nav-dogs.gif	0.026	0
nav-cats.gif	0.035	0
nav-birds.gif	0.038	0
splash.gif	0.020	0

Figure 4-7 Welcome page processing of our sample application with FRCA off

In this diagram, the bars are scaled and show the time to load all elements of the welcome page. The different colors represent parts of the overall response time spent in different locations. The overall response time spent to load individual components is the total length of the bar adding all colors. The color coding is as follows:

- Yellow describes the time needed to open a socket.
- Blue describes the server response time.
- Green describes delivery time. If some additional data has been sent from the server because it couldn't be sent in the initial reply, it is measured here.

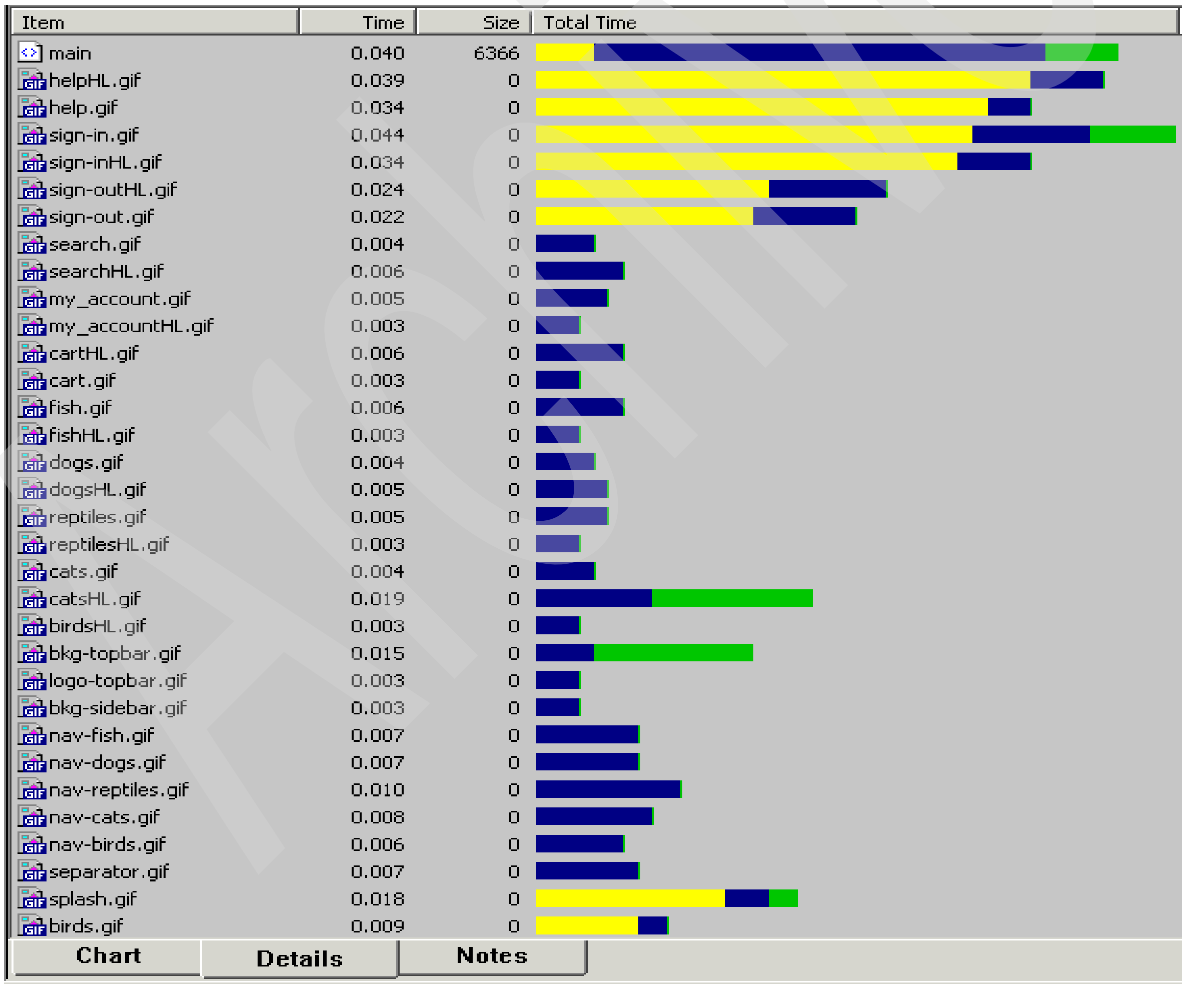
This bar diagram is also translated to numerical data. We put an example for the largest gif image in Example 4-3.

Example 4-3 Times and sizes for logo-topbar.gif (without FRCA)

```
Item: logo-topbar.gif
           Type: image/gif
            URL: http://wtsc59.itso.ibm.com:8085/estore/images/logo-topbar.gif
           Date: 03/28/2003 09:25:56
```

```
    Page Offset: 0.287 seconds
Previous Offset: 0.001 seconds

Response Times:
       0.011 seconds Total Time
       0.009 seconds Server Response Time
       0.002 seconds Delivery Time
       0.001 seconds Delivery Idle Time

Item Sizes:                       Sent   Received      Total
                 HTTP Headers:     458        251        709
             Application Data:       0          0          0
                  Total Bytes:     458        251        709
                Overhead Data:     458        251        709

Communication:
          Socket ID: 1452
     Remote Address: 9.12.6.16 : 8085
```

For the same configuration, but now with FRCA enabled, the results are shown in Figure 4-8.



Item	Time	Size	Total Time
main	0.040	6366	
helpHL.gif	0.039	0	
help.gif	0.034	0	
sign-in.gif	0.044	0	
sign-inHL.gif	0.034	0	
sign-outHL.gif	0.024	0	
sign-out.gif	0.022	0	
search.gif	0.004	0	
searchHL.gif	0.006	0	
my_account.gif	0.005	0	
my_accountHL.gif	0.003	0	
cartHL.gif	0.006	0	
cart.gif	0.003	0	
fish.gif	0.006	0	
fishHL.gif	0.003	0	
dogs.gif	0.004	0	
dogsHL.gif	0.005	0	
reptiles.gif	0.005	0	
reptilesHL.gif	0.003	0	
cats.gif	0.004	0	
catsHL.gif	0.019	0	
birdsHL.gif	0.003	0	
bkg-topbar.gif	0.015	0	
logo-topbar.gif	0.003	0	
bkg-sidebar.gif	0.003	0	
nav-fish.gif	0.007	0	
nav-dogs.gif	0.007	0	
nav-reptiles.gif	0.010	0	
nav-cats.gif	0.008	0	
nav-birds.gif	0.006	0	
separator.gif	0.007	0	
splash.gif	0.018	0	
birds.gif	0.009	0	

Figure 4-8 Welcome page processing of our sample application with FRCA on

The numerical data for logo-topbar.gif is shown in Example 4-4.

Example 4-4 Times and sizes for logo-topbar.gif with FRCA on

```
Item: logo-topbar.gif
           Type: image/gif
            URL: http://wtsc59.itso.ibm.com:8085/estore/images/logo-topbar.gif
           Date: 03/28/2003 09:35:32
    Page Offset: 0.275 seconds
Previous Offset: 0.001 seconds

Response Times:
       0.003 seconds Total Time
       0.003 seconds Server Response Time
       0.011 seconds Local Closed Socket Offset

Item Sizes:                         Sent   Received      Total
                HTTP Headers:        419        189        608
            Application Data:          0          0          0
                 Total Bytes:        419        189        608
               Overhead Data:        419        189        608

Communication:
          Socket ID: 1572
     Remote Address: 9.12.6.16 : 8085
```

These measurements were not done in a strictly controlled scientific benchmarking environment. We followed some guidelines for the measurements, like restarting the servers and clearing all caches. Comparing the results, it is obvious that you can get additional throughput enhancements with FRCA for your static content. Note that the bar diagrams cannot be compared directly because they are scaled to the longest bar in each case, but you can look at the times beside the bars to compare.

4.4 Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in

This configuration exploits the IBM HTTP Server for z/OS running with the new WebSphere HTTP Plug-in for WebSphere Application Server for z/OS and OS/390 (see Figure 4-9 on page 116).

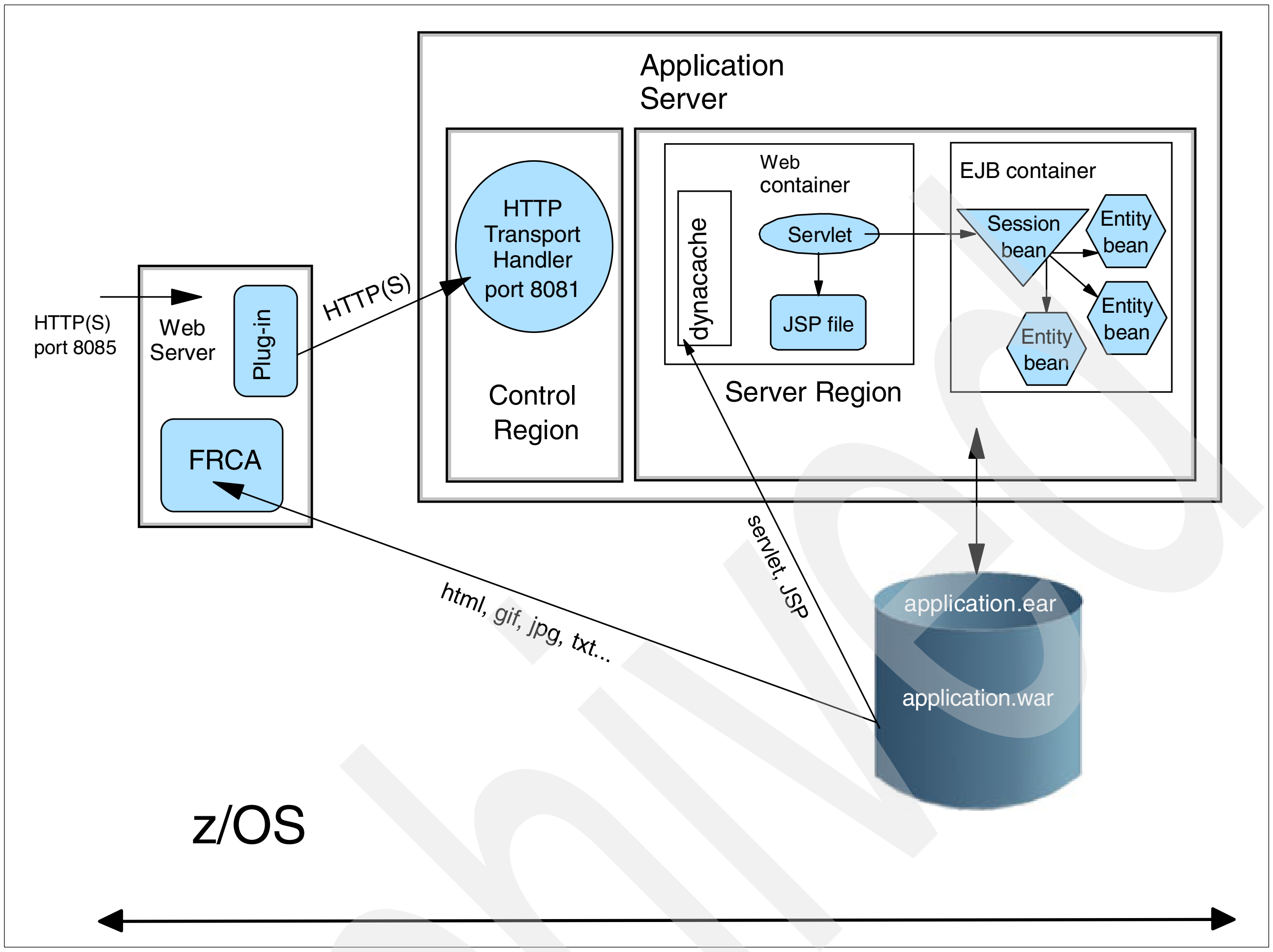


Figure 4-9 IBM HTTP Server for z/OS with WebSphere HTTP Plug-in

The plug-in is shipped with service level W401500, APAR PQ68250. For more info about setting up this plug-in, see 5.2.3, “Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in” on page 144.

With this configuration, the IBM HTTP Server is able to forward requests to J2EE server instances configured in the plug-in’s configuration file. This configuration supports multiple server instances, even multiple server instances on other systems within the scope of one WebSphere node.

With some configuration effort it is possible to get static content served directly from the Web server (see 5.2.3, “Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in” on page 144) with no additional overhead for the application server, while dynamic requests are forwarded to the Web container. It is also possible to enable the Fast Response Cache Accelerator (FRCA) for this configuration.

The difference with 4.3, “Configuration 1: Local IBM HTTP Server for static file handling” on page 109 is that, using the plug-in, we can distribute selected traffic to specific server instances in WebSphere Application Server for z/OS and OS/390 by using different <Server> and <Transport Hostname> statements in the plugin-cfg.xml file. The workload distribution is based (depending on the plug-in version) on weighting and, of course, session affinity. The plug-in adds some additional front-end functionality for the WebSphere Application Server into IBM HTTP Server (certificate handling, request sanitation, and more).

4.4.1 HTTP session considerations

The WebSphere HTTP Plug-in supports session affinity and also allows forwarding to IP sprayers (like Sysplex Distributor) that live between the plug-in and the Web container. This configuration fully supports HTTP session affinity. If you distribute workload from this plug-in to a remote Web container you will lose (with the configuration described here) the ability to serve static content in the IBM HTTP Server. You can solve this problem by sharing the content HFS and configuring the Web server accordingly.

4.4.2 Security considerations

See 4.3.2, “Security considerations” on page 110. Same considerations apply for both configurations.

4.4.3 System management considerations

This configuration does not require special management, other than the usual for WebSphere Application Server for z/OS and OS/390.

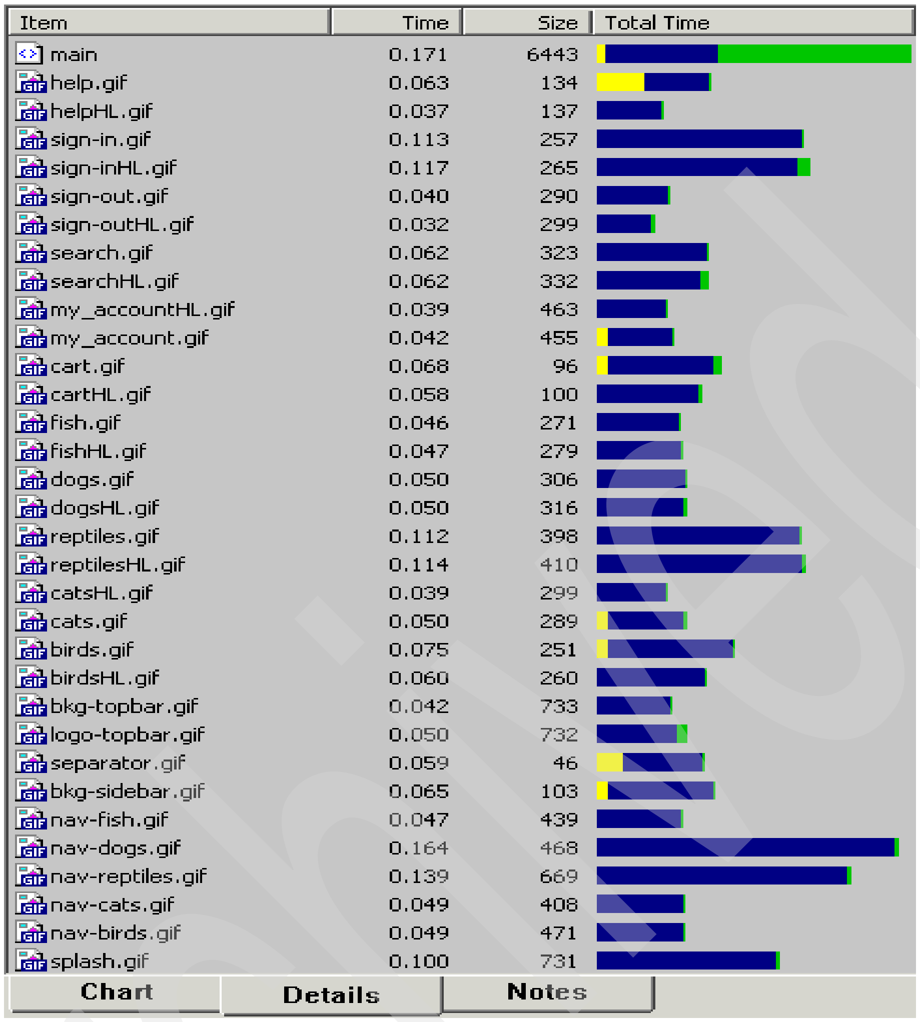
To be sure that static content is served by the IBM HTTP Server and not by the application server, you can disable static content management in the Web container as described in “Integrated scenario” on page 138.

4.4.4 Performance considerations

The same generic rules apply as described in 4.3.4, “Performance considerations” on page 111.

For our tests, dynamic fragment caching was enabled in the application server. We ran two performance measurements, one with and one without additional FRCA caching in the IBM HTTP Server on z/OS. As a test case we measured the welcome page of our sample application.

With FRCA disabled, the results we experienced are shown in Figure 4-10 on page 118.



Item	Time	Size	Total Time
main	0.171	6443	
help.gif	0.063	134	
helpHL.gif	0.037	137	
sign-in.gif	0.113	257	
sign-inHL.gif	0.117	265	
sign-out.gif	0.040	290	
sign-outHL.gif	0.032	299	
search.gif	0.062	323	
searchHL.gif	0.062	332	
my_accountHL.gif	0.039	463	
my_account.gif	0.042	455	
cart.gif	0.068	96	
cartHL.gif	0.058	100	
fish.gif	0.046	271	
fishHL.gif	0.047	279	
dogs.gif	0.050	306	
dogsHL.gif	0.050	316	
reptiles.gif	0.112	398	
reptilesHL.gif	0.114	410	
catsHL.gif	0.039	299	
cats.gif	0.050	289	
birds.gif	0.075	251	
birdsHL.gif	0.060	260	
bkg-topbar.gif	0.042	733	
logo-topbar.gif	0.050	732	
separator.gif	0.059	46	
bkg-sidebar.gif	0.065	103	
nav-fish.gif	0.047	439	
nav-dogs.gif	0.164	468	
nav-reptiles.gif	0.139	669	
nav-cats.gif	0.049	408	
nav-birds.gif	0.049	471	
splash.gif	0.100	731	

Figure 4-10 Welcome page processing of our sample application with FRCA off

For an explanation of the color codes refer to 4.3.4, “Performance considerations” on page 111.

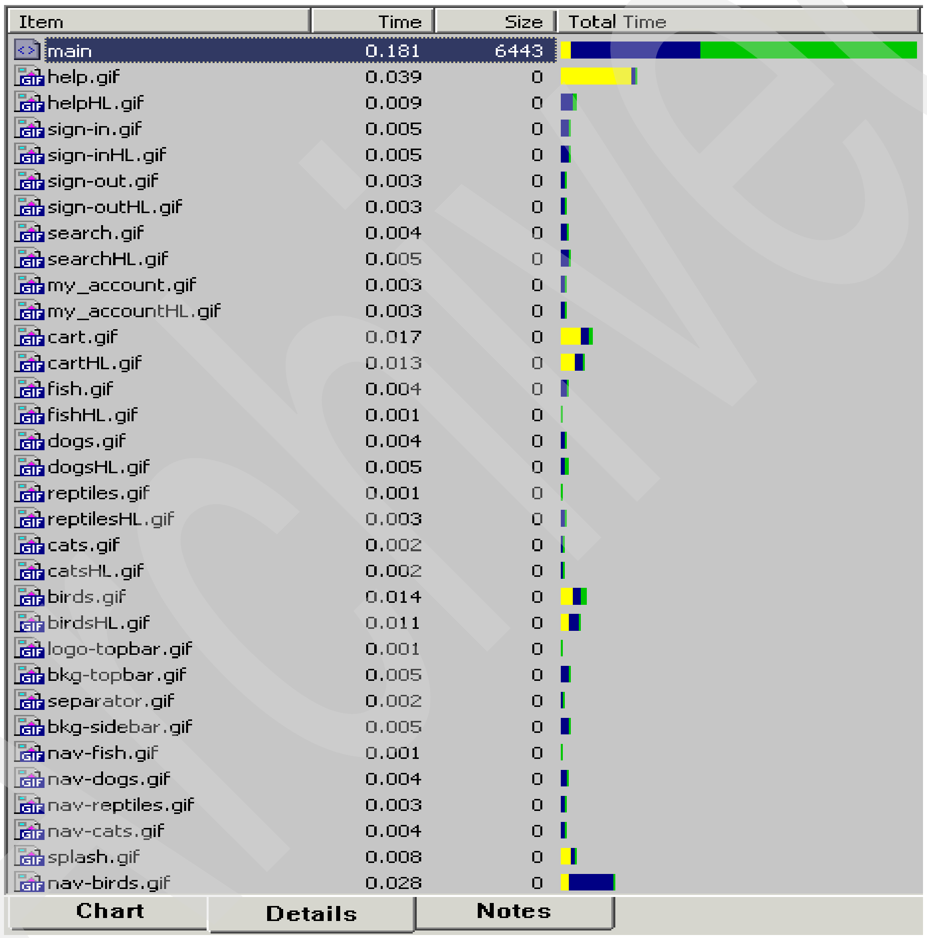
In Example 4-5 we also present some numerical data, as a sample, for one of the several gif images in the application.

Example 4-5 Times and sizes for logo-topbar.gif with FRCA off

```
Item: logo-topbar.gif
           Type: image/gif
            URL: http://wtsc59.itso.ibm.com:8085/estore/images/logo-topbar.gif
           Date: 03/29/2003 16:08:51
    Page Offset: 1.000 seconds
Previous Offset: 0.009 seconds

Response Times:
       0.050 seconds Total Time
       0.044 seconds Server Response Time
       0.006 seconds Delivery Time
       0.002 seconds Delivery Idle Time
       0.050 seconds Local Closed Socket Offset
```

```
Item Sizes:                          Sent    Received      Total
                 HTTP Headers:        419         292        711
             Application Data:          0         732        732
                  Total Bytes:        419        1024       1443
                Overhead Data:        419         292        711

Communication:
          Socket ID: 1264
     Remote Address: 9.12.6.16 : 8085
      Local Address: 127.0.0.1 : 2435
```

For a plug-in configuration with FRCA enabled, we had the results shown in Figure 4-11.



Item	Time	Size	Total Time
main	0.181	6443	
help.gif	0.039	0	
helpHL.gif	0.009	0	
sign-in.gif	0.005	0	
sign-inHL.gif	0.005	0	
sign-out.gif	0.003	0	
sign-outHL.gif	0.003	0	
search.gif	0.004	0	
searchHL.gif	0.005	0	
my_account.gif	0.003	0	
my_accountHL.gif	0.003	0	
cart.gif	0.017	0	
cartHL.gif	0.013	0	
fish.gif	0.004	0	
fishHL.gif	0.001	0	
dogs.gif	0.004	0	
dogsHL.gif	0.005	0	
reptiles.gif	0.001	0	
reptilesHL.gif	0.003	0	
cats.gif	0.002	0	
catsHL.gif	0.002	0	
birds.gif	0.014	0	
birdsHL.gif	0.011	0	
logo-topbar.gif	0.001	0	
bkg-topbar.gif	0.005	0	
separator.gif	0.002	0	
bkg-sidebar.gif	0.005	0	
nav-fish.gif	0.001	0	
nav-dogs.gif	0.004	0	
nav-reptiles.gif	0.003	0	
nav-cats.gif	0.004	0	
splash.gif	0.008	0	
nav-birds.gif	0.028	0	

Figure 4-11 Welcome page processing of our sample application with FRCA on

Again, if you want to compare the results, look at the times, because the bars are scaled differently in each picture.

In numerical data, for the same graphical element, we get the results shown in Example 4-6.

Example 4-6 Times and sizes for logo-topbar.gif with FRCA on

```
Item: logo-topbar.gif
           Type: image/gif
           URL: http://wtsc59.itso.ibm.com:8085/estore/images/logo-topbar.gif
          Date: 03/29/2003 16:21:38
   Page Offset: 0.390 seconds
Previous Offset: 0.015 seconds

Response Times:
       0.001 seconds Total Time
       0.001 seconds Server Response Time

Item Sizes:                       Sent   Received      Total
                 HTTP Headers:     419        194        613
             Application Data:       0          0          0
                  Total Bytes:     419        194        613
                Overhead Data:     419        194        613

Communication:
          Socket ID: 1264
     Remote Address: 9.12.6.16 : 8085
      Local Address: 127.0.0.1 : 2490
```

Comparing the results, it is again clear that additional throughput enhancements can be achieved with FRCA for your static content. You can also see, and this is most important, that you can use FRCA together with the plug-in if you architect accordingly.

4.4.5 Infrastructure considerations for configurations 1 and 2

Architecting an infrastructure means considering many operational, organizational, and application-related aspects. Availability, workload distribution, scalability, and integration aspects are some of them.

High availability considerations

Architecting for high availability is to select and combine bullet proof components. It is also the elimination of single point-of-failure components. zSeries servers and the z/OS operating system make things a lot easier, because many of these characteristics are built into the platform. Anyhow, you still need to think and architect a nice solution to really avoid possible outages. Many of the techniques you can use are described in *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

Because of its tight integration, WebSphere Application Server for z/OS and OS/390 takes advantage of the availability features provided by z/OS and the zSeries platform.

The most common way to achieve high availability is redundancy of components, as shown in Figure 4-12 on page 121. This is a very simple starting point and for simplicity assumes that an integrated topology was selected.

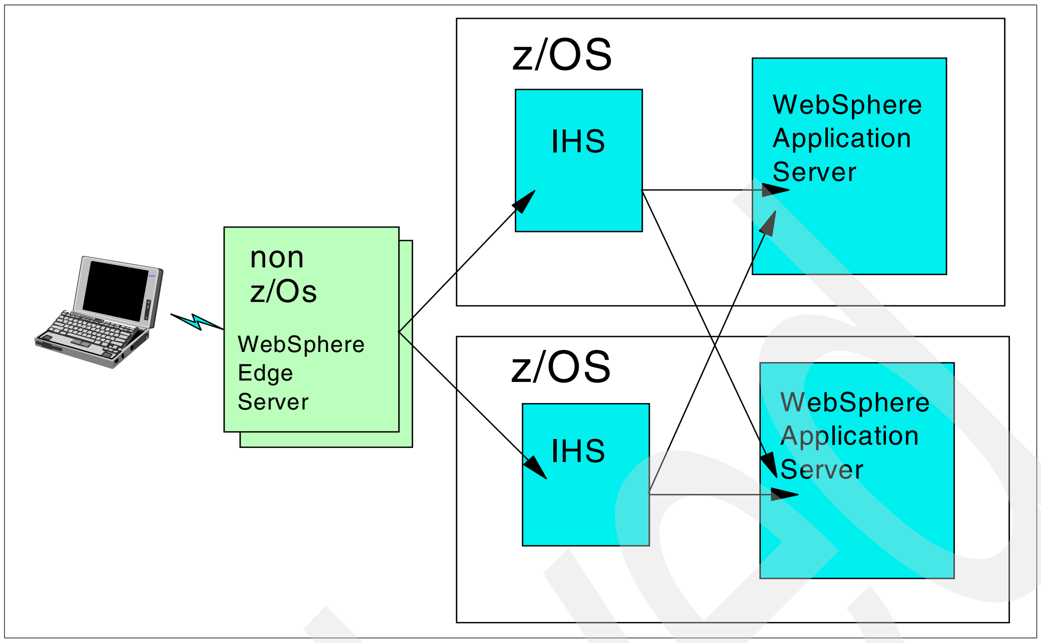


Figure 4-12 Simple high availability scenario for configurations 1 & 2

This example shows redundancy for hardware and software components. Static offloading is made by properly configuring both application servers and both HTTP servers. Any of the possibilities discussed in 4.3, "Configuration 1: Local IBM HTTP Server for static file handling" on page 109 or 4.4, "Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in" on page 115 fits into this availability scheme since the structure and relationship among application servers and HTTP servers described there are preserved.

We can benefit from the WebSphere Edge Server components for several tasks. Using the Network Dispatcher we may improve the availability by clustering application servers and Web servers.

In Figure 4-12, one of the WebSphere Edge Servers has to be configured as primary and the other one as backup so that the backup will take over in case of failure. That's why both of them have to communicate with both HTTP servers.

Scalability and workload distribution

Scalability can be achieved on the z/OS operating system with features like Workload Manager (WLM) support for the whole WebSphere infrastructure and the IBM HTTP Server. The z/Series platform adds features like Capacity Backup to cope with peak loads. The sysplex implementation lets your operating system images scale horizontally.

Horizontal scalability is always very closely connected to high availability topics, such as clustering, cloning, sharing, and workload distribution. These topics are entirely covered in *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

Networking considerations

Configuration 1 and 2 have no specific requirements on the networking topology and should fit on almost all common TCP/IP networks.

DMZ integration and security considerations

Component interaction in configuration 1 and 2 is completely HTTP based. HTTP is nicely flowed through all known firewalls, so this configuration would fit seamlessly into any DMZ environment. Often, corporate security policies do not allow to connect an IBM HTTP Server on z/OS directly into a perimeter network. In this case a reverse proxy within the DMZ can forward the request to the IBM HTTP Server behind the second firewall.

Other DMZ integration options are:

- The enablement of z/OS firewall technologies
- Connecting a z/OS LPAR, with only an IBM HTTP Server for z/OS running, to the DMZ.

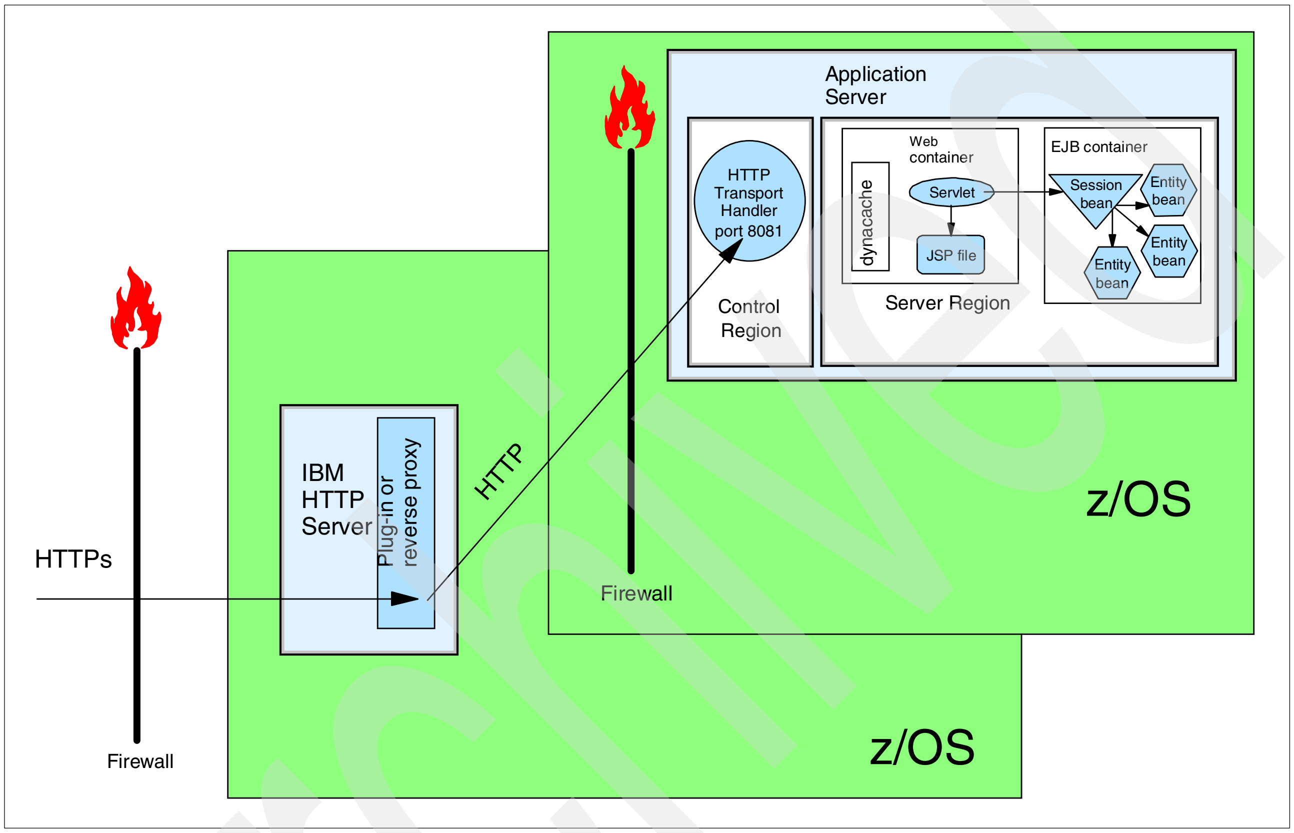


Figure 4-13   Firewall technologies enabled within z/OS

Figure 4-13 shows a possible setup with an LPAR running IBM HTTP Server on z/OS directly connected to the DMZ. Requests from this server are routed through a firewall that is installed on the system that is hosting the WebSphere Application Server for z/OS.

4.5 Configuration 3: Remote reverse proxy caching server

Configuration 3 and 4 (4.6, “Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in” on page 126) are implementing the WebSphere Application Server front end on a distributed platform exploiting either proxy or plug-in functionality. This configuration uses a reverse proxy setup. In a standard proxy cache configuration, the proxy acts as a proxy for the client. In this reverse proxy configuration, the reverse proxy server acts as a proxy and front end for the server. As a reverse proxy cache it can store specific content, whereas proxy and transparent caches store frequently requested content (and maintain their caches automatically).

The major difference from the local configurations is that in this remote configuration the static content is not locally available. To make it locally available to the Web server, you can copy it after the deployment process or export it to the Web server using, for example, a Network File System (NFS) implementation.

Apart form this, the configurations won't differ too much from the local configurations 1 and 2. Also, for the remote configurations we still use all available performance facilities. Dynamic fragment caching is activated in our application server, and proxy caching is used in the IBM HTTP Server. For more details about dynacache, refer to 4.2, "Dynamic fragment caching concepts" on page 105.

This configuration supports Adaptive Fast Path Architecture (AFPA), a static file caching architecture available on distributed platforms. AFPA is a software architecture for high-performance network servers. It is specifically designed to be general purpose. However, most implementations are focussed on Web servers.

The implementation guidelines for this configuration are described in 5.2.4, "Configuration 3: Remote reverse proxy caching server" on page 146.

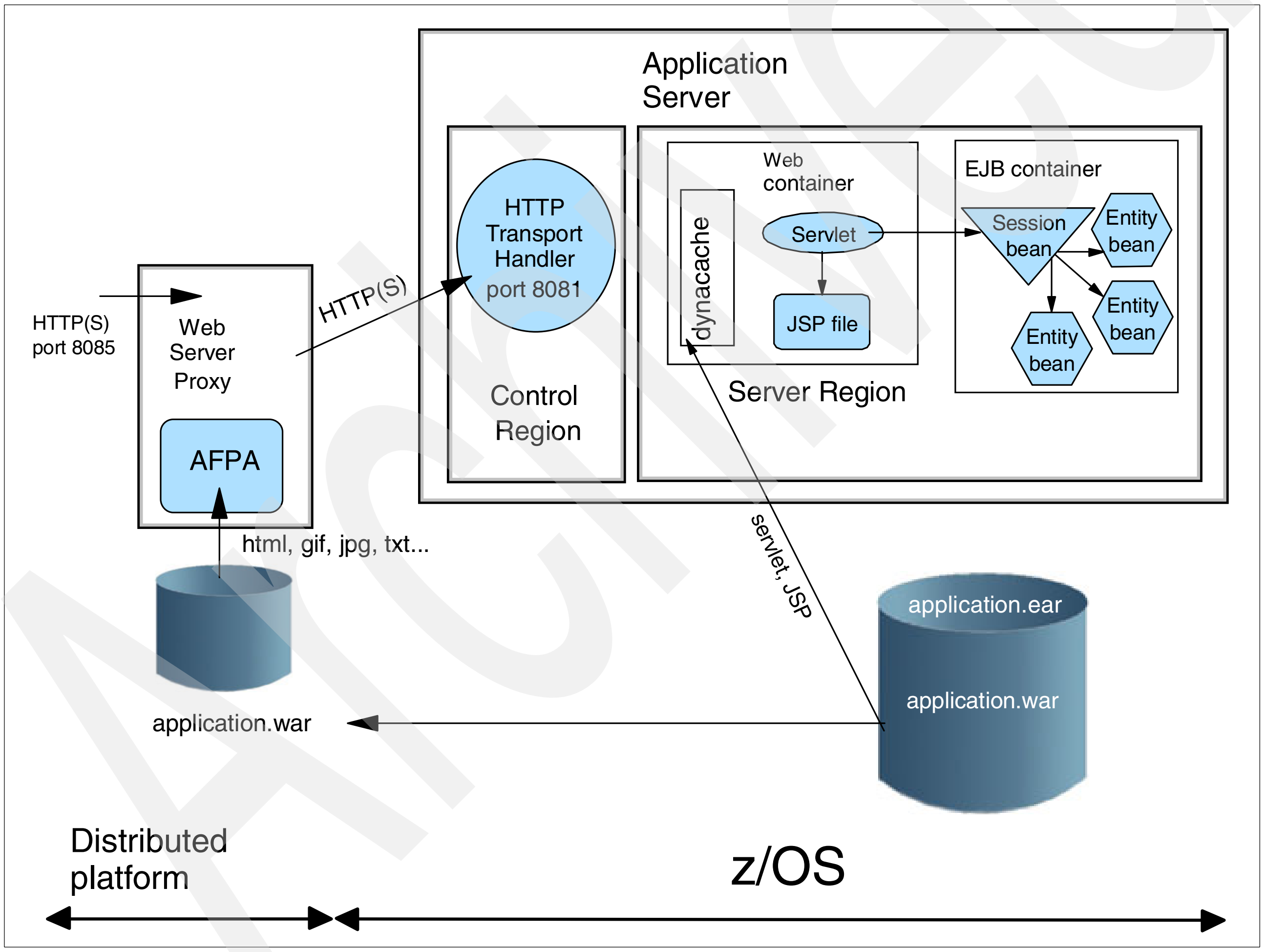


Figure 4-14 Configuration 3: remote IBM HTTP Server as a proxy server

Another possible configuration is to run IBM HTTP Server as forwarding proxy server. A forwarding proxy would have caching functions but no capability to serve local files. Instead, a reverse proxy is able to manage static content and serve it itself while it sends all other requests to the application server.

4.5.1 HTTP session considerations

If you have HTTP sessions in memory and multiple server instances, your infrastructure needs to maintain session affinity. This configuration, as it is, does not support session affinity. You should rather look at the WebSphere HTTP plug-in configurations described in 4.4, "Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in" on page 115 and 4.6, "Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in" on page 126.

4.5.2 Security considerations

See 4.3.2, "Security considerations" on page 110.

Usually you do not have the same user registry on a distributed platform as you do on z/OS (though it is possible to export the RACF database to an LDAP server that can be used in the DMZ for authentication).

Based on this fact it is obvious that it is difficult for this kind of configuration to synchronize the authentication setups. We really recommend that the quality of the static content is such that it can be served in an unprotected fashion.

4.5.3 System management considerations

To allow the IBM HTTP Server to serve static content, it needs to be made locally available to the Web Server.

There are a couple of ways to do this. One is to use NFS or any other kind of network sharing system to make the necessary content available to the HTTP Server. You can also copy the content from the Web container where it is deployed to the Web servers file system.

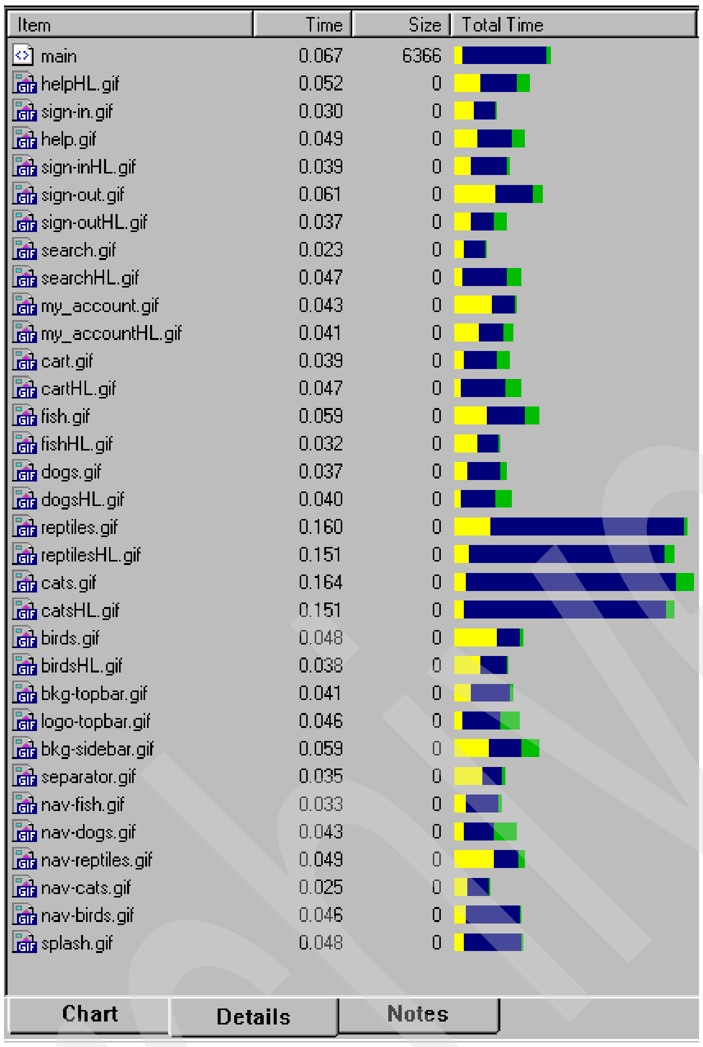
For this simple setup we unpacked the application.war file into the Web server's Document Root directory.

To be sure that all static content was served by the IBM HTTP Server, we disabled static managing in the Web container (see "Disabling file serving in the Web container" on page 138).

4.5.4 Performance considerations

The same generic rules apply as described in 4.3.4, "Performance considerations" on page 111.

For our tests, dynamic fragment caching was enabled in the application server. As a test case, we measured the response time of the welcome page of our sample application; see Figure 4-15 on page 125.



Item	Time	Size	Total Time
main	0.067	6366	
helpHL.gif	0.052	0	
sign-in.gif	0.030	0	
help.gif	0.049	0	
sign-inHL.gif	0.039	0	
sign-out.gif	0.061	0	
sign-outHL.gif	0.037	0	
search.gif	0.023	0	
searchHL.gif	0.047	0	
my_account.gif	0.043	0	
my_accountHL.gif	0.041	0	
cart.gif	0.039	0	
cartHL.gif	0.047	0	
fish.gif	0.059	0	
fishHL.gif	0.032	0	
dogs.gif	0.037	0	
dogsHL.gif	0.040	0	
reptiles.gif	0.160	0	
reptilesHL.gif	0.151	0	
cats.gif	0.164	0	
catsHL.gif	0.151	0	
birds.gif	0.048	0	
birdsHL.gif	0.038	0	
bkg-topbar.gif	0.041	0	
logo-topbar.gif	0.046	0	
bkg-sidebar.gif	0.059	0	
separator.gif	0.035	0	
nav-fish.gif	0.033	0	
nav-dogs.gif	0.043	0	
nav-reptiles.gif	0.049	0	
nav-cats.gif	0.025	0	
nav-birds.gif	0.046	0	
splash.gif	0.048	0	

Figure 4-15 Welcome page processing of our sample application with AFPA on

For an explanation of the color codes, refer to 4.3.4, “Performance considerations” on page 111.

Some numerical data for one of the several gif images in the application explains the performance behavior of individual elements in detail, as shown in Example 4-7.

Example 4-7 Times and sizes for logo-topbar.gif

```
Item: logo-topbar.gif
           Type: image/gif
            URL: http://wtsc59.itso.ibm.com:8085/estore/images/logo-topbar.gif
           Date: 03/31/2003 21:21:15
    Page Offset: 0.786 seconds
Previous Offset: 0.008 seconds

Response Times:
       0.046 seconds Total Time
       0.006 seconds Connect Time
       0.026 seconds Server Response Time
       0.014 seconds Delivery Time
       0.001 seconds Delivery Idle Time
```

```
      0.051 seconds Local Closed Socket Offset

Item Sizes:                          Sent   Received      Total
                 HTTP Headers:        437        227        664
             Application Data:          0          0          0
                  Total Bytes:        437        227        664
                Overhead Data:        437        227        664

Communication:
          Socket ID: 1656
     Remote Address: 10.1.15.3 : 80
```

As in other figures, if you want to compare results, look at the times column because the bars are differently scaled in each picture. Of course, these results can't be regarded as exhaustive, but are useful to get an idea of the behavior of each configuration. If you compare these numbers with the measurements for the integrated scenarios, you will see that the integrated scenarios perform slightly better.

Again, our performance measurements were done to get a feeling of how the scenarios behave, but no scientific rules were applied. Anyhow, as a platform on the distributed side, we used high-end PC servers.

4.6 Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in

This setup is almost similar to the one described in 4.4, "Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in" on page 115: we have an IBM HTTP Server with the WebSphere HTTP Plug-in enabled sending requests to the active Web container.

As shown in Figure 4-16 on page 127, for this setup the HTTP server needs the application's static content available locally. Therefore, the Web application's content needs to be copied or exported using a network file system into the Web server's file system.

The implementation guidelines for this scenario can be found in 5.2.5, "Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in" on page 149.

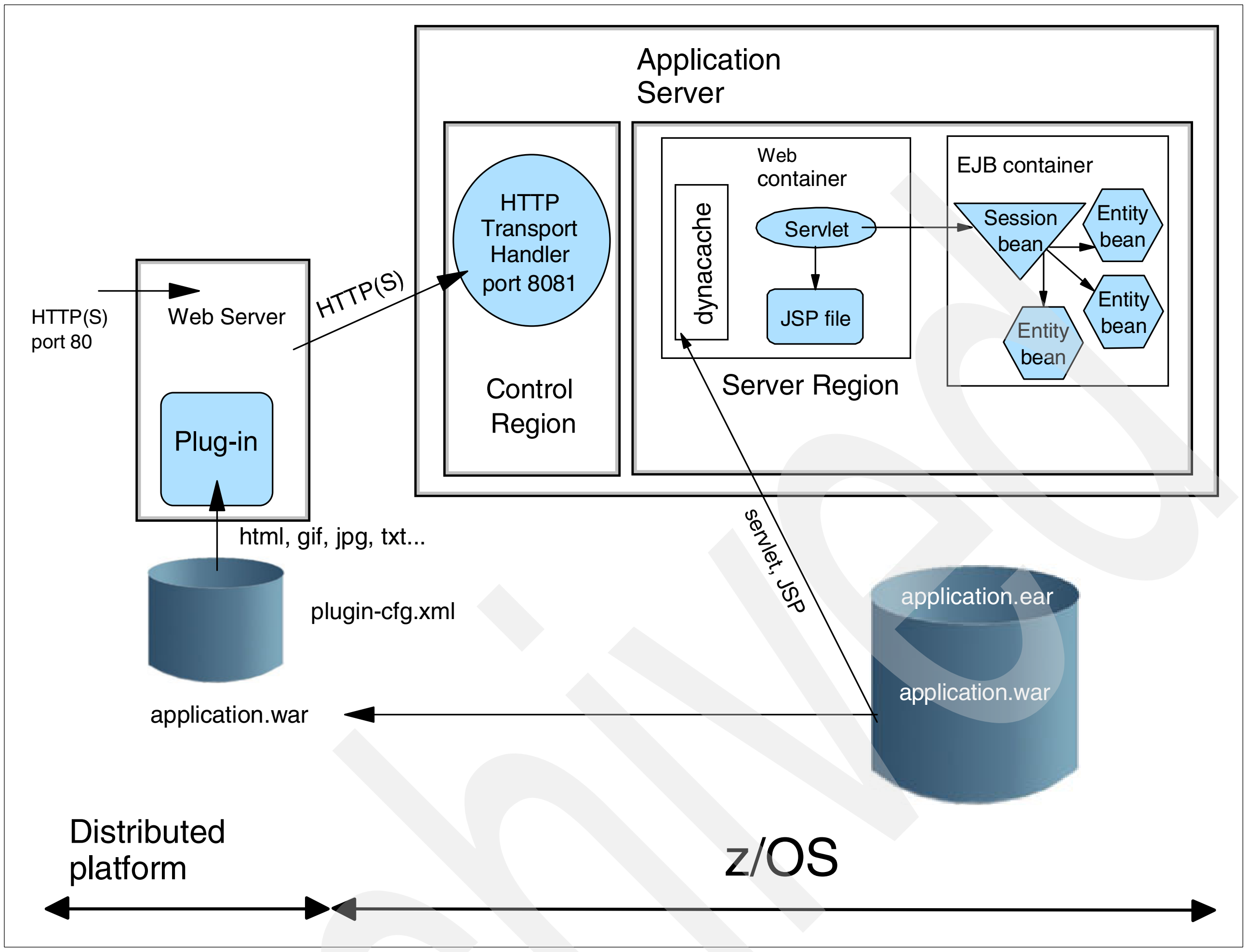


Figure 4-16 IBM HTTP Server for Windows with WebSphere HTTP Plug-in

Figure 4-16 depicts the components of configuration 4.

4.6.1 HTTP session considerations

Distributed WebSphere HTTP plug-ins have the same functionalities as the ones for z/OS. Refer to 4.4.1, “HTTP session considerations” on page 117.

4.6.2 Security considerations

The same security considerations as for scenario 3 apply. See 4.5.2, “Security considerations” on page 124.

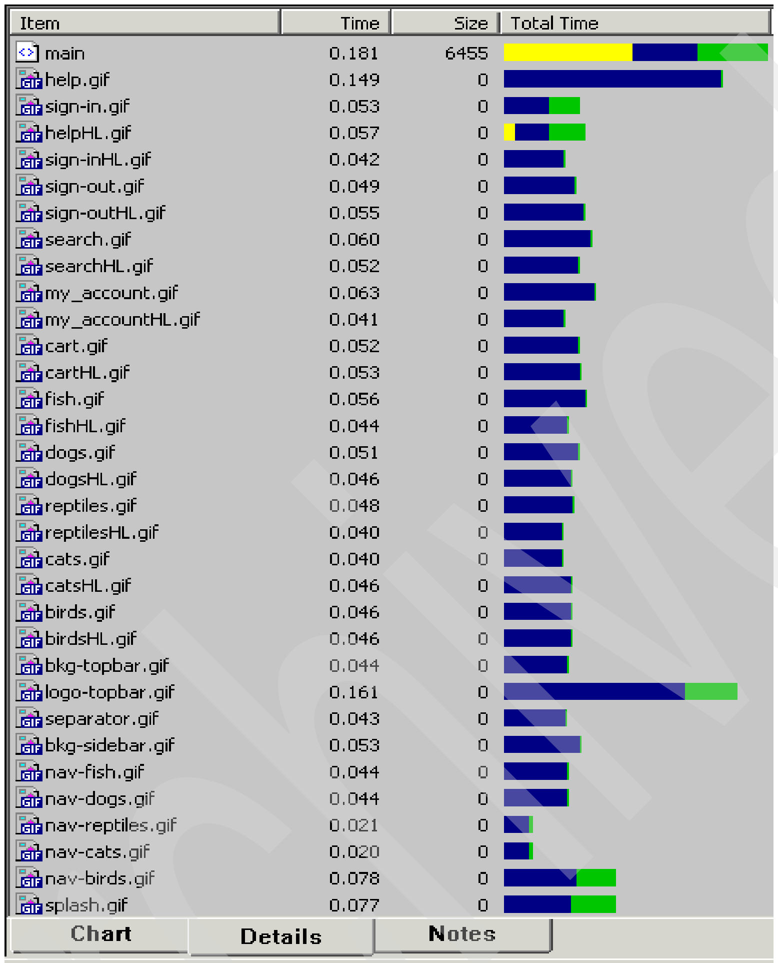
4.6.3 System management considerations

See 4.5.3, “System management considerations” on page 124.

4.6.4 Performance considerations

The same generic rules apply as described in 4.3.4, “Performance considerations” on page 111.

For our tests, dynamic fragment caching was enabled in the application server. FRCA or AFPA caching could not be enabled in our configuration. As a test case, we measured the response time of the welcome page of our sample application; see 4.3.4, “Performance considerations” on page 111.



Item	Time	Size	Total Time
main	0.181	6455	
help.gif	0.149	0	
sign-in.gif	0.053	0	
helpHL.gif	0.057	0	
sign-inHL.gif	0.042	0	
sign-out.gif	0.049	0	
sign-outHL.gif	0.055	0	
search.gif	0.060	0	
searchHL.gif	0.052	0	
my_account.gif	0.063	0	
my_accountHL.gif	0.041	0	
cart.gif	0.052	0	
cartHL.gif	0.053	0	
fish.gif	0.056	0	
fishHL.gif	0.044	0	
dogs.gif	0.051	0	
dogsHL.gif	0.046	0	
reptiles.gif	0.048	0	
reptilesHL.gif	0.040	0	
cats.gif	0.040	0	
catsHL.gif	0.046	0	
birds.gif	0.046	0	
birdsHL.gif	0.046	0	
bkg-topbar.gif	0.044	0	
logo-topbar.gif	0.161	0	
separator.gif	0.043	0	
bkg-sidebar.gif	0.053	0	
nav-fish.gif	0.044	0	
nav-dogs.gif	0.044	0	
nav-reptiles.gif	0.021	0	
nav-cats.gif	0.020	0	
nav-birds.gif	0.078	0	
splash.gif	0.077	0	

Figure 4-17 Welcome page processing of our sample application

For an explanation of the color codes, refer to 4.3.4, “Performance considerations” on page 111. As in other figures, if you want to compare the results, look at the time column because the bars are scaled differently in each picture.

And the numerical data for one of the gif images (same as in other examples) is shown in Example 4-8.

Example 4-8 Times and sizes for logo-topbar.gif with WebSphere Plug-in

```
Item: bkg-topbar.gif
           Type: gif
            URL: http://tot134.itso.ibm.com/estore/images/bkg-topbar.gif
           Date: 03/31/2003 12:22:14
    Page Offset: 1.364 seconds
Previous Offset: 0.038 seconds

Response Times:
       0.044 seconds Total Time
       0.044 seconds Server Response Time
```

```
Item Sizes:                          Sent   Received      Total
                 HTTP Headers:        408        203        611
             Application Data:          0          0          0
                  Total Bytes:        408        203        611
                Overhead Data:        408        203        611

Communication:
          Socket ID: 1220
     Remote Address: 10.1.15.3 : 80
      Local Address: 127.0.0.1 : 2450
```

Of course, these results cannot be considered exhaustive, but are useful for getting an idea of the behavior of the configuration. And again, comparing the data with the integrated configurations shows interesting views.

4.6.5 Infrastructure considerations for configurations 3 and 4

Architecting an infrastructure means considering many operational, organizational and application-related aspects. Availability, workload distribution, scalability, and integration aspects are some of them.

High availability considerations

Architecting for high availability is to select and combine bullet proof components. It also means the elimination of single point-of-failure components. zSeries servers and the z/OS operating system make things a lot easier, because many of these characteristics are built into the platform. Anyhow, you still need to architect a nice solution to really avoid possible outages. Many of the techniques that you can use are described in *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

Due to its tight integration, WebSphere Application Server for z/OS and OS/390 takes advantage of the availability features provided by z/OS and the zSeries platform.

For the components that are not living in the zSeries server, you need to take some extra steps to make them reliable and to ensure that they won't disrupt your valuable service on the WebSphere Application Server for z/OS and OS/390.

The most common way to achieve high availability is redundancy of components, as shown in Figure 4-12 on page 121. All components that are involved in passing the request forward to the Web container need to be redundant. Takeover and distribution mechanisms need to take care that in case of a failure the alternate element is immediately selected.

Some architectural elements, such as WebSphere Edge Server, have built-in high availability mode (hot stand-by, MAC takeover), as shown in Figure 4-18 on page 130. It is easier to architect highly available topologies if you can use servers that deliver built-in high availability.

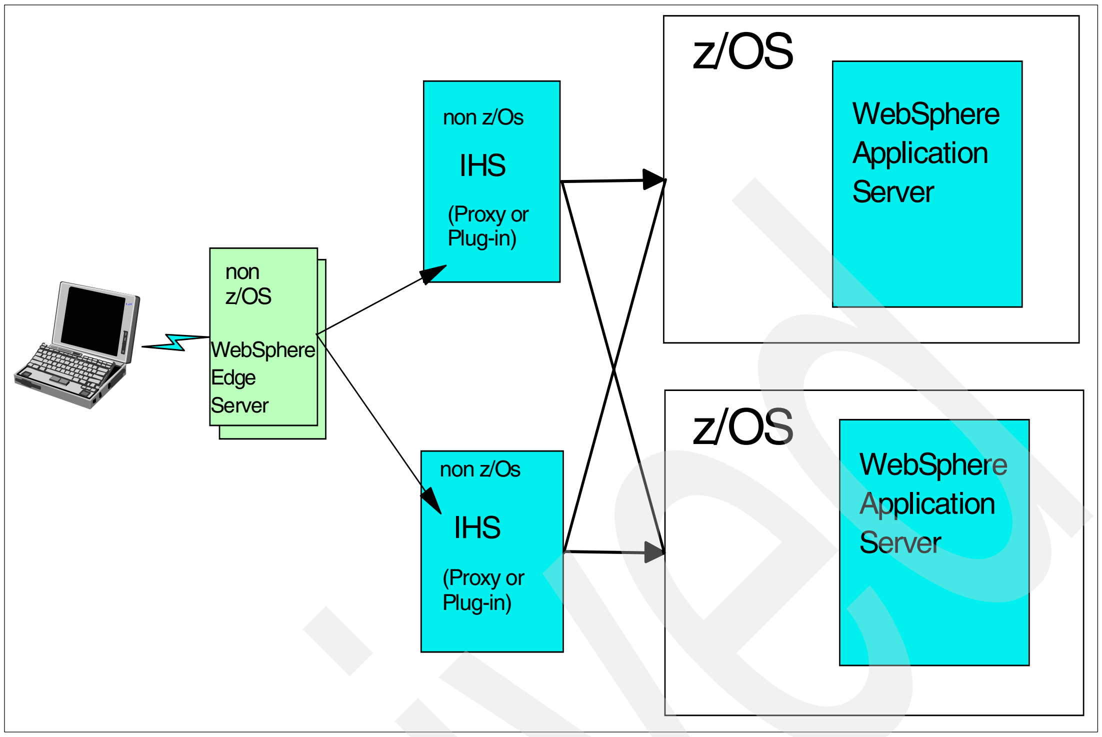


Figure 4-18 High availability for configurations 3 & 4

For specific discussions about availability, refer to *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

Scalability and workload distribution

Scalability can be achieved on the z/OS operating system with features such as Workload Manager Support for the whole WebSphere infrastructure and the IBM HTTP Server. The z/Series platform adds features like Capacity Backup to cope with peak loads. The sysplex implementation lets your operating system images scale horizontally.

Horizontal scalability is always very closely connected to high availability topics, such as clustering, cloning, sharing, and workload distribution. All these topics are entirely covered in *Enabling High Availability e-business on e-server zSeries,* SG24-6850.

WebSphere Edge Server or another third-party IP sprayer plays a central role for workload distribution to the distributed Web servers. We may add Web servers as the need increases for dispatching more requests.

The Web Servers with the plug-in then are in charge of forwarding requests to other intelligent workload distribution mechanisms, or directly to the Web container.

Networking considerations

See "Networking considerations" on page 121.

DMZ integration and security considerations

Using standard Web servers on distributed platforms an purely HTTP(S), it is very easy to plug into existing DMZ zones.

Figure 4-19 on page 131 depicts this DMZ integration.

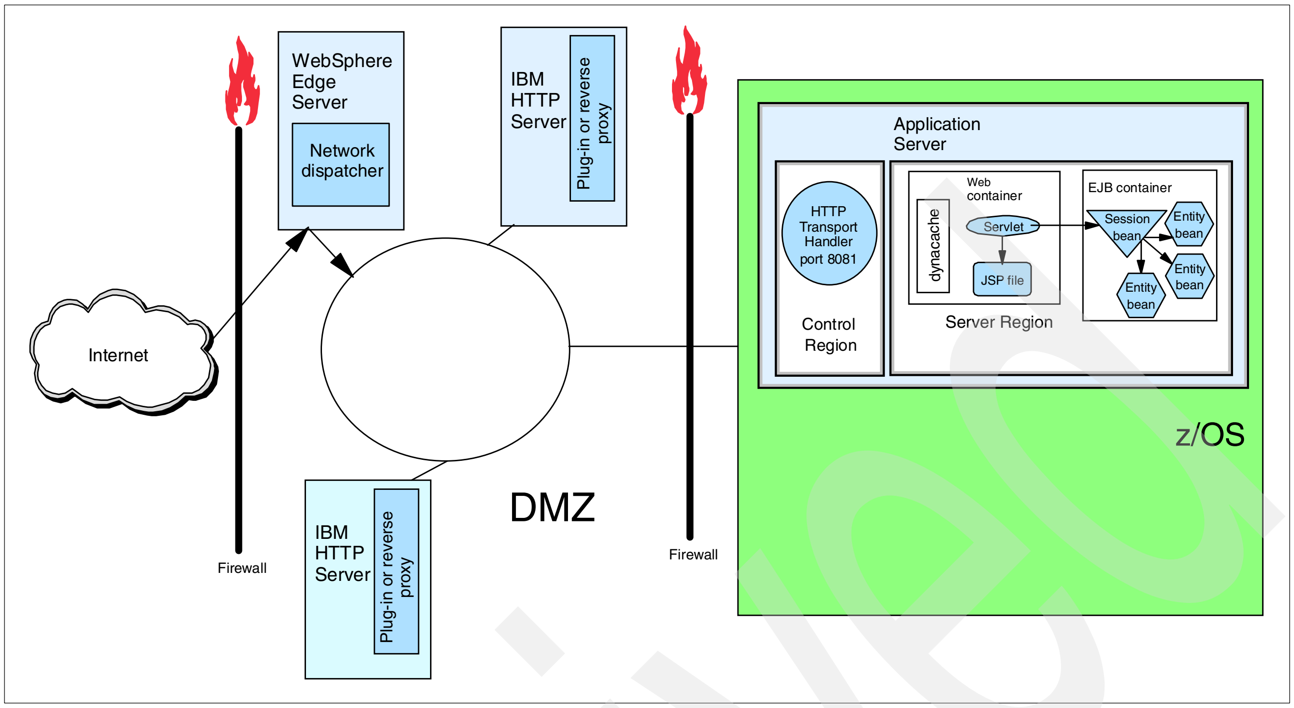


Figure 4-19 DMZ with a WebSphere Edge Server and a couple of IBM HTTP Servers

Adapt this example according to how your tiers and network structure are divided.

4.7 Application considerations

Static content response time acceleration is not described in the J2EE standards. There are development and application assembly aspects to support static content off-loading sufficiently.

4.7.1 Application programming and assembly

Most programming models used today are dynamic; presentation layers are managed by JSPs.

A Web site today has a consistent look and feel. Design elements such as banners, navigation bars, and buttons are constantly replayed. The corporate identity is represented by consistent images and colors on every page.

All this content is static and should be delivered accordingly. Application architects need to consider this when designing new applications.

4.8 Trends and directions

We discussed and tested four different scenarios, but many more are suitable. These configurations are particularly suitable for Linux, either in a distributed flavor or Linux for zSeries.

zLinux is particularly interesting because you can have different servers in a logical partition together with WebSphere Application Server for z/OS and OS/390 in a different LPAR inside the same box; this allows communication integrity between the Web server and the application server.

It is possible to have the whole infrastructure needed inside one box. Proxy servers, DNS servers, Web servers, and workload distributors can live here together with your z/OS system hosting the WebSphere Application Server and backend systems.

Figure 4-20 depicts a complex mix and match environment that also incorporates static offloading in zLinux, as explained in 4.6, “Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in” on page 126.

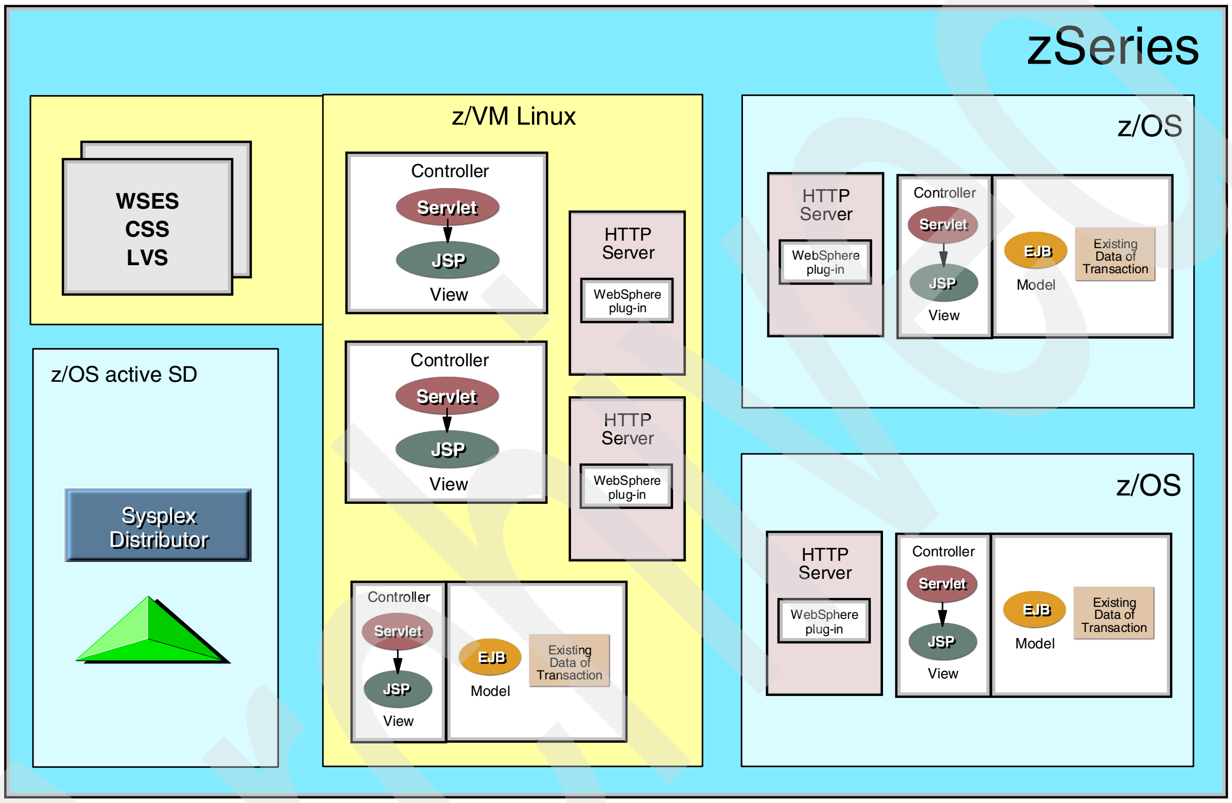


Figure 4-20 Static offloading using IBM HTTP Server with WebSphere Plug-in on Linux for zSeries

Another kind of offload is available when you use the new features and functions offered by WebSphere Edge Server. Application offload can work moving the presentation layer and some business logic to the WebSphere Edge Server. The applications have to be prepared for that and we talk about “edgified applications”. There are also special deployment tools. For further information, refer to *WebSphere Edge Server: New Features and Functions in Version 2,* SG24-6511.

Part 2

Implementation guidelines

This part of the book introduces the method of separation in Web-tier, and the method of separation on IIOP boundary between Web-tier and EJB-tier using a sample application (Java Pet Store Demo). It covers how to analyze an application, how to configure and modify this specific application, how to set up the infrastructure, and how to deploy the application into a multi server environment.

In Chapter 5, “Implementing static Web content acceleration scenarios” on page 135, we introduce four configuration s about offloading static Web contents. These configurations are mainly aimed at optimization of the performance of the application.

In Chapter 6, “Implementing IIOP-based cross-platform scenarios” on page 153, we aim at the modification of the application so that it will be useful when you separate real applications.

5

Implementing static Web content acceleration scenarios

In this chapter, the following topics are discussed:

- How to analyze the application to search for static and dynamic contents (5.1.1, “Analyzing the application” on page 136)
- How to set up the application for optimum management from the static, and sometimes dynamic, point of view (5.1.2, “Assembling the application” on page 138, and 5.1.3, “Deploying the application” on page 138)
- What possible infrastructure is needed (5.2, “Infrastructure implementation” on page 140)
- How to set up that infrastructure (5.2, “Infrastructure implementation” on page 140)

We show, with examples, how you can implement the solutions discussed in Chapter 4, “Static Web component optimization” on page 101.

5.1 Application development and deployment

Various aspects need to be considered when you are preparing a given application for hybrid deployment. Application inspection and analysis, development, assembly and deployment are covered here.

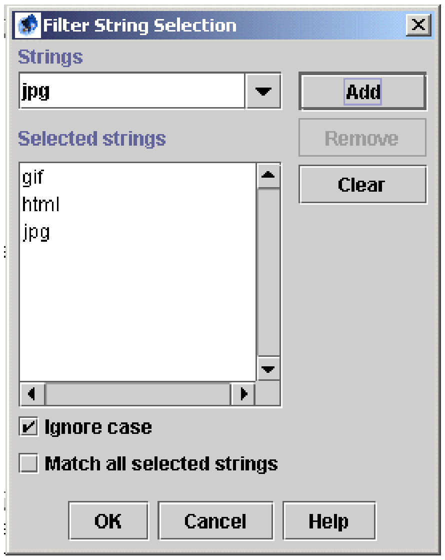
5.1.1 Analyzing the application

For optimized management of the static and semi-static contents (or semi-dynamic, depending on how we understand the behavior of the JSPs), an analysis of the application is recommended. It is important to differentiate contents because the resources available from the system are different for completely static contents, like HTML files, than those for JSPs or servlets. Some resources will be provided by the IBM HTTP Server, others by WebSphere Application Server for z/OS and OS/390, or WebSphere Application Server Enterprise V5. However, for the z/OS platform, we had to restrict our tests with the application server to WebSphere Application Server for z/OS and OS/390.

Static contents

You need to know how many static contents there are in your application, and their paths, to have a picture of the configuration changes you will have to make. There are several ways to do this if you can't ask the developers of the application; the most common three are (in order of difficulty):

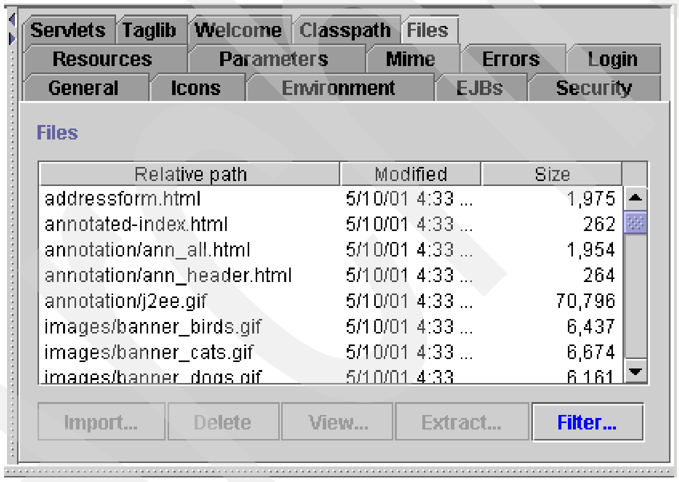
- Unpack the application into one of the directories of your PC and then go to **Start -> Search -> Files or Folders**, and search for the HTML, GIF and JPG files under the directory where the Java Pet Store Demo has been unpacked.
- Open the .ear file containing the application with a compression tool. Inside the .ear file there are the .war files with the Web applications; open them again with the compression tool and order them by type.
- The last possibility is to start the Application Assembly Tool (AAT) and import the applicationName.ear file to be deployed (if it isn't already). To view the application contents, it is necessary to expand the tree and select **Application name -> Web Apps -> Web Tier**; in the Files tab, click **Filter**, then add the extensions of the files you want to look at (Figure 5-1 on page 137). In this way, we get a list (Figure 5-2 on page 137) with all the static contents for this Web application and the relative paths under the applicationName.war file.



Ignore case: [✓]
Match all selected strings: [ ]

Figure 5-1 Filtering static contents with AAT

Of course, this list will not give you any idea of the percentage of static content you have in your application, but you will have the URIs to create the necessary directives for the HTTP Server or to find your way through the application paths, if you don't have the developers at hand.



Relative path	Modified	Size
addressform.html	5/10/01 4:33 ...	1,975
annotated-index.html	5/10/01 4:33 ...	262
annotation/ann_all.html	5/10/01 4:33 ...	1,954
annotation/ann_header.html	5/10/01 4:33 ...	264
annotation/j2ee.gif	5/10/01 4:33 ...	70,796
images/banner_birds.gif	5/10/01 4:33 ...	6,437
images/banner_cats.gif	5/10/01 4:33 ...	6,674
images/banner_dogs.gif	5/10/01 4:33	6,161

Figure 5-2 Static content filtered list

Dynamic content

Enabling caching for the dynamic contents is not as straight orward as for the static. First you have to get a good understanding of the application architecture and workflow for the servlets and JSPs.

What we found is that most of the screens in our sample application are built using several JSPs. There are a couple of xml files used to bind the URIs to the screens and the screens to the JSPs taking part in each of them. The file for binding of the URIs to the screen is requestmappings.xml, and the one defining which JSPs build each screen is screendefinitions.xml. This is quite normal in the JSP tag standard.

After this we had to check the JSPs, one by one, to know if we needed to code cache variables for any of the JSPs. This can be very time consuming for large applications with many JSPs. There were a number of JSPs that had attributes depending on the language used for the application (English or Japanese were available). Since we were just working with the English version, we could simplify the coding. Other JSPs needed parameters (Example 5-2 on page 141) that had to be coded to avoid unexpected effects, such as selecting a product and getting a different one on your screen.

For further information about dynacache parameters, see *Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836.

5.1.2 Assembling the application

No special assembling is necessary for this topic. The only recommendation would be to keep all static contents under one path, if possible, to reduce the number of pass directives (if using the proxy approach), and the search time for the contents.

5.1.3 Deploying the application

There are some recommended practices and tasks for deploying the static and semi-static contents to the infrastructure we are using.

Generally, there are no changes to the normal deployment process. What we have done to get better management of our contents is to give the IBM HTTP Servers we were using access to the contents that should be served by them. Therefore, we can differentiate, once more, between the integrated and the cross-platform scenario.

Integrated scenario

This is the easiest way from the deployment point of view. Since all the application components remain in the same platform (z/OS environment for us), there are not many changes in the configuration (see 5.2.2, “Configuration 1: Local IBM HTTP Server for static file handling” on page 142, and 5.2.3, “Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in” on page 144), and few changes also for the deployment process.

When we deploy an application to the WebSphere Application Server for z/OS and OS/390, it resides entirely in the HFS structure, including the .ear file used for that deployment. We can decide whether the static contents are managed by the Web container or not. This decision depends on the infrastructure we are using. If somebody else, such as an HTTP server or the WebSphere Edge Server, is taking care of these contents, we can disable them; if not, then of course we cannot.

Note: If static content serving is disabled in the Web container, be sure that these files will be served by the Web server. If you use this, neither image nor HTML files will be sent from the Web container to the client.

Disabling file serving in the Web container

To disable static contents managing in the Web container, you have to look for the file ibm-web-ext.xmi in the .war file of the application (which is also in the .ear file) and change the parameter fileServingEnable to “false”. The usual path is: <install-path>/applicationname/webapplication.war/WEB-INF/ibm-web-ext.xmi.

To activate this change, you have to recycle the J2EE server.

This can also be done during the process of creating the .ear file. In WebSphere Studio Application Developer V4 (WSAD) go to the project where the application is, project/webapplication/web-inf/ibm-web-ext.xmi.

If you try to execute the application using the port given to the Transport Handler (TH), you will notice that no images are sent to your browser, but if you try to execute the application server through the IBM HTTP Server configured as in 5.2.2, “Configuration 1: Local IBM HTTP Server for static file handling” on page 142, you will see that the application’s visual elements are composed.

Cross-platform scenario

These considerations about handling static contents in the Web container are also applicable in this one scenario.

The same parameter with the same function is used in WebSphere Application Server Enterprise V5 for distributed platforms, and the way to activate and deactivate it is the same as in the previous section.

Anyway, if you choose a cross-platform model with the IBM HTTP Server in a platform other than z/OS, you have to do some additional steps after the deployment to WebSphere Application Server for z/OS and OS/390 has been done.

You have to move all the static contents to the platform where the IBM HTTP Server is working. This is considered here as part of the deployment process, though we don’t use any of the specific tools for that task, because the application will not be fully working in the model we have chosen till these steps are done.

To implement this, after the deployment of the .ear file, we had to follow the path till we got into the applications (/petstore.war) directory. Then we had to move or copy this folder and all its contents to the place where the IBM HTTP Server is, and finally configure the Web server to tell it where the static contents were. For this configuration, see 5.2.4, “Configuration 3: Remote reverse proxy caching server” on page 146.

5.1.4 Testing the application

To test the functionality of the application and the infrastructure supporting it, we usually performed three tests.

- The first one was to navigate across the application manually, checking all the screens and transactions.
- For the second one we used the IBM WebSphere Studio Workload Simulator. With this tool we recorded some scripts navigating through the application and executed them over a long time period.
- For the last test, we used the same scripts but executed them with a big workload, with several users and repetitions.

We show a portion of a script as example (Example 5-1) of the tests we performed. Of course, the tool allows options such as setting the think time, selecting the number of clients, or the use of dynamic cookies. See *WebSphere Studio Workload Simulator Programming Reference,* SC31-6308 and *WebSphere Studio Workload Simulator User’s Guide,* SC31-6307 for further information about the options and possibilities of this tool.

Example 5-1 Part of a script used to test the application

```
//HTTPServer Plugin on z/OS
startpage(1);
```

```
thinktime(0);
   getpage("wtsc59.itso.ibm.com:8085","/estore/control/main",1,close,0,start,"",
   "Accept: */*",
   "Accept-Language: en-us",
   "Accept-Encoding: gzip, deflate",
   "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)",
   "Host: wtsc59.itso.ibm.com:8085",
   "Connection: Keep-Alive",
   "Authorization: Basic ZGllZ286ZGllZ28=",
   "Cookie: JSESSIONID=0000aOMziJ9OBZqgu3Wapy5RtU4:BBOHYB.BBOHYBA1");

   get("wtsc59.itso.ibm.com:8085","/estore/images/help.gif",1,close,130,start,"",
   "Accept: */*",
   "Referer: http://wtsc59.itso.ibm.com:8085/estore/control/main",
   "Accept-Language: en-us",
   "Accept-Encoding: gzip, deflate",
   "If-Modified-Since: Thu, 13 Mar 2003 20:52:51 GMT",
   "User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)",
   "Host: wtsc59.itso.ibm.com:8085",
   "Connection: Keep-Alive",
   "Authorization: Basic ZGllZ286ZGllZ28=",
   "Cookie: JSESSIONID=0000aOMziJ9OBZqgu3Wapy5RtU4:BBOHYB.BBOHYBA1");
```

5.2 Infrastructure implementation

To manage the static contents of an application and serve them out of the Web container of the WebSphere Application Server for z/OS and OS/390, you can benefit from the IBM HTTP Server (either in z/OS or distributed). We tested two possibilities (see Chapter 4, “Static Web component optimization” on page 101): Use the HTTP server as a reverse proxy to forward requests to the HTTP Transport Handler (TH), and use the IBM HTTP Server with the WebSphere HTTP Plug-in.

Tip: Even if all our scenarios are based on WebSphere HTTP Plug-in architecture, be aware that *not* implementing the Plug-in will give the best performance for your application. Then you will lose functionality that is delivered by the Plug-in.

We always used the same scenario with different configurations: An IBM HTTP Server processing and sending requests to our sample application deployed in the WebSphere Application Server for z/OS and OS/390.

Taking this into account, we have four configurations for our infrastructure as conceptually explained in Chapter 4, “Static Web component optimization” on page 101:

- 5.2.2, “Configuration 1: Local IBM HTTP Server for static file handling” on page 142
- 5.2.3, “Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in” on page 144
- 5.2.4, “Configuration 3: Remote reverse proxy caching server” on page 146
- 5.2.5, “Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in” on page 149

All were set up and used as a means for offloading some work from the Web container.

5.2.1 Common elements of the configurations

IBM HTTP Server for z/OS

The HTTP Server we used on the z/OS platform for our test was installed and configured almost by default except for the necessary directives needed for the different functionalities we wanted to get. As a rule, we made as few changes as possible to the original configuration.

IBM HTTP Server for distributed paltforms

The same considerations as for the previous paragraph apply to the IBM HTTP Server for distributed paltforms.

WebSphere Application Server for z/OS and OS/390

The application server we used was accessed using the Transport Handler listening on port 8081. All the accesses from the different IBM HTTP Servers we described till now were using this port because the protocol used by the HTTP servers is HTTP in both modes, proxy and Plug-in (we used the WebSphere HTTP Plug-in for z/OS and for distributed platforms).

This also gave us the opportunity to test the application server by sending requests to the Transport Handler directly from our browsers.

Dynamic fragment caching

We used dynamic fragment caching functions provided by the application server. These functions can be used independently of the HTTP Server and are quite useful when the application has many servlets and JSPs, as is the case with the sample application we used.

In real-life applications, the complexity and amount of the JSPs could be greater than the examples we are showing here. A deeper analysis of each JSP and servlet of the application should be necessary to get better dynamic caching policies. For example, some of the JSPs in this application need a parameter as an argument when they are invoked; those JSPs need special coding in the servletcache.xml file (see Example 5-2 on page 141). In this sample file there are two JSPs with parameters coded, productcatergory.jsp and product.jsp.

For more details about parameters, see *Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836.

Example 5-2 dynacache.xml and servletcache.xml files for the sample application

```
dynacache.xml:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cacheUnit SYSTEM "dynacache.dtd">

<cacheUnit>
  <cache size="8000" priority="1" />
</cacheUnit>


servletcache.xml:
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE servletCache SYSTEM "servletcache.dtd">

<servletCache>
  <servlet>
      <path uri="/estore/index.jsp" />
  </servlet>
  <servlet>
      <path uri="/estore/petfooter.jsp" />
  </servlet>
  <servlet>
      <path uri="/estore/topindex.jsp" />
  </servlet>
  <servlet>
      <path uri="/estore/signin.jsp" />
  </servlet>
  <servlet>
      <path uri="/estore/signinsuccess.jsp" />
  </servlet>

  <servlet>
      <path uri="/estore/signoff.jsp" />
  </servlet>

<servlet>
        <path uri="/estore/productcategory.jsp" />
   <request>
      <parameter id="category_id">
   </request>
  </servlet>

  <servlet>
        <path uri="/estore/product.jsp" />
   <request>
      <parameter id="product_id">
   </request>
  </servlet>

</servletCache>
```

To activate the dynamic caching, you have to create the files in Example 5-2 and set a couple of properties in the jvm.properties file for the corresponding J2EE server; see Example 5-3.

Example 5-3 dynacaching properties in the jvm.properties file.

```
com.ibm.ws390.wc.config.dynxmlfilename=/WebSphere390/CB390/
   controlinfo/envfile/OPPLEX/BBOHYBA1/dynacache.xml

com.ibm.ws390.wc.config.dynsrvxmlfilename=/WebSphere390/CB390/
   controlinfo/envfile/OPPLEX/BBOHYBA1/servletcache.xml

Note that the properties listed here are each contained on a single line in the
jvm.properties file.
```

For further information about how to activate and configure dynamic caching in WebSphere Application Server for z/OS and OS/390, see*Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836.

5.2.2 Configuration 1: Local IBM HTTP Server for static file handling

The first configuration we considered to manage the static contents of the sample application was to use the IBM HTTP Server for z/OS configured as a proxy (see Figure 5-3). With this infrastructure we gain the advantages of caching the static contents and avoiding sending unnecessary requests to the Web container, since these contents are managed directly by the IBM HTTP Server. We use the Fast Response Cache Accelerator (FRCA) and, as described in “Integrated scenario” on page 138, you can even disable the Web container static file serving capabilities.

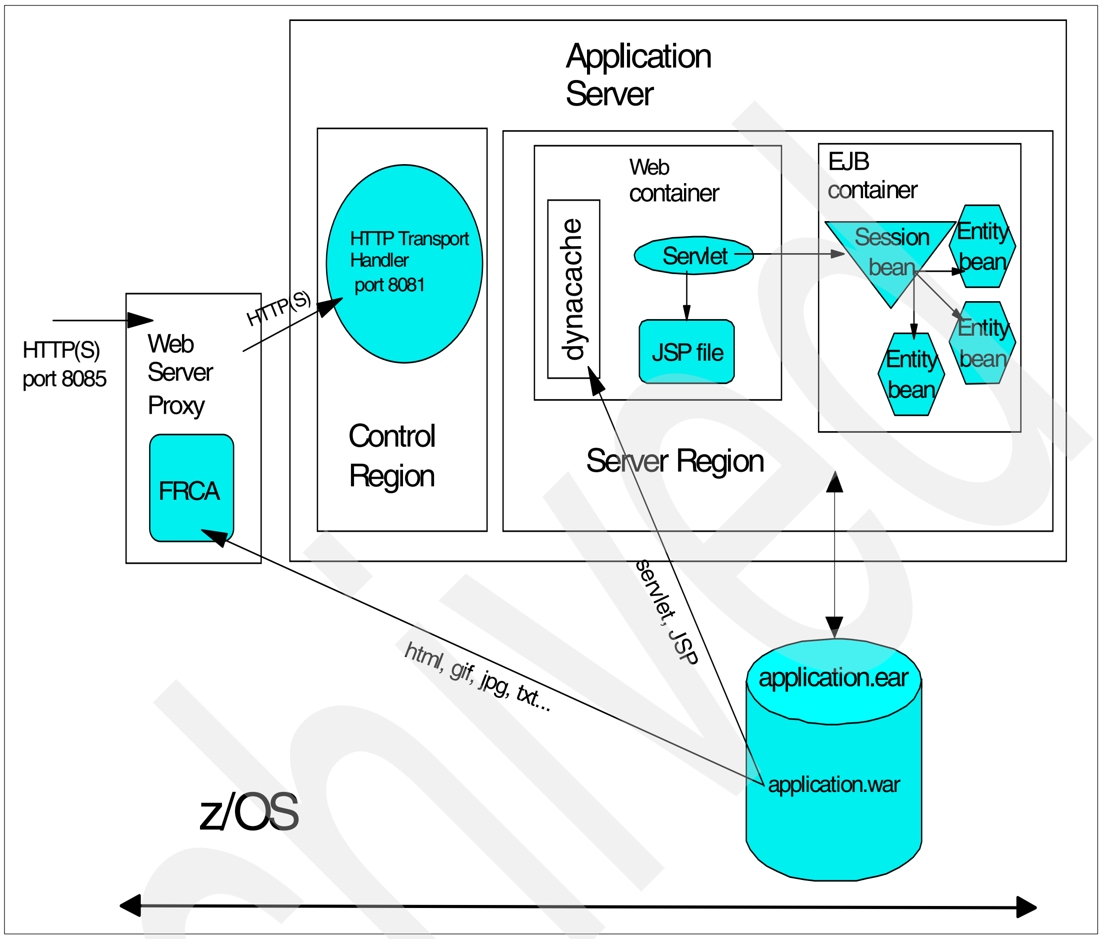


Figure 5-3 Configuration 1. Using the IBM HTTP Server for z/OS and OS/390 as a proxy

The directives used to get this behavior from the IBM HTTP Server for z/OS are shown in Example 5-4 on page 143.

Example 5-4 IBM HTTP Server httpd.conf file directives; caching proxy configuration

```
#        Basic directives
Port     8085

#        Pass rules
Pass    /estore/*.html        /WebSphere390/CB390/apps/BBOHYB/petstore/petstore.war/*
Pass    /estore/images/*.gif  /WebSphere390/CB390/apps/BBOHYB/petstore/petstore.war/*
Pass    /estore/images/*.jpg  /WebSphere390/CB390/apps/BBOHYB/petstore/petstore.war/*

#        Proxy directives
Proxy   /estore/*  http://wtsc59.itso.ibm.com:8081/estore/*

#        EnableFRCA directive
EnableFRCA on
```

To get better performance from the ensemble (IBM HTTP Server + WebSphere Application Server), we enabled the TCP/IP caching possibilities by setting the EnableFRCA to on. To get caching of the static elements, the Pass directives are necessary.

Of course, it is also recommended to review logging and caching policies. We activated and deactivated them as needed (for information about logging policies in the IBM HTTP Server for z/OS and OS/390, see *HTTP Server Planning, Installing and Using,* SC31-8690). You can get useful information about the caching behavior from those logs.

5.2.3 Configuration 2: Local IBM HTTP Server with WebSphere HTTP Plug-in

The IBM HTTP Server Plug-in is the component responsible for dispatching the requests to available J2EE servers. In our sandbox we used the WebSphere HTTP Plug-in for z/OS. To use this plug-in you have to apply PTF UQ74160 for APAR PQ68250, service level W401500. For a complete description and further documentation, see the documentation for the APAR.

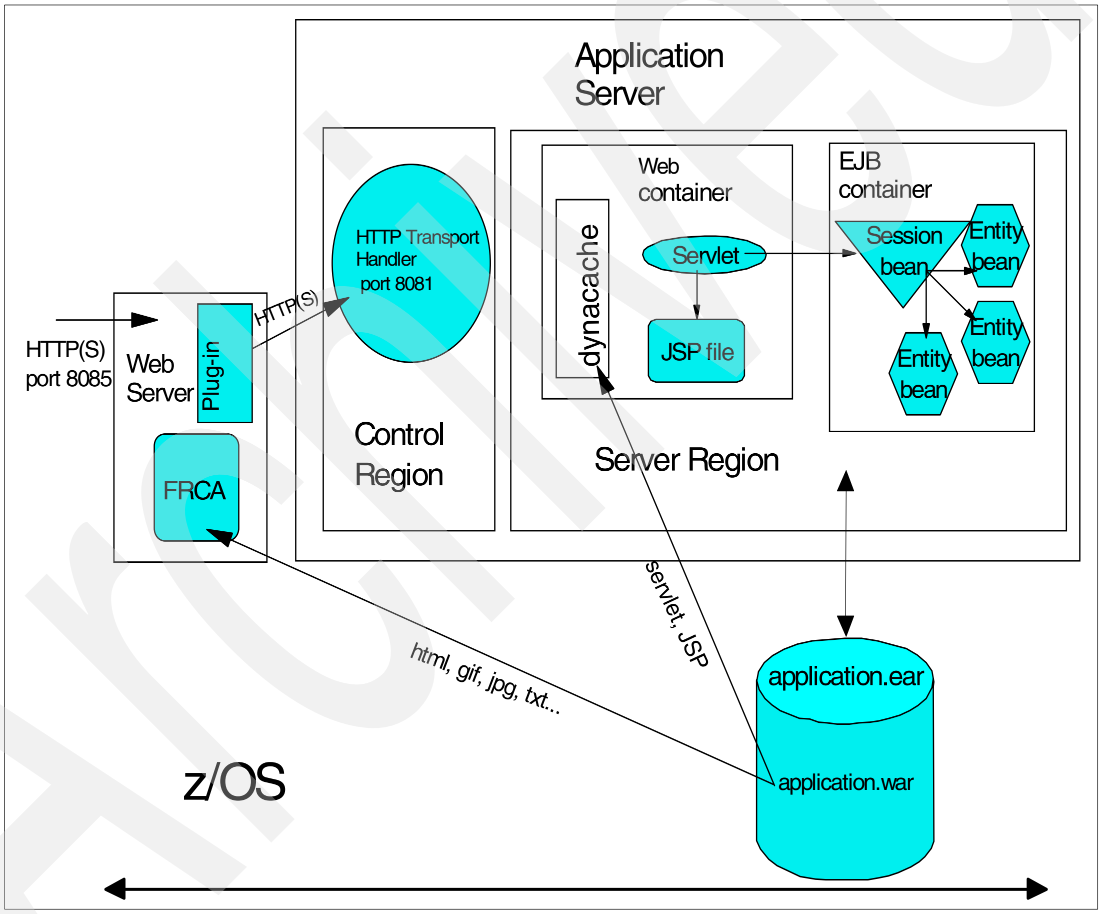


Figure 5-4 Configuration 2. Using the IBM HTTP Server for z/OS and OS/390 Plug-in

To realize the configuration of the WebSphere HTTP Plug-in for z/OS, there are some directives for using the plug-in (see Example 5-5).

Example 5-5 IBM HTTP Server httpd.conf directives

```
#    ===========================
#    *** WAS directives ***
#    ===========================
ServerInit /usr/lpp/WebSphere/WebServerPlugIn/bin/ihs390WASPlugin_http.so:init_exit
/web/hyb/plugin-cfg.xml
ServerTerm /usr/lpp/WebSphere/WebServerPlugIn/bin/ihs390WASPlugin_http.so:term_exit

service /estore/*
/usr/lpp/WebSphere/WebServerPlugIn/bin/ihs390WASPlugin_http.so:service_exit

   Note that the directives listed here are each contained on a single line in the
   httpd.conf file.
```

In addition to the directives needed to activate the plug-in, it must be configured to route requests to the correct J2EE servers where we want to dispatch the work. To do this we had to look for the plug-in.xml file template and make the necessary changes. This template should be in the /usr/lpp/WebSphere/WebServerPlugIn/bin directory (for further information, see APAR PQ68250).

Our configuration xml file is shown in Example 5-6.

Example 5-6 WebSphere HTTP Plug-in for z/OS configuration file

```
<Config>
   <!-- The LogLevel controls the amount of information that gets written to
        the plugin log file. Possible values are Error, Warn, and Trace. -->
   <Log Name="/web/hyb/logs/plugin.trace" LogLevel="Trace"/>

   <!-- Server groups provide a mechanism of grouping servers together. -->
   <ServerGroup Name="WTSC59">
      <Server CloneID="BBOHYBC.BBOHYBA1" Name="BBOHYBA1">
         <Transport Hostname="9.12.6.38" Port="8081" Protocol="http"/>
      </Server>
   </ServerGroup>

   <!-- Virtual host groups provide a mechanism of grouping virtual hosts together. -->
   <VirtualHostGroup Name="default_host">
      <VirtualHost Name="wtsc59.itso.ibm.com:8081"/>
       <VirtualHost Name="wtsc59:8081"/>
   </VirtualHostGroup>

   <!-- URI groups provide a mechanism of grouping URIs together. Only
        the context root of a web application needs to be specified unless
        you want to restrict the request URIs that get passed to the application
        server.  -->

   <UriGroup Name="default_host_URIs">

      <Uri Name="/estore/*"/>

   </UriGroup>

   <!-- A route ties together each of the above components. -->
   <Route ServerGroup="WTSC59" UriGroup="default_host_URIs"
VirtualHostGroup="default_host"/>
</Config>
```

In the <Server CloneID> statement we point to the physical server instances, where the application that we want to send requests to is contained.

The <Transport> statements contain the IP address and port of the J2EE server where the application we want to execute is. The port is the one specified by the environment variable BBOC_HTTP_PORT of the J2EE server, that is, the Transport Handler port.

The virtual host definitions set in the webcontainer.conf file of our J2EE server are also set in the <VirtualHost> statements.

The <UriGroup> identifies the part of the URI that will be routed through the plug-in to the application server. We just coded the one used for our test application, but others could be added.

And finally, the <Route> sentence associates the server instances in the ServerCloneID statements with the URIs reflected in the <UriGroup>, so that the requests are routed where we want them.

If you have been using the WebSphere HTTP Plug-in for distributed platforms and want to use it for z/OS, you can easily use the same xml configuration file. There are only minor changes from one file to another. See 5.2.5, “Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in” on page 149.

Example 5-7 Using FRCA with the WebSphere HTTP Plug-in

```
Pass  /estore/*.html  /WebSphere390/CB390/apps/BBOHYB/petstore/petstore.war/*
Pass  /estore/*.gif   /WebSphere390/CB390/apps/BBOHYB/petstore/petstore.war/*
Pass  /estore/*.jpg   /WebSphere390/CB390/apps/BBOHYB/petstore/petstore.war/*

ServerInit  /usr/lpp/WebSphere/WebServerPlugIn/bin/ihs390WASPlugin_http.so:init_exit
       /web/hyb/plugin-cfg.xml
ServerTerm  /usr/lpp/WebSphere/WebServerPlugIn/bin/ihs390WASPlugin_http.so:term_exit

service   /*  /usr/lpp/WebSphere/WebServerPlugIn/bin/ihs390WASPlugin_http.so:service_exit

EnableFRCA on


Note that the directives listed here are each contained on a single line in the httpd.conf
file.
```

There is a way to use the benefits of FRCA together with the WebSphere HTTP Plug-in on z/OS. To use it you have to set the EnableFRCA directive to “on” and add pass directives before the ServerInit and service rules (see Example 5-7).

5.2.4 Configuration 3: Remote reverse proxy caching server

With this configuration we wanted to get the same behavior and functionalities as with the IBM HTTP Server for z/OS. We still want to get all static content managed by the HTTP Server and relieve the application server of this task.

We can choose between a forwarding proxy and a reverse proxy. We get the most advantages, for our purpose, from the reverse proxy (for a discussion, see 4.5, “Configuration 3: Remote reverse proxy caching server” on page 122).

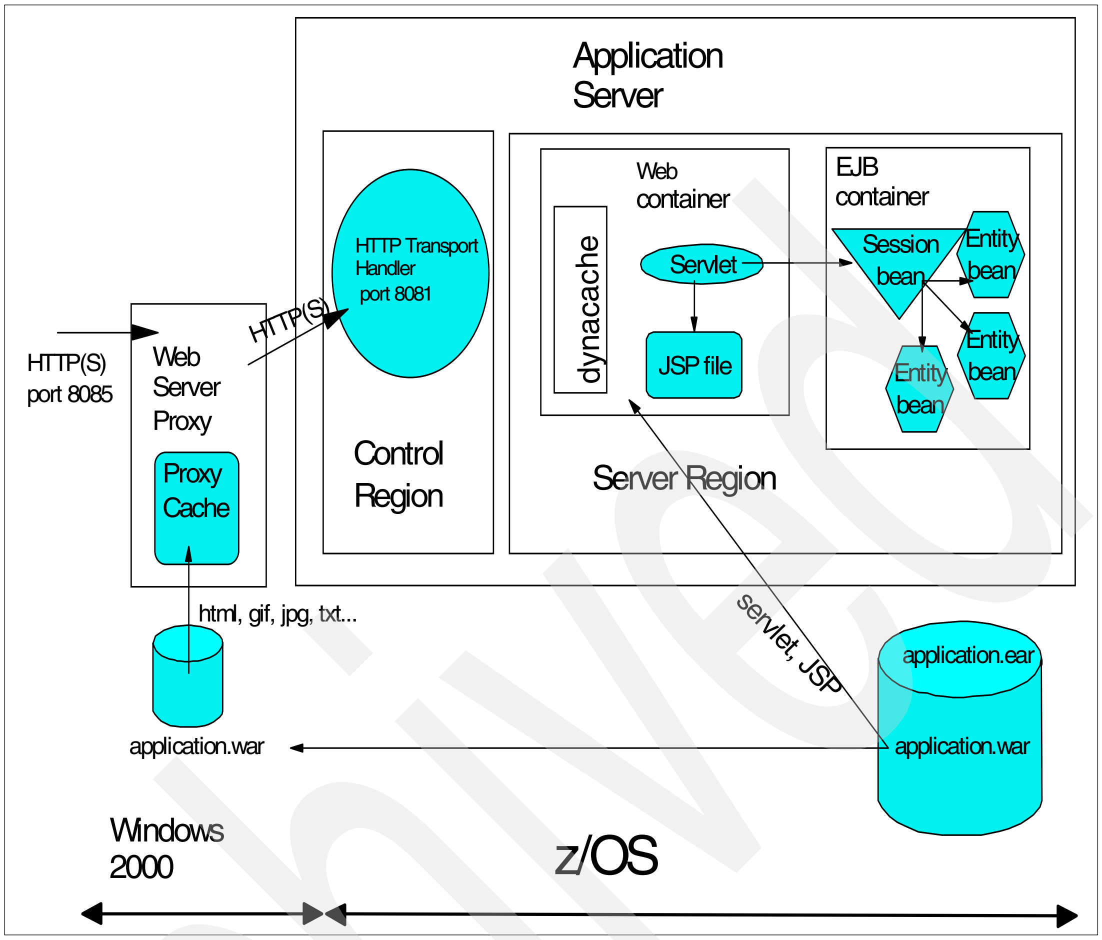


Figure 5-5 Configuration 3: IBM HTTP Server as a proxy on W2K platform and WebSphere Application Server for z/OS and OS/390

We used different versions of the IBM HTTP Server for Windows; for 1.3.x versions there are no differences in the configuration files when using the necessary directives to get proxy functions; for 2.0.x versions there are changes in the configuration directives for proxying and within the modules used for caching (see 4.5, “Configuration 3: Remote reverse proxy caching server” on page 122).

Note: The proxy configurations shown here are basic examples showing the functionality. You will need further directives to make your environment more secure and reliable. See the product documentation for further information.

For a direct proxy in 1.3.x versions, the httpd.conf file needs the directives shown in Example 5-8.

Example 5-8 httpd.conf directives to get a forwarding proxy in IBM HTTP Server 1.3.x versions

```
LoadModule proxy_module modules/ApacheModuleProxy.dll

AfpaEnable
AfpaCache on
AfpaLogFile "C:/IBM HTTP Server/logs/afpalog" V-ECLF

ProxyRequests On
```

```
<Directory proxy:*>
Order deny,allow
Allow from all
</Directory>

NOTE: Afpa directives are not required to get the proxy function, but don’t forget our aim
is to get the fastest way for static content to be served.
```

And to get a reverse proxy you just have to add ProxyPass directives with the URIs for your applications after the proxy definitions. Our case is shown in Example 5-9.

Example 5-9 ProxyPass for our example application

```
ProxyPass /estore/control/ http://wtsc59.itso.ibm.com:8081/estore/control/
```

For IBM HTTP Server, though the concept is the same, there are some changes for the proxy directives, as shown in Example 5-10.

Example 5-10 httpd.conf directives to get a forwarding proxy in IBM HTTP Server 2.0.x versions

```
LoadModule proxy_module modules/mod_proxy.so
LoadModule ibm_afpa_module modules/mod_afpa_cache.so

<IfModule mod_afpa_cache.c>
AfpaEnable
AfpaCache on
AfpaPort  80
AfpaLogFile "C:/Program Files/IBM HTTP Server 2.0/logs/afpalog" V-ECLF
</IfModule>

<IfModule mod_proxy.c>
ProxyRequests On

<Proxy *>
    Order deny,allow
    Allow from all
</Proxy>
</IfModule>
```

In the documentation for IBM HTTP Server 2.0.x, it is stated that the caching functions have been split and are not present in mod_proxy. These functions were set in mod_cache. This means that we have to pay special attention to configure caching because the IBM HTTP Server 2.0.x proxy doesn’t do that job.

To get a reverse proxy, you just have to add ProxyPass directives for your applications, as shown in Example 5-9.

To avoid sending static requests to the Web container, we extracted the .war file from the .ear file and copied it into the htdocs directory for the HTTP Server so that all content was locally available (see Figure 5-5 on page 147).

Tip: We also deactivated static content management in the Web container, as explained in “Integrated scenario” on page 138.

Another solution is to share all static content from the system where it is by using NFS or any other kind of network sharing.

5.2.5 Configuration 4: Remote IBM HTTP Server with WebSphere HTTP Plug-in

Support for Web server plug-ins was introduced in WebSphere Application Server for z/OS and OS/390 with Fix level 4 (W401400). With this configuration we used the WebSphere HTTP Plug-in on distributed platforms to dispatch requests to WebSphere Application Server for z/OS and OS/390. In Figure 5-6 we show how it was set up.

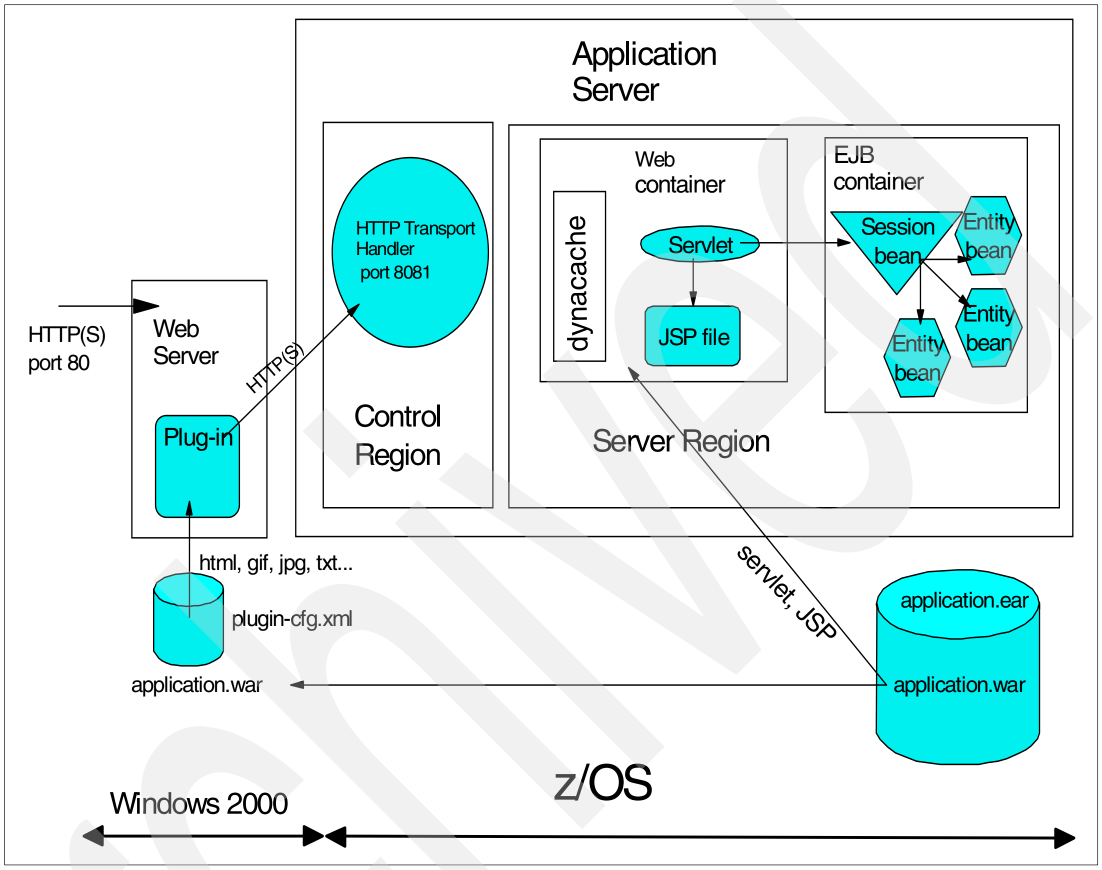


Figure 5-6 Configuration 4: IBM HTTP Server as with WebSphere HTTP Plug-in on W2K platform and WebSphere Application Server for z/OS and OS/390

The configuration file, httpd.conf, was modified as shown in Example 5-11.

Example 5-11 IBM HTTP Server httpd.conf file directives. Plug-in configuration.

```
ServerName tot134.itso.ibm.com

LoadModule expires_module modules/ApacheModuleExpires.dll
LoadModule ibm_app_server_http_module
C:/WebSphere/AppServer/bin/mod_ibm_app_server_http.dll

Port 80

WebSpherePluginConfig C:\WebSphere\AppServer\config\plugin-cfg.xml
```

With these directives, the plug-in is initialized when the Web server starts. We also have to configure the plug-in; for this task we used the same plugin-cfg.xml file shown in Example 5-6 on page 145, modified to specify a new virtual host for the IBM HTTP Server for Windows. This virtual host must be defined in the webcontainer.conf file of the corresponding J2EE server, shown in Example 5-12.

Example 5-12 WebSphere HTTP Plug-in for the Windows configuration file

```
<Config>
   <!-- The LogLevel controls the amount of information that gets written to
        the plugin log file. Possible values are Error, Warn, and Trace. -->
   <Log Name="c:\WebSphere\AppServer\logs\plugin.trace" LogLevel="Trace"/>

   <!-- Server groups provide a mechanism of grouping servers together. -->
   <ServerGroup Name="WTSC59">
      <Server CloneID="BBOHYBC.BBOHYBA1" Name="BBOHYBA1">
         <Transport Hostname="9.12.6.38" Port="8081" Protocol="http"/>
      </Server>
   </ServerGroup>

   <!-- Virtual host groups provide a mechanism of grouping virtual hosts together. -->
   <VirtualHostGroup Name="default_host">
      <VirtualHost Name="tot134.itso.ibm.com:80"/>
   </VirtualHostGroup>

   <!-- URI groups provide a mechanism of grouping URIs together. Only
        the context root of a web application needs to be specified unless
        you want to restrict the request URIs that get passed to the application
        server.  -->

   <UriGroup Name="default_host_URIs">

      <Uri Name="/estore/control/*"/>

   </UriGroup>

   <!-- A route ties together each of the above components. -->
   <Route ServerGroup="WTSC59" UriGroup="default_host_URIs"
VirtualHostGroup="default_host"/>
</Config>
```

As you can see, the only difference between these files is the <VirtualHost> sentence.

In this case, we cannot get the benefits of caching static content, but there are some ways for it to be offloaded. From a general point of view, you need to remove from the plugin-cfg.xml file all URIs you don't want to be forwarded to the application server. This can be a delicate task because sometimes URIs containing static content cannot be properly separated from those that invoke dynamic activity.

In our sample application, all images are invoked under /estore/image/imageName.gif. Taking this into account, the proper policy would be to code all other URIs in the configuration file, p.e. /estore/control/* so that every request matching that pattern will be sent to the application server, and any other request, like /estore/images/*, will be discarded by the plug-in and must be managed by the Web server. In this example, we learn that a clear relative path policy may make it easier setting up the environment when we have both local and remote requests in the same application.

This structure requires all static content to be made available to the IBM HTTP Server by copying the application .war structure into the Web server box or by sharing it as a Network File System (NFS).

You can even deactivate static content management in the Web container as described in “Integrated scenario” on page 138.

6

Implementing IIOP-based cross-platform scenarios

This chapter describes how to split one J2EE application into two independent components or artifacts: one that contains only EJB modules, and one that contains only Web modules—and how to deploy them to the different platforms.

We use the IBM version (not Sun's original version) of Java Pet Store Demo Version 1.1.2 as an example, which is supplied as sample with WebSphere Application Server Advanced Edition. For more details, refer to the following URL:

```
http://www-3.ibm.com/software/webservers/appserv/zos_os390/doc/v401/pstore/petstore.html
```

For development, we use WebSphere Studio Application Developer V4 (WSAD). Initially, there is one J2EE application project, Java Pet Store Demo. There are also seven EJB projects and one Web project in WSAD workspace. All of these EJB and Web modules are included in Java Pet Store Demo J2EE application project (Table 6-1lists each EJB setting on WSAD).

Table 6-1 Each EJB Web module setting

EJB/Web	Displayname	module	jar/war file name
Account	TheAccount	Customer Component	customerEjb.jar
Customer	TheCustomer		
Order	TheOrder		
Inventory	TheInventory	Inventory Component	inventoryEjb.jar
Mailer	TheMailer	Mail Component	mailerEjb.jar
ProfileMgr	TheProfileMgr	Personalization Component	personalizationEjb.jar
ShoppingClientController	TheShoppingClientController	Petstore EJB Component	petstoreEjb.jar
Catalog	TheCatalog	ShoppingCart Component	shoppingcartEjb.jar

EJB/Web	Displayname	module	jar/war file name
ShoppingCart	TheCart	SignOn Component	signonEjb.jar
SignOn	TheSignOn		
petstore	WebTier	WebTier	petstore.war

We use WebSphere Application Server V4.01 for z/OS and OS/390 for the EJB-tier, and WebSphere Application Server Advanced Edition (WebSphere AE) for the Web-tier. We use DB2 Universal Database for OS/390 and z/OS (DB2 for OS/390 and z/OS) as DBMS.

6.1 Application development and deployment

In this section, we describe how to split one J2EE application into two independent components. We use Java Pet Store Demo Version 1.1.2, but the contents described here would be applicable to other real applications.

If you want to follow the steps we performed with our sample application, then also follow the steps descibed in 6.2, "Importing the Java Pet Store Demo application into WebSphere Studio Application Developer V4" on page 167.

6.1.1 Analyzing the application

In the J2EE design, applications are split on the IIOP boundary between Web-tier and EJB-tier. Therefore, it should be easy to split one J2EE application into two independent parts: one that contains only EJB modules, and one that contains only Web modules.

To do this, however, there are some considerations, as follows:

- Accessing EIS from Web modules
- Accessing EJB from Web modules
- Calling EJB modules' classes in Web modules
- Calling Web modules' classes in EJB modules

Each balloon in Figure 6-1 on page 155 shows the point to consider in splitting a J2EE application.

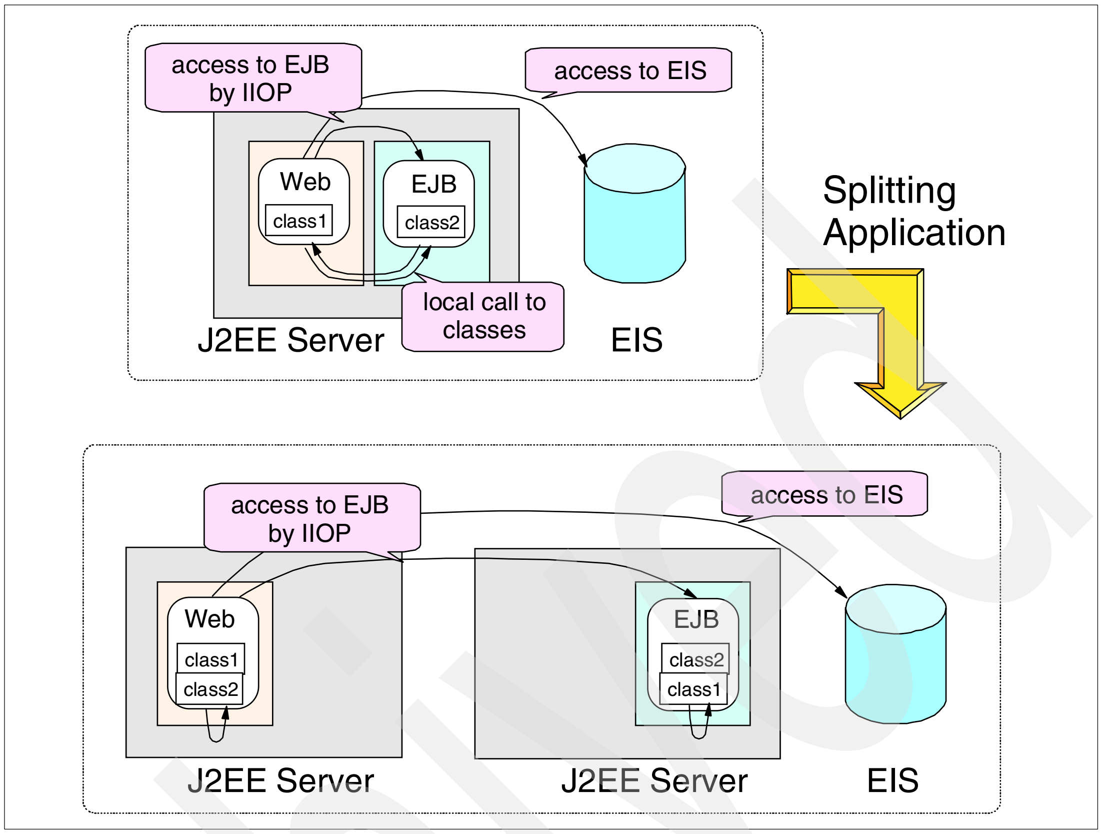


Figure 6-1 Possible split points for a J2EE application

Access to EIS from Web modules

Java Pet Store Demo Version 1.1.2 does not strictly follow the MVC design pattern because it incorporates database access into a servlet. Therefore, you need to set up the application environment in order to access the database from the J2EE server where the Web module runs. In the case of DB2 for OS/390 and z/OS, you need to set up Distributed Relational Data Architecture (DRDA) to connect to DB2 for OS/390 and z/OS from a Web module, and to define to the J2EE resource on the J2EE server where the Web module runs.

The following authorities for the J2EE resource definition are required:

- Authority to perform SELECT from SYSIBM.SYSTABLES in order to confirm whether all required tables are complete in the Populate servlet.
- Authority to perform SELECT from two application tables, PRODUCT and ITEM.

The following SQL is an example used in Java Pet Store Demo:

```
SELECT PRODUCTID, NAME, DESCN FROM PRODUCT WHERE CATEGORY = 'FISH' ORDER BY NAME FOR
FETCH ONLY
```

Since the High Level Qualifier (HLQ) is not specified in the program of Java Pet Store Demo, you must specify the HLQ of the tables used in Java Pet Store Demo. You can specify the HLQ in Client Configuration Assistant of DB2 Universal Database for Windows by taking the steps shown in “Setting up the DRDA environment on Windows” on page 163.

Of course you can modify the source code of Java Pet Store Demo in order to add the HLQ to the table name. In this case, you should modify only one method, `getTableName()` in `com/sun/j2ee/blueprints/shoppingcart/util/DatabaseNames.java` class. Just add your HLQ before the returned variable.

Specify the user ID and password that are used by the database connection. You can choose between three methods to specify them. We chose to specify them with DB2 connect in this chapter. For more details about the methods, refer to 3.2.3, “Security” on page 78.

See 6.1.3, “Deploying the application into multiple tiers” on page 159 about setting DRDA.

Accessing EJB from Web modules

You need to consider the following:

- In order to get the Interoperable Object Reference (IOR) of the EJBs’ home objects, the EJB clients (that is, Web modules) need to get access to the JNDI naming service. Because EJB modules and Web modules are separated physically, Web modules must get access to the remote naming service in WebSphere Application Server for z/OS and OS/390. Therefore, it is necessary to explicitly set property values and pass them to the InitialContext constructor in order to get access to the remote naming service. Usually you let these default—you only need them for remote naming service access:

  ```
  javax.naming.Context.INITIAL_CONTEXT_FACTORY
  javax.naming.Context.PROVIDER_URL
  ```

- In order to get access to the remote methods of EJBs, Web modules make use of the EJBs’ stub classes included in the EJB .jar. When the Web container is in a separate application server from the EJB containers, Web modules cannot use these stub classes in the EJB .jar file. Therefore, you have to add these stub classes to CLASSPATH of the Web container, or include them in the Web modules.

  For packaging classes, see “Calling EJB modules’ classes in Web modules” on page 158.

In the Pet Store application, modify the following two parts:

- In order to get access to the remote naming service of WebSphere Application Server for z/OS and OS/390, you need to modify the source code to pass the remote naming server's URL as an argument to the constructor of the InitialContext, although originally the constructor doesn't have any arguments.

  Look-up for EJB by servlet is performed by only one class, `com/sun/j2ee/blueprints/petstore/util/EJBUtil.java`. You need to modify the source code as shown in Example 6-1.

Example 6-1 Addition of properties as argument for the constructor of InitialContext

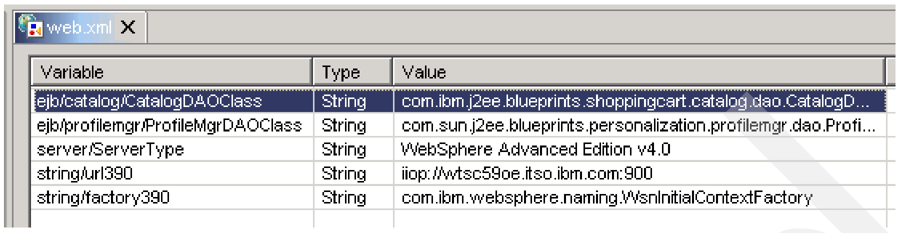
```
String provider_url = "iiop://wtsc59oe.itso.ibm.com:900" ;
String factory_class = "com.ibm.websphere.naming.WsnInitialContextFactory" ;
Properties prop = new Properties() ;
prop.put(javax.naming.Context.PROVIDER_URL, provider_url) ;
prop.put(javax.naming.Context.INITIAL_CONTEXT_FACTORY, factory_class) ;
InitialContext initial = new InitialContext(prop) ;
```

Of course, hard-coding of specific host names reduces the portability of the application. It's better to get these strings from outside of the application with an environment naming entry as shown in Example 6-2.

Example 6-2 Use of an environmental context

```
InitialContext _init = new InitialContext() ;
String provider_url = (String)_init.lookup("java:comp/env/string/url390") ;
String factory_class = (String)_init.lookup("java:comp/env/string/factory390") ;
```

In this case, you need to set the values *string/url390* and *string/factory390* in the web.xml editor at J2EE Perspective in WSAD as shown in Figure 6-2.



Variable	Type	Value
ejb/catalog/CatalogDAOClass	String	com.ibm.j2ee.blueprints.shoppingcart.catalog.dao.CatalogD...
ejb/profilemgr/ProfileMgrDAOClass	String	com.sun.j2ee.blueprints.personalization.profilemgr.dao.Profi...
server/ServerType	String	WebSphere Advanced Edition v4.0
string/url390	String	iiop://wtsc59oe.itso.ibm.com:900
string/factory390	String	com.ibm.websphere.naming.WsnInitialContextFactory

Figure 6-2 The web.xml editor

Or simply add the entries to web.xml as shown in Example 6-3.

Example 6-3 Addition of new entries to web.xml

```
<env-entry>
  <env-entry-name>string/url390</env-entry-name>
  <env-entry-value>iiop://wtsc59oe.itso.ibm.com:900</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
<env-entry>
  <env-entry-name>string/factory390</env-entry-name>
  <env-entry-value>com.ibm.websphere.naming.WsnInitialContextFactory</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

- As described in the J2EE specification, it is recommended to use the `java:comp/env` context for JNDI lookup. However, if you use WebSphere AE Version 4.0.1, you may need to modify the source code so that the Web module doesn't use the `java:comp/env` context because WebSphere AE Version 4.0.1 cannot process this context for remote JNDI service of WebSphere Application Server V4.01 for z/OS and OS/390.

  In this case, the class you need to modify is:
  `com/sun/j2ee/blueprints/petstore/util/EJBUtil.java`

  If the JNDI name specified in SMEUI may change, you should specify from outside the application which environment naming context for the JNDI name should be used by JNDI lookup. For example, we modified EJBUtil.java as follows:

Example 6-4 Modification of EJBUtil.java

```
//Object objref = init.lookup(JNDINames.CUSTOMER_EJBHOME); // original code
String homename = (String)_init.lookup("java:comp/env/string/Customer") ;
Object objref = init.lookup(homename) ;
```

We then added the following entry to web.xml.

Example 6-5 Addition of web.xml

```
<env-entry>
   <env-entry-name>string/Customer</env-entry-name>
   <env-entry-value>OPPLEX/BBOHYB/petstore/Customer
   Component/TheCustomer/CustomerHome</env-entry-value>
   <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Calling EJB modules' classes in Web modules

Because EJB modules and Web modules are split, Web modules cannot make use of the classes in EJB .jars. For example, Web modules use EJBs' stub classes in order to call remote methods of EJBs, so you need to pack these stub classes into Web modules since they are only included in EJB jars.

In the IBM version of Java Pet Store Demo, the following class in the shoppingcartEjb EJB .jar file is also used by Web modules:

```
com/ibm/j2ee/blueprints/shoppingcart/catalog/dao/CatalogDAOIDBImpl.java
```

These classes, stub classes, and the CatalogDAOIDBImpl.java class, are contained in each EJB .jar file. Therefore, the easiest way to include these classes in Web modules is to use the EJB .jar file without change. The following procedure shows how to include EJB .jar files in Web modules with WSAD:

1. If there are EJB jar files in the `petstore/webApplication/WEB-INF/lib` folder on Web Perspective, delete them by right-clicking them and selecting **Delete** from the pop-up menu.
2. Move to **J2EE Perspective** and open **J2EE View**.
3. Right-click **Petstore EJB Component** in the EJB Modules folder, and select **Export EJB Jar...** from pop-up menu.
4. Confirm that petstoreEjb is chosen in the What resources... selection box.
5. Specify the directory and filename to export, and press **Finish**. The petstoreEjb.jar file is exported to the directory.
6. Export all other EJB components in the same way.
7. Move to **Web Perspective** and open **Navigator**.
8. Click the `webApplication/WEB-INF/lib` folder in the petstore project, and select **File** -> **Import**.
9. Select **File system** and press **Next**.
10. Specify the same directory name as in 5.
11. Select all EJB .jar files displayed in the right pane.
12. Confirm that the `petstore/webApplication/WEB-INF/lib` folder is set in the Folder input form and press **Next**.

Calling Web modules' classes in EJB modules

In almost all cases, EJB modules don't use any classes of Web modules.

In Java Pet Store Demo, there is no such case.

6.1.2 Assembling the application

Now we are ready to split Java Pet Store Demo into two independent components capable of being deployed in separate containers. First, the following procedure shows the way to create a J2EE project that only contains EJB modules:

1. Move to **J2EE Perspective**.
2. Right-click the **Enterprise Application** folder in J2EE View, and select **New** -> **Enterprise application project**.
3. Type `petstoreEJBonly` in the Enterprise application project name form.

4. Uncheck all check boxes of Application Client project name, EJB project name, and Web project name.
5. Press **Finish**.
6. Confirm that the petstoreEJBonly project is added in the Enterprise Application folder.
7. Right-click the **petstoreEJBonly** project, and Select **Open With** -> **Application Editor**. Then application.xml is opened.
8. Click the **Modules** tab in the application.xml editor.
9. Press **Add**, select **customerEjb,** and press **OK**.
10. Do the same for inventoryEjb, mailerEjb, personalizationEjb, petstoreEjb, shoppingcartEjb, and signonEjb.
11. Press Ctrl+S to save it.
12. Right-click the **petstoreEJBonly** project in the Enterprise applications folder, and select **Export EAR File...**
13. Specify the directory and file name, for example, *c:\temp\petstoreEJBonly.ear*.
14. Press **Finish**.

The procedure that creates a J2EE project which only contains Web modules is as follows:

1. Move to **J2EE Perspective**.
2. Right-click the **Enterprise Application** folder in J2EE View, and select **New** -> **Enterprise Application Project**.
3. Type `petstoreWebonly` in the **Enterprise application project name** form.
4. Uncheck all check boxes of Application Client project name, EJB project name, and Web project name.
5. Press **Finish**.
6. Confirm that the petstoreWebonly project is added in the Enterprise Application folder.
7. Right-click the **petstoreWebonly** project, and select **Open With** -> **Application Editor**. Application.xml is opened.
8. Click the **Modules** tab in the application.xml editor.
9. Press **Add**, select **petstore** and press **OK**.
10. Press Ctrl+S to save it.
11. -Right-click the **petstoreWebonly** project in the Enterprise applications folder, and select **Export EAR File...**
12. Specify the directory and the file name which you want to name *petstoreWebonly* Enterprise Application Archive file, for example, c:\temp\petstoreWebonly.ear.
13. Press **Finish**.

6.1.3 Deploying the application into multiple tiers

The outline for deploying Java Pet Store Demo into multiple tiers is as follows:

- Deploy the petstoreEJBonly .ear file exported previously into WebSphere Application Server V4.01 for z/OS and OS/390.
- Set up the DRDA environment on Windows that WebSphere AE runs.
- Deploy the petstoreWebonly .ear file exported previously into WebSphere AE.

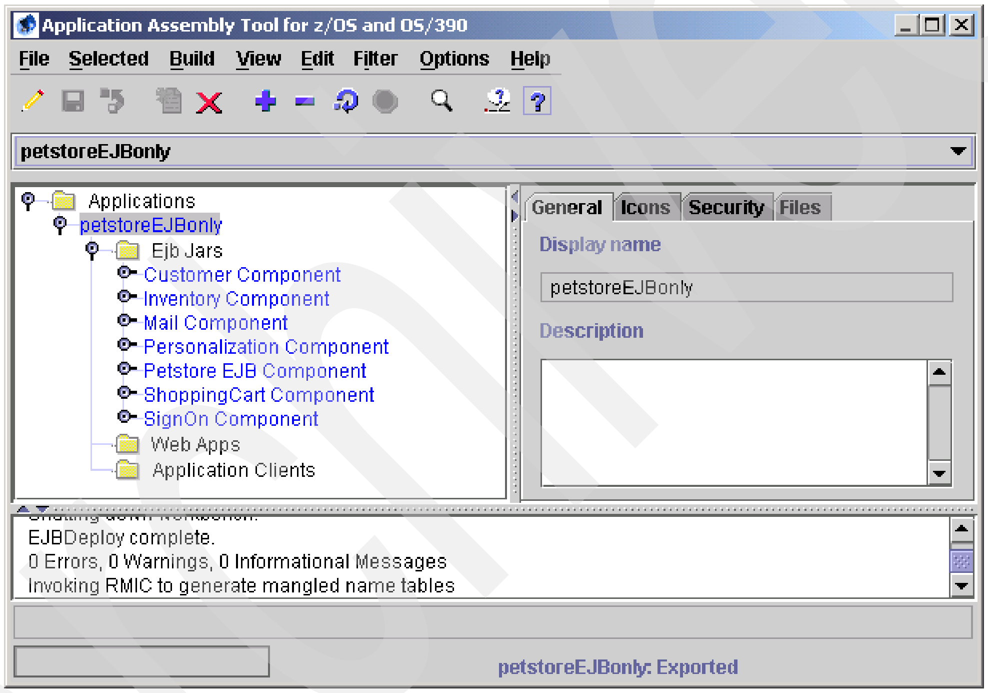
Deploying the petstoreEJBonly .ear file

The following procedure shows how to deploy the petstoreEJBonly.ear file that is exported in 6.1.2, "Assembling the application" on page 158 into WebSphere Application Server V4.01 for z/OS and OS/390.

First, you need to use Application Assembly Tool for z/OS and OS/390 (AAT).

Do the following:

1. Start Application Assembly Tool for z/OS and OS/390 (AAT).
2. Right-click the **Applications** folder and select **Import**.
3. Press **Choose**, and specify the directory and name of the petstoreEJBonly.ear file that you exported in 6.1.2, "Assembling the application" on page 158, for example, c:\temp\petstoreEJBonly.ear.
4. Press **OK** to import the petstoreEJBonly.ear file into AAT (see Figure 6-3).



```
EJBDeploy complete.
0 Errors, 0 Warnings, 0 Informational Messages
Invoking RMIC to generate mangled name tables
```

petstoreEJBonly: Exported

Figure 6-3 Inporting the petstoreEJBonly.ear file into AAT

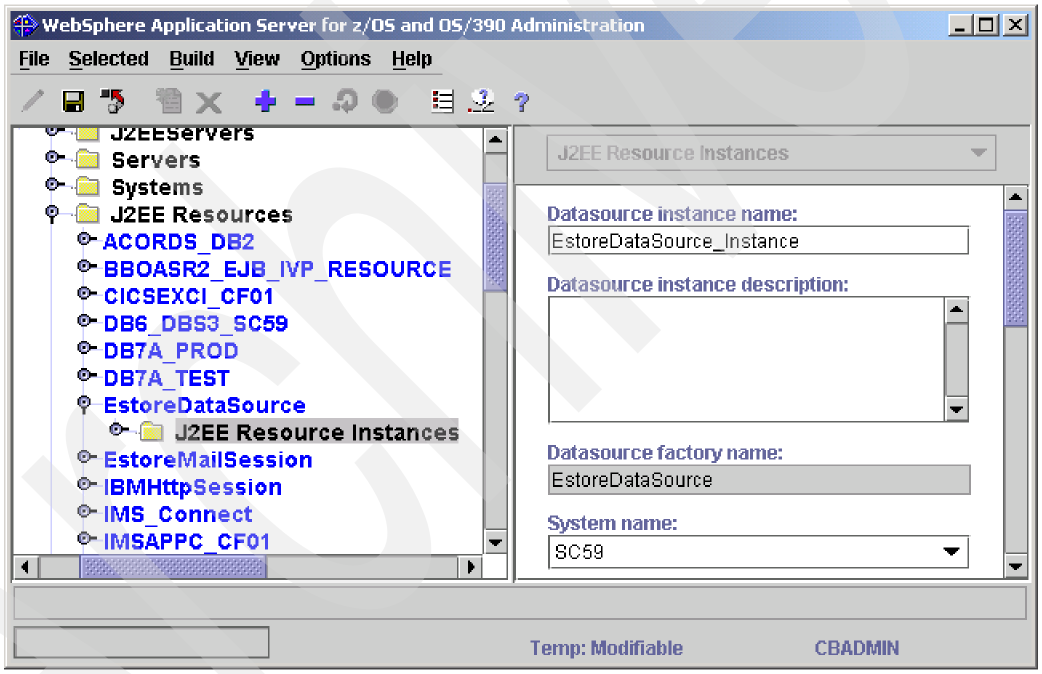
5. Right-click **petstoreEJBonly** in the Applications folder, and select **Validate**.
6. Right-click **petstoreEJBonly** in the Applications folder, and select **Deploy**.
7. Right-click **petstoreEJBonly** in the Applications folder, and select **Export**.
8. Specify the directory and file name, for example, *c:\temp\petstoreEJBonly.ear.*
9. Press **OK** to export it.

Next you need to use System Management End User Interface (SMEUI). As an example, a conversation name is set to *Installing petstoreEJBonly*, the server in which petstoreEJBonly.ear is installed is *BBOHYBC*, and the J2EE resource definition name on SMEUI is *EstoreDataSource here*. We don't introduce the procedure how to create the J2EE server here. Refer to *WebSphere Application Server for z/OS and OS/390 v4.0.1: Installation and Customization,* GA22-7834 for the procedure.

1. Start SMEUI on Windows.
2. Right-click the **Conversations** folder, and select **Add**.
3. Specify the conversation name, `Installing petstoreEJBonly`, and click the diskette icon in the upper left to save it.

The following procedure is for defining a J2EE data source in SMEUI:

1. Expand **Conversations** -> **Installing petstoreEJBonly** -> **Sysplexes** -> ***your sysplex name*** -> **J2EE Resources**.
2. Right-click the **J2EE Resources** folder, and select **Add**.
3. Type `EstoreDataSource` in the J2EE Resource name input form.
4. Select **DB2datasource** as J2EE Resource type.
5. Click the diskette icon in the upper left to save it.
6. Expand **EstoreDataSource**, right-click the **J2EE Resource Instances** folder under EstoreDataSource, and select **Add**.
7. Specify `EStoreDataSource_Instance` in the Datasource instance name input form.



Datasource instance name: EstoreDataSource_Instance

Datasource instance description:

Datasource factory name: EstoreDataSource

System name: SC59

Temp: Modifiable CBADMIN

Figure 6-4 Defining J2EE Resource Instances

8. Specify the location name of your DB2 for OS/390 and z/OS in the Location Name input form.

   Note: If you don't know the configuration of DB2 for OS/390 and z/OS, you can see it in the z/OS system log after DB2 initialization:

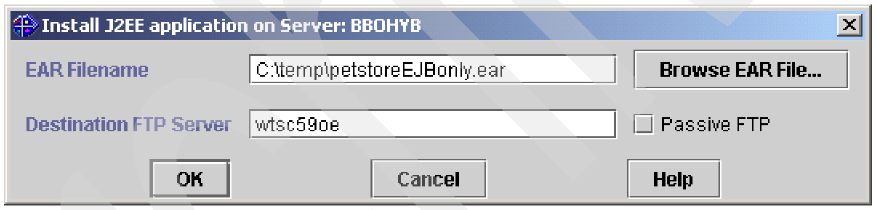
   ```
   DSNL004I  -DB7A DDF START COMPLETE 902
              LOCATION  DB7A
              LU        USIBMSC.SCPDB7A
              GENERICLU -NONE
              DOMAIN    wtsc59oe.itso.ibm.com
              TCPPORT   33700
              RESPORT   33701
   ```

   You can also query it by using the **./DIS DDF** command.

9. Specify the HLQ of the tables that Java Pet Store Demo uses in the DB2 SQLID input form.
10. Click the diskette icon in the upper left to save it.

Next, install the petstoreEJBonly .ear file with the following steps:

1. Expand **Conversations** -> **Installing petstoreEJBonly** -> **Sysplexes** -> ***your sysplex name*** -> **J2EEServers** -> **BBOHYB**.
2. Right-click **BBOHYB**, and select **Install J2EE Application**.
3. Specify the filename you exported from AAT, and press **OK** (see Figure 6-5).

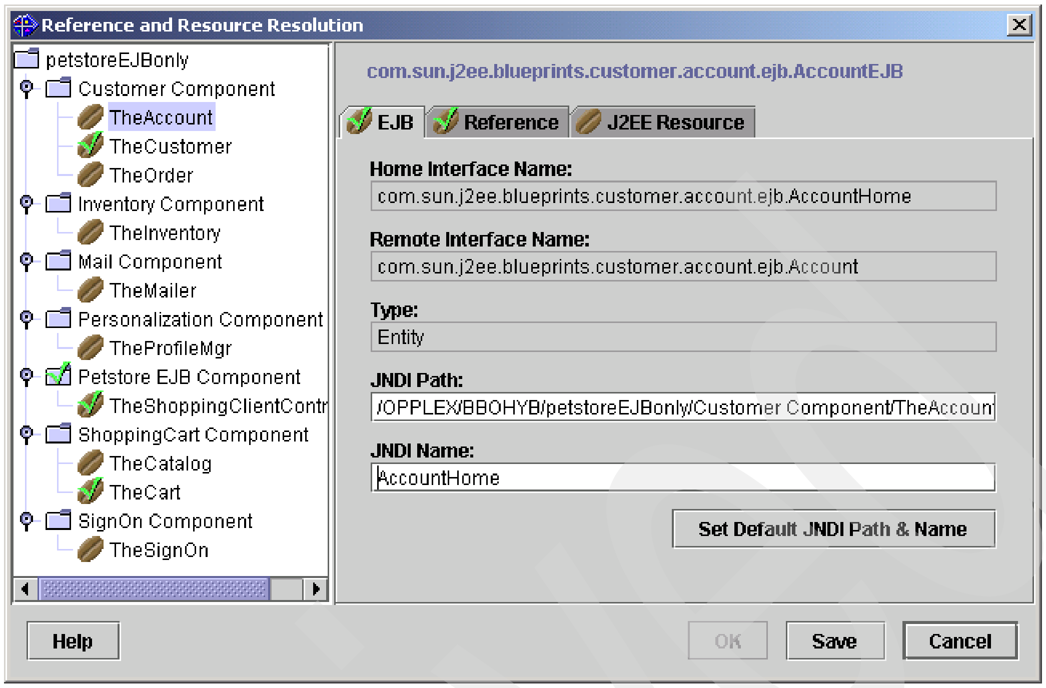

Install J2EE application on Server: BBOHYB

EAR Filename: C:\temp\petstoreEJBonly.ear

Destination FTP Server: wtsc59oe

Figure 6-5 Specifying petstoreEJBonly.ear file exported from AAT

4. Expand **petstoreEJBonly** -> **Customer Component**.
5. Select **TheAccount** under Customer Component, and select the **EJB** tab on the right pane.
6. Click **Set Default JNDI Path & Name**, and remove the package name from the JNDI Name input form (see Figure 6-6 on page 163).


Reference and Resource Resolution

com.sun.j2ee.blueprints.customer.account.ejb.AccountEJB

Home Interface Name: com.sun.j2ee.blueprints.customer.account.ejb.AccountHome

Remote Interface Name: com.sun.j2ee.blueprints.customer.account.ejb.Account

Type: Entity

JNDI Path: /OPPLEX/BBOHYB/petstoreEJBonly/Customer Component/TheAccoun

JNDI Name: AccountHome

Figure 6-6 Removing the package name from JNDI Name

7. Click the **J2EE Resource** tab.
8. Select **EstoreDataSource** added above.
9. Repeat steps 5-8 for all EJBs.
10. Press **OK**.
11. Right-click **Installing petstoreEJBonly** in the Conversations folder, and select **Validate**.
12. Right-click **Installing petstoreEJBonly** again, and select **Commit**.
13. Right-click **Installing petstoreEJBonly** again, and select **Complete -> All tasks**.
14. Right-click **Installing petstoreEJBonly** again, and select **Activate**.

Setting up the DRDA environment on Windows

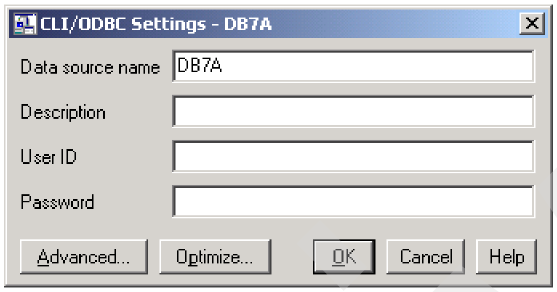
Following is the procedure to set up DRDA on Windows. We assume that DB2 Connect has been installed in Windows.

1. Click **Start** -> **Programs** -> **IBM DB2** -> **Client Configuration Assistant**.
2. Press **Add**. The Add Database Wizard window is displayed.
3. Choose **Manually configure a connection to a database** under the “1. Source” tab, and press **Next**.
4. Choose **TCP/IP** and check **The database physically resides on a host or AS/400® system** under the “2. Protocol” tab. Choose **Connect directly to the server** and press **Next**.
5. Fill in the host name and port number of your DB2 for OS/390 and z/OS under the “3. TCP/IP” tab, and press **Next**.
6. Fill in the Database name field in the “4. Database” tab and press **Next**.
7. Check **Register this database for ODBC** and click **As a system data source** in the “5. ODBC” tab, and press **Next**.

8. Click **Finish**.

Next, the procedure to specify the HLQ of tables that Java Pet Store Demo uses is as follows:

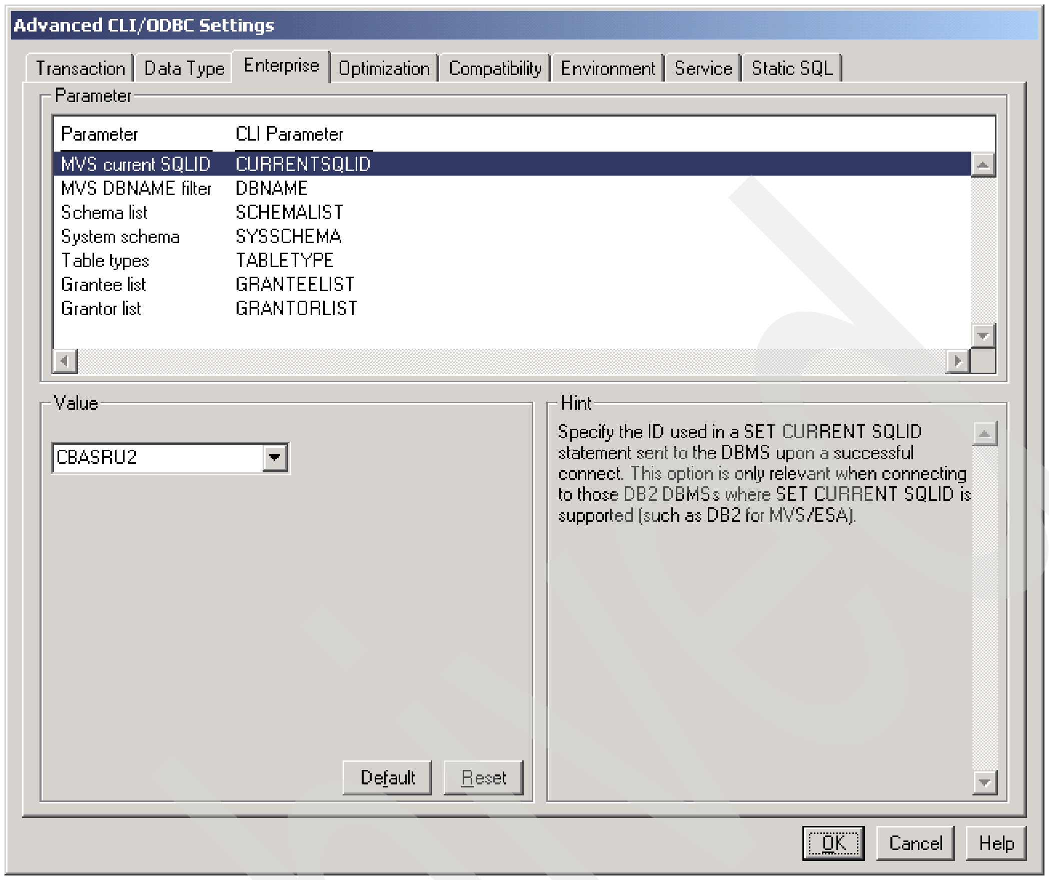
1. Start Client Configuration Assistant.
2. Select the database alias that is used by Java Pet Store Demo, and click **Properties**.
3. Press **Setting** on the Database Properties window. If the DB2 Message window appears, press **No**.
4. The CLI/ODBC Settings window appears (see Figure 6-7). If you want to specify the user ID and password that are used by datasource connection here, fill in User ID and Password.



Data source name	DB7A
Description	
User ID	
Password	

Figure 6-7 CLI/ODBC Settings

5. Press **Advanced**.
6. Select the **Enterprise** tab.
7. Specify the HLQ value in the Value input form (see Figure 6-8 on page 165).
8. Press **OK** and close the Client Configuration Assistant window.



Advanced CLI/ODBC Settings

Transaction | Data Type | Enterprise | Optimization | Compatibility | Environment | Service | Static SQL

Parameter

Parameter	CLI Parameter
MVS current SQLID	CURRENTSQLID
MVS DBNAME filter	DBNAME
Schema list	SCHEMALIST
System schema	SYSSCHEMA
Table types	TABLETYPE
Grantee list	GRANTEELIST
Grantor list	GRANTORLIST

Value: CBASRU2

Hint: Specify the ID used in a SET CURRENT SQLID statement sent to the DBMS upon a successful connect. This option is only relevant when connecting to those DB2 DBMSs where SET CURRENT SQLID is supported (such as DB2 for MVS/ESA).

Figure 6-8 Specifying the HLQ that Java Pet Store Demo uses

Deploying the petstoreWebonly .ear file

The following procedure shows how to deploy the petstoreWebonly .ear file which is exported in 6.1.2, “Assembling the application” on page 158 into WebSphere Application Server Advanced Edition V4 for Windows.

First, you need to define the datasource of , as follows:

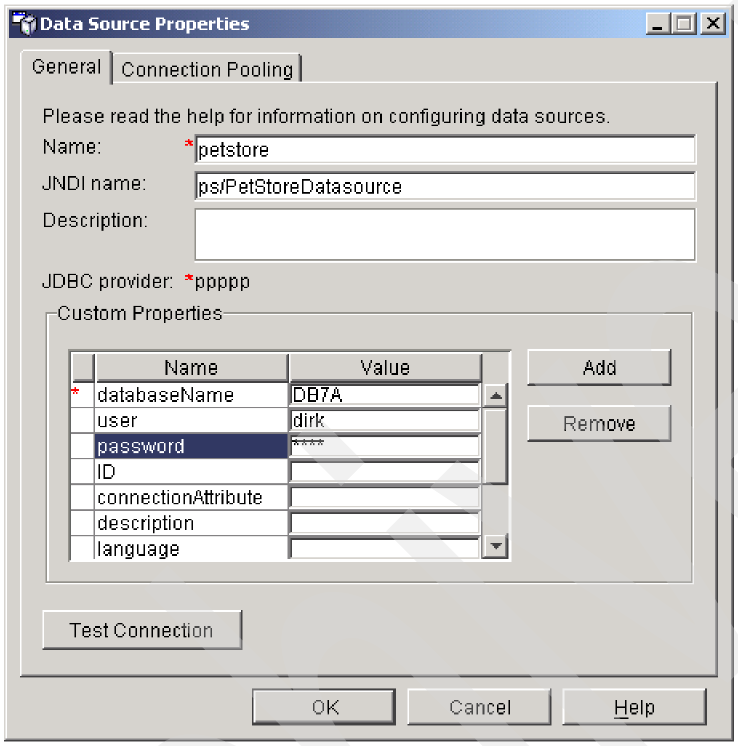
1. Start WebSphere Administrative Console on Windows.
2. Expand the **Resources** folder.
3. Right-click the **JDBC Providers** folder under the Resources folder.
4. Select **New**.
5. Fill in the Name field with the name; for example, `Petstore`.
6. Select **COM.ibm.db2.jdbc.DB2ConnectionPoolDataSource** as Implementation class.
7. Click the **Nodes** tab.
8. Click **Install New**. The Install Driver window is displayed.
9. Press **Specify Driver**. The Specify the Driver Files window is displayed.
10. Press **Add Driver**, and specify the DB2_INSTALL_PATH\SQLLIB\java\db2java.zip file.
11. Press **Set** to close the Specify the Driver Files window.
12. Press **Install** in the Install Driver window.
13. Press **OK** in JDBC Provider Properties.

14. Expand the Petstore under **JDBC Providers**.

15. Right-click the **Data Source** folder under Petstore, and select **New**.

16. Fill in Name, JNDI name, databaseName, user, and password (see Figure 6-9).

17. Press **OK**.



Please read the help for information on configuring data sources.

Name: * petstore

JNDI name: ps/PetStoreDatasource

Description:

JDBC provider: *ppppp

Custom Properties

	Name	Value
*	databaseName	DB7A
	user	dirk
	password	****
	ID	
	connectionAttribute	
	description	
	language	

Figure 6-9 Data Source Properties

Next, you need to deploy the petstoreWebonly .ear file into WebSphere Application Server Advanced Edition V4, as follows:

1. Expand the **Enterprise Applications** folder in WebSphere Administrative Console and right-click it.
2. Select **Install Enterprise Application**. The Install Enterprise Application Wizard window is displayed.
3. Choose **Install Application(*.ear)** and fill in the Path form with the path of c:\temp\*petstoreWebonly.ear* exported in 6.1.2, “Assembling the application” on page 158, and press **Next**.
4. Press **Next** four times to skip Mapping Users to Roles, Mapping EJB RunAs Roles to Users, Binding Enterprise Beans to JNDI Names, and Mapping EJB References to Enterprise Beans.
5. Click **jdbc/EstoreDataSource** -> **Select Resource** in Mapping Resource References to Resources. The Select Resource window is displayed.
6. Select **petstoreDatasource** and press **OK** -> **Next**.
7. Press **Next** three times to skip Specifying the Default Datasource for EJB Modules, Specifying Data Sources for Individual CMP Beans, and Selecting Virtual Hosts for Web Modules.
8. Press **Finish**.

6.2 Importing the Java Pet Store Demo application into WebSphere Studio Application Developer V4

We now introduce how to import Java Pet Store Demo Version 1.1.2 into WebSphere Studio Application Developer V4 (WSAD). We strongly recommend that you understand the license attached in Java Pet Store Demo Version 1.1.2 well and follow it, before any modification or redistribution of it.

6.2.1 Preparation of files

You need to use the IBM version of Java Pet Store Demo Version 1.1.2. The name of this file is petstore.ear, and can be acquired from the following directories:

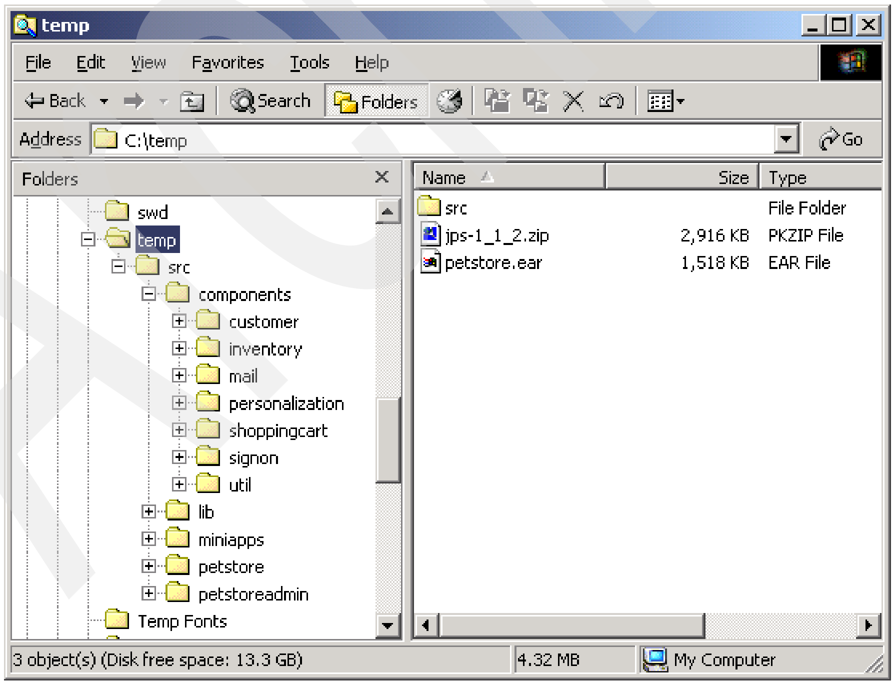
- The installableApps\petstore directory in WebSphere Application Server Advanced Edition
- The plugins\com.ibm.etools.websphere.runtime\installableApps\petstore directory in WebSphere Studio Application Developer V4

Copy the petstore.ear file to a working directory, for example, c:\temp.

This file doesn't include source code. Download the source code of Java Pet Store Demo from the following URL:

```
http://java.sun.com/blueprints/code/index.html
```

From this site, get the zip file jps-1_1_2.zip and extract the code. For this book, we assume that the files are put in the c:\temp directory, and that the jps-1_1_2.zip is extracted into c:\temp (see Figure 6-10).



```
swd
temp
  src
    components
      customer
      inventory
      mail
      personalization
      shoppingcart
      signon
      util
    lib
    miniapps
    petstore
    petstoreadmin
Temp Fonts
```

Name	Size	Type
src		File Folder
jps-1_1_2.zip	2,916 KB	PKZIP File
petstore.ear	1,518 KB	EAR File

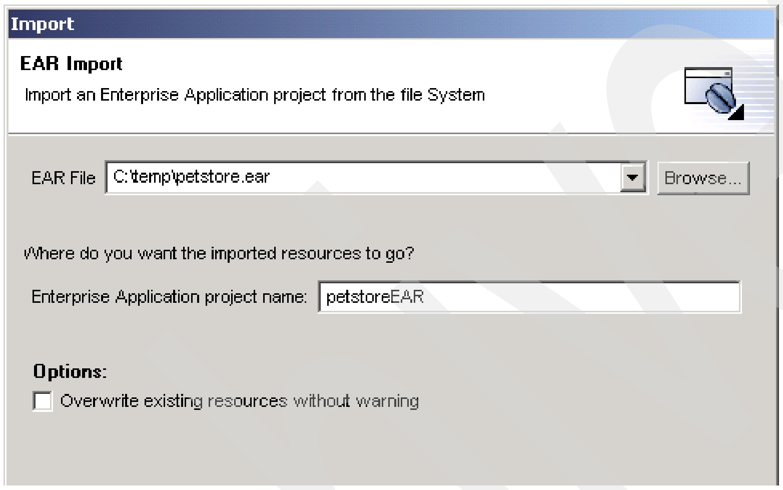
3 object(s) (Disk free space: 13.3 GB) 4.32 MB My Computer

Figure 6-10 Extracting jps-1_1_2.zip into the c:\temp directory

6.2.2 Importing petstore.ear into WSAD

The following procedure imports the petstore.ear file into WSAD:

1. Start WSAD.
2. Select **File** -> **Import**. The Import window appears.
3. Select **EAR file**, and press **Next**.
4. Type `c:\temp\petstore.ear` in the .ear file input form.
5. Type `petstoreEAR` in the Enterprises Application project name input form (see Figure 6-11).
6. Press **Finish**.



Import an Enterprise Application project from the file System

EAR File C:\temp\petstore.ear

Where do you want the imported resources to go?

Enterprise Application project name: petstoreEAR

☐ Overwrite existing resources without warning

Figure 6-11 Import window

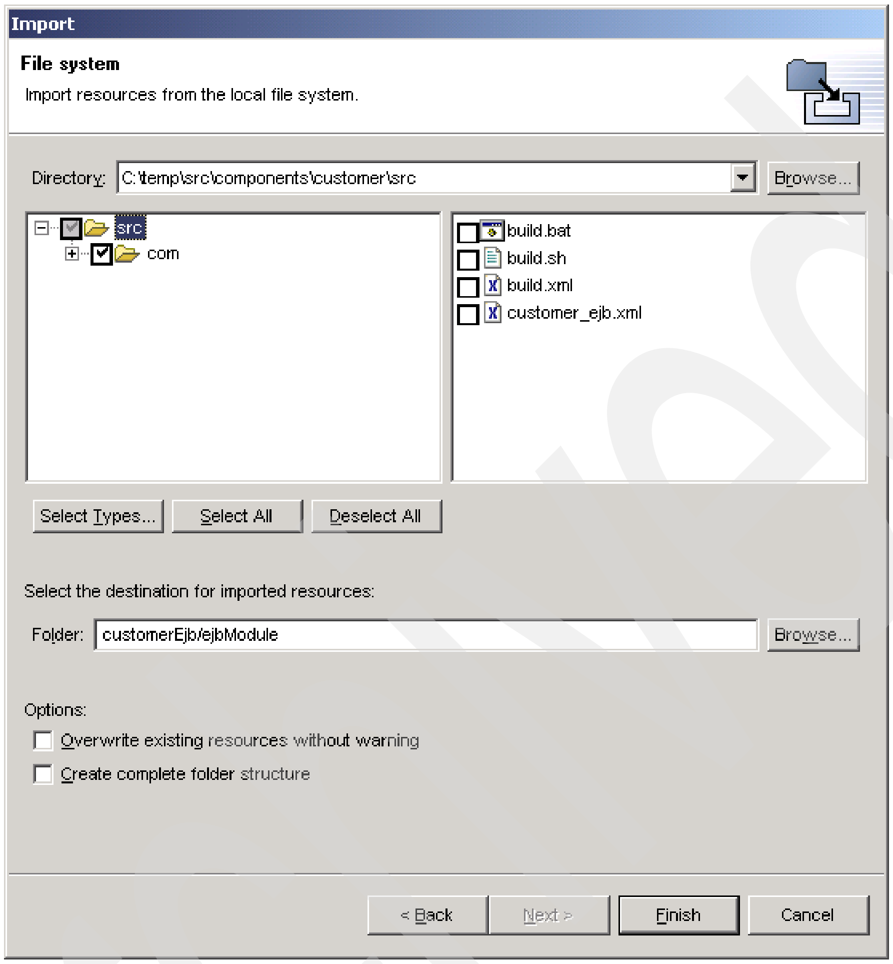
6.2.3 Importing source code into WSAD

The procedure to import the source code in jps-1_1_2.zip is as follows.

1. Move to the Web perspective.
2. Expand **customerEjb**.
3. Click the **ejbModule** folder under customerEjb, and select **File** -> **Import**. The Import window appears.
4. Select **File system**, and press **Next**.
5. Press **Browser** and specify the c:\temp\src\components\customer\src directory (see Figure 6-12 on page 169).
6. Check the com directory under the src directory.
7. Confirm that customerEjb/ejbModule is selected as Folder.
8. Press **Finish**.
9. Click the **ejbModule** folder under customerEjb, and select **File** -> **Import**. The Import window appears.
10. Select **File system**, and press **Next**.
11. Press **Browser** and specify the c:\temp\src\components\util\tracer\src directory.
12. Check the com directory under the src directory.

13.Confirm that customerEjb/ejbModule is selected as Folder.

14.Press **Finish**.



Directory: C:\temp\src\components\customer\src

- ☑ src
  - ☑ com

- ☐ build.bat
- ☐ build.sh
- ☐ build.xml
- ☐ customer_ejb.xml

Select the destination for imported resources:

Folder: customerEjb/ejbModule

Options:

- ☐ Overwrite existing resources without warning
- ☐ Create complete folder structure

Figure 6-12 Importing the source directory of Java Pet Store Demo

15.Repeat steps 2-14 for inventoryEjb, mailerEjb, personalizationEjb, shoppingcartEjb, signonEjb, and petstoreEjb. That is, you need to import the valid source code directory shown in Table 6-2, and the c:\temp\src\components\util\tracer\src directory into the ejbModule folder under each EJB project.

Table 6-2 .Petstore source code directories

EJB module	Directory including source code
customerEjb	*c:\temp\src\components\customer\src*
inventoryEjb	*c:\temp\src\components\inventory\src*
mailerEjb	*c:\temp\src\components\mail\src*
personalizationEjb	*c:\temp\src\components\personalization\src*
shoppingcartEjb	*c:\temp\src\components\shoppingcart\src*
signonEjb	*c:\temp\src\components\signon\src*
petstoreEjb	***c\temp\src\petstore\src***

16. For Web component, import the C:\temp\src\petstore\src and c:\temp\src\components\util\tracer\src directories into the petstore/source folder. It's not necessary to check docroot and the lib directory under the src directory.

17. (Optional) See 6.1.1, "Analyzing the application" on page 154 and 6.1.2, "Assembling the application" on page 158 for splitting into EJB modules and Web modules.

6.2.4 Testing the application in WebSphere Studio Application Developer V4

The Web module petstore.war, and the EJB module petstoreEjb.jar, contain the exact same classes as the procedure in 6.2, "Importing the Java Pet Store Demo application into WebSphere Studio Application Developer V4" on page 167. This fact may affect your testing in WSAD.

If you follow the above procedure and split Java Pet Store Demo into two EAR projects, you should have three EAR projects in WSAD, petstore, petstoreWebonly, and petstoreEJBonly. And you modify the source code of the Web module in order to access a remote (probably 390s) JNDI server. When you now start the petstore project, which includes all modules in the WSAD test environment, would this Web module be able to access the remote JNDI server?

Of course, if the class you modified is used by the WSAD test environment, the module will be able to access the remote server. However, the class in petstoreEjb.jar may be used instead of the class in petstore.war. In that case, the module won't be able to access the remote server. By default, the class loader visibility mode is *Application*, and modules are able to access any other classes in the same .ear file. Therefore, if the visibility mode is set to *Module*, the Web module will be able to access the remote JNDI server. However, in this case, each EJB module in petstore.ear won't work well because of the dependencies on each other.

The following figure shows this situation. Even if yo modify any classes in Web module, the classes may not be used in petstore.ear because the same classes are included in EJB module, petstoreEjb.jar, and it may be used instead of modified classes.

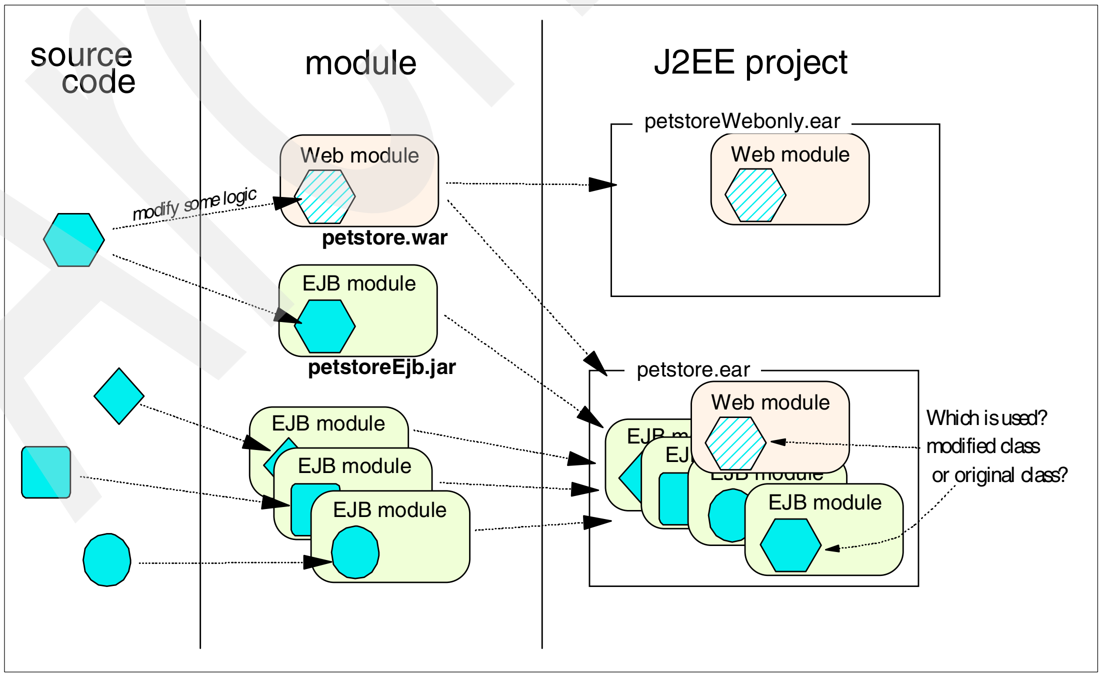


Figure 6-13 Java Pet Store Demo imported into WSAD

This fact suggests two important consideration points. One is that the same classes must not be included into two modules. If the same classes are included into two or more modules, the behavior of the application becomes uncertain when these classes have inconsistency by partial modification. Another is that it's important to decide the application's granularity at the design phase of the application. Even if the design of J2EE application is divided into Web-tier and EJB-tier on the concept, it's not so easy to actually split the existing application physically.

6.2.5 Debugging Java Pet Store Demo

TheJava Pet Store Demo application is a good J2EE reference application, but it's not good for debugging because almost all exceptions are replaced with GeneralFailureException or ServletException. These exceptions tell us only where the exceptions occur.

In Java Pet Store Demo, each module (EJB modules and Web modules) has the com.sun.j2ee.blueprints.util.tracer.Debug class. If you need more debug information for running Java Pet Store Demo, set the debuggingOn variable in the debug class to true, as follows:

```
public static final boolean debuggingOn = true;
```

However, this debug class is included in all modules, so you need to confirm which debug class is used in your WSAD test environment.

6.2.6 Problems encountered while splitting Java Pet Store Demo

This is a record of our experience in splitting Java Pet Store Demo Version 1.1.2. We hope it can serve as a useful reference.

Hangup of EJB client (lookup failure of EJBHome)

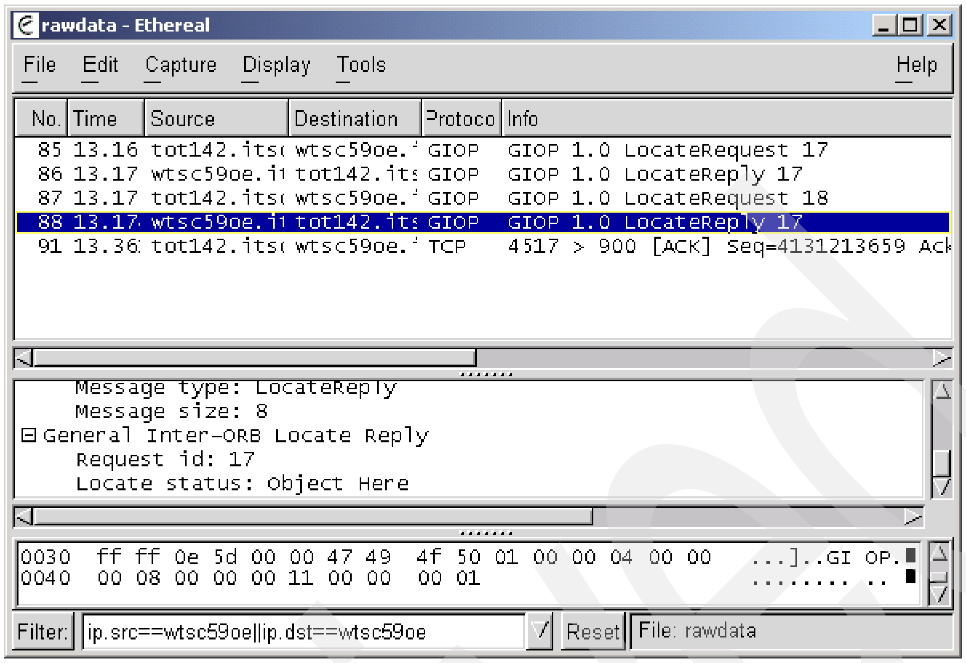
When we ran the EJB client of Java Pet Store Demo in the WebSphere Application Server Advanced Edition for the first time, the client hung up and displayed nothing.

Investigation showed that the client had stopped during lookup of EJBHome for the InitialContext object. In the initialization process of InitialContext, there are some GIOP interactions between WebSphere Application Server V4.01 for z/OS and OS/390 and the client in order to get the reference of the JNDI server. Then the client sends queries about the EJBHome object for the JNDI name. In this case, however, the result of a TCP/IP trace showed that the second GIOP reply message from WebSphere Application Server for z/OS and OS/390 was inaccurate.

GIOP messages consist of a pair of a request message and the reply message. The reply message must have the same ID as the request ID in the request message. In this case, the second GIOP reply message had the same ID as the first request message. Therefore, the client had to continue waiting for a reply message with the valid ID until time-out.

We found this problem using *Ethereal*, which is a tool for capturing TCP/IP packets, like *sniffer*. The result of a TCP/IP trace showed the following (see Figure 6-14 on page 172):

- The ID of the second reply message was inaccurate.
- There was no exchange on TCP/IP between WebSphere Application Server for z/OS and OS/390 and the client after that.


rawdata - Ethereal
File Edit Capture Display Tools Help
Filter: ip.src==wtsc59oe||ip.dst==wtsc59oe
Reset
File: rawdata

```
No. Time  Source      Destination Protoco Info
 85 13.16 tot142.its( wtsc59oe.' GIOP    GIOP 1.0 LocateRequest 17
 86 13.17 wtsc59oe.i1 tot142.its GIOP    GIOP 1.0 LocateReply 17
 87 13.17 tot142.its( wtsc59oe.' GIOP    GIOP 1.0 LocateRequest 18
 88 13.17 wtsc59oe.i1 tot142.its GIOP    GIOP 1.0 LocateReply 17
 91 13.36 tot142.its( wtsc59oe.' TCP     4517 > 900 [ACK] Seq=4131213659 Ack

    Message type: LocateReply
    Message size: 8
⊟General Inter-ORB Locate Reply
    Request id: 17
    Locate status: Object Here

0030  ff ff 0e 5d 00 00 47 49  4f 50 01 00 00 04 00 00   ...]..GI OP.
0040  00 08 00 00 00 11 00 00  00 01                     ........ ..
```

Figure 6-14 Mismatch of the request ID between the No. 57 and No.58 packets

Attention: This problem is now addressed by APAR PQ71503. For your reference, we still document how to debug and solve this kind of problem.

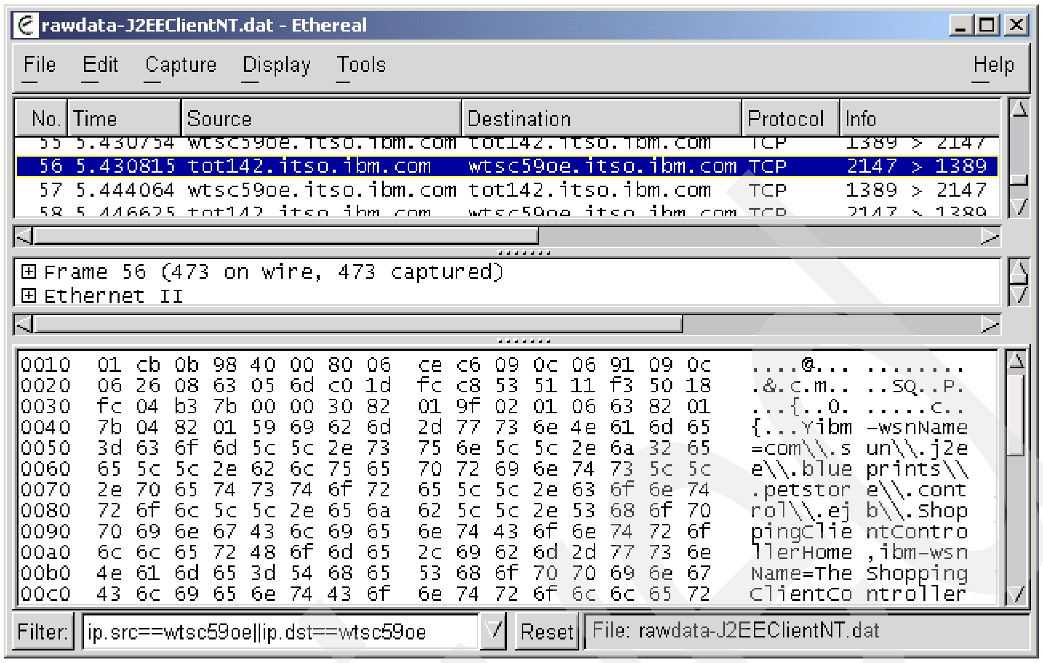
NamingNotFoundException for EJBHome lookup

After the above trouble was fixed, we got NamingNotFoundException in EJBHome lookup. We tried to execute this EJB client on WebSphere Application Server Advanced Edition and WebSphere Application Server Client Version 4, which can be downloaded from /usr/lpp/WebSphere/bin/J2EEClient_NT.zip on HFS, but the result was the same exception. Next, we found that the client was executed successfully with the Java Technology Client.

Note: The Java Technology Client used to be available at `http://www6.software.ibm.com/dl/websphere20/zosos390-p`. Now it is no longer available as a download because its function has been added to the code as of PTF version UQ72838. Apply service level UQ72838 to obtain the Java Technology Client.

We compared two TCP/IP traces of WebSphere Application Server Advanced Edition and the Java Technology Client, and found that two backslashes were added before a dot character in the case of WebSphere Application Server Advanced Edition (see Figure 6-15 on page 173). On the other hand, no backslash was added in the case of the Java Technology Client (see Figure 6-16 on page 173).

We avoided this by removing dot characters from each JNDI name that was specified in SMEUI.



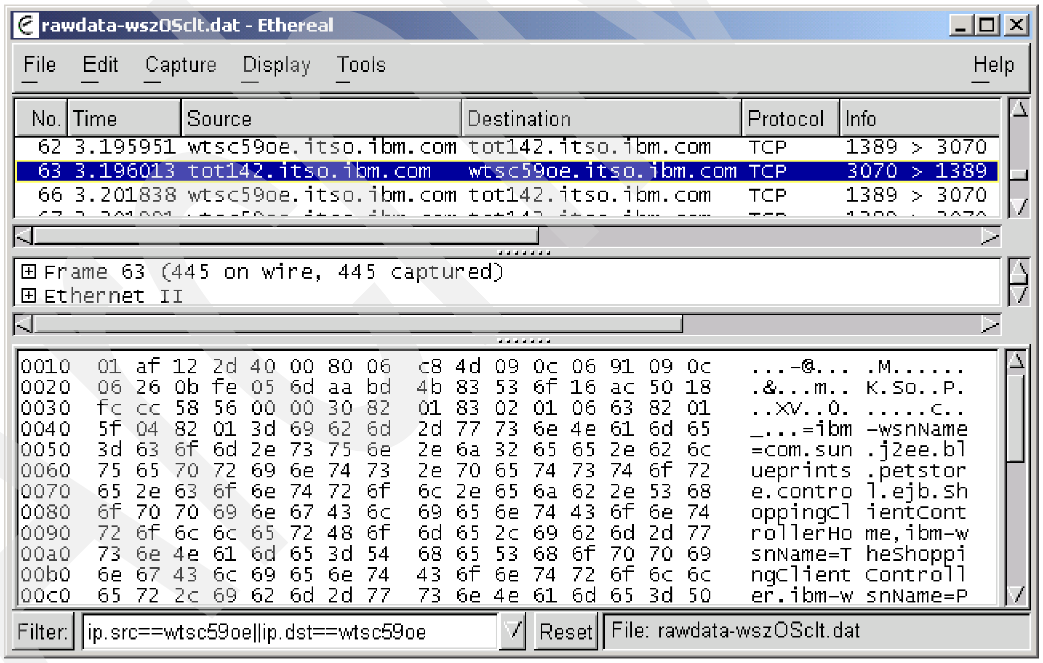
```
No. Time     Source                Destination           Protocol Info
 55 5.430754 wtsc59oe.itso.ibm.com tot142.itso.ibm.com   TCP      1389 > 2147
 56 5.430815 tot142.itso.ibm.com   wtsc59oe.itso.ibm.com TCP      2147 > 1389
 57 5.444064 wtsc59oe.itso.ibm.com tot142.itso.ibm.com   TCP      1389 > 2147
 58 5.446625 tot142.itso.ibm.com   wtsc59oe.itso.ibm.com TCP      2147 > 1389
```

```
⊞ Frame 56 (473 on wire, 473 captured)
⊞ Ethernet II
```

```
0010  01 cb 0b 98 40 00 80 06  ce c6 09 0c 06 91 09 0c   ....@... ........
0020  06 26 08 63 05 6d c0 1d  fc c8 53 51 11 f3 50 18   .&.c.m.. ..SQ..P.
0030  fc 04 b3 7b 00 00 30 82  01 9f 02 01 06 63 82 01   ...{..0. .....c..
0040  7b 04 82 01 59 69 62 6d  2d 77 73 6e 4e 61 6d 65   {...Yibm -wsnName
0050  3d 63 6f 6d 5c 5c 2e 73  75 6e 5c 5c 2e 6a 32 65   =com\\.s un\\.j2e
0060  65 5c 5c 2e 62 6c 75 65  70 72 69 6e 74 73 5c 5c   e\\.blue prints\\
0070  2e 70 65 74 73 74 6f 72  65 5c 5c 2e 63 6f 6e 74   .petstor e\\.cont
0080  72 6f 6c 5c 5c 2e 65 6a  62 5c 5c 2e 53 68 6f 70   rol\\.ej b\\.Shop
0090  70 69 6e 67 43 6c 69 65  6e 74 43 6f 6e 74 72 6f   pingClie ntContro
00a0  6c 6c 65 72 48 6f 6d 65  2c 69 62 6d 2d 77 73 6e   llerHome ,ibm-wsn
00b0  4e 61 6d 65 3d 54 68 65  53 68 6f 70 70 69 6e 67   Name=The Shopping
00c0  43 6c 69 65 6e 74 43 6f  6e 74 72 6f 6c 6c 65 72   ClientCo ntroller
```

Filter: ip.src==wtsc59oe||ip.dst==wtsc59oe   File: rawdata-J2EEClientNT.dat

Figure 6-15   The case of WebSphere Application Server Advanced Edition



```
No. Time     Source                Destination           Protocol Info
 62 3.195951 wtsc59oe.itso.ibm.com tot142.itso.ibm.com   TCP      1389 > 3070
 63 3.196013 tot142.itso.ibm.com   wtsc59oe.itso.ibm.com TCP      3070 > 1389
 66 3.201838 wtsc59oe.itso.ibm.com tot142.itso.ibm.com   TCP      1389 > 3070
```

```
⊞ Frame 63 (445 on wire, 445 captured)
⊞ Ethernet II
```

```
0010  01 af 12 2d 40 00 80 06  c8 4d 09 0c 06 91 09 0c   ...-@... .M......
0020  06 26 0b fe 05 6d aa bd  4b 83 53 6f 16 ac 50 18   .&...m.. K.So..P.
0030  fc cc 58 56 00 00 30 82  01 83 02 01 06 63 82 01   ..XV..0. .....c..
0040  5f 04 82 01 3d 69 62 6d  2d 77 73 6e 4e 61 6d 65   _...=ibm -wsnName
0050  3d 63 6f 6d 2e 73 75 6e  2e 6a 32 65 65 2e 62 6c   =com.sun .j2ee.bl
0060  75 65 70 72 69 6e 74 73  2e 70 65 74 73 74 6f 72   ueprints .petstor
0070  65 2e 63 6f 6e 74 72 6f  6c 2e 65 6a 62 2e 53 68   e.contro l.ejb.Sh
0080  6f 70 70 69 6e 67 43 6c  69 65 6e 74 43 6f 6e 74   oppingCl ientCont
0090  72 6f 6c 6c 65 72 48 6f  6d 65 2c 69 62 6d 2d 77   rollerHo me,ibm-w
00a0  73 6e 4e 61 6d 65 3d 54  68 65 53 68 6f 70 70 69   snName=T heShoppi
00b0  6e 67 43 6c 69 65 6e 74  43 6f 6e 74 72 6f 6c 6c   ngClient Controll
00c0  65 72 2c 69 62 6d 2d 77  73 6e 4e 61 6d 65 3d 50   er,ibm-w snName=P
```

Filter: ip.src==wtsc59oe||ip.dst==wtsc59oe   File: rawdata-wszOSclt.dat

Figure 6-16   The case of the Java Technology Client

6.2.7 Processing a Unicode XML file in WebSphere Application Server Advanced Edition Version 5

There are some XML files in Java Pet Store Demo that are for the Japanese version. One XML file uses Unicode as character encoding in order to be written in Japanese. But our WebSphere Application Server Advanced Edition Version 5 couldn't process this Unicode

XML file. So we had to convert the character encoding from Unicode to Shift_JIS using the native2ascii tool, as follows:

```
> ren screendefinitions.xml screendefinitions.xml.bak

> native2ascii -encoding Unicode screendefinitions.xml.bak | native2ascii -encoding
Shift_JIS -reverse > screendefinitions.xml
```

This XML file works In WebSphere Application Server Advanced Edition Version 4 without any modification.

6.2.8 Testing the application

You can check the application by manually accessing the WebSphere Application Server Advanced Edition which is assigned to the Web-tier. Therefore, you can make use of the methods described in 5.1.4, “Testing the application” on page 139.

Ethereal

If you want to confirm its behavior excepting application logs, you can do so by capturing TCP/IP packets. You will see the following packets in the TCP/IP trace:

- GIOP packets
- LDAP packets
- Packets connecting to DB2 for OS/390 and z/OS

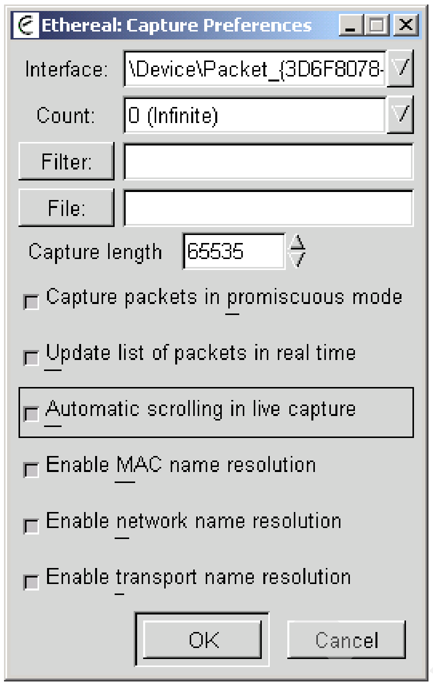
The following procedure is a brief method for capturing TCP/IP packets with Ethereal.

Note: Ethereal is an open source product and can be downloaded freely from http://www.ethereal.com/. Refer to the site for details about installation and operation.

1. Start Ethereal on Windows.
2. Fill in the bottom input form as follows:

   ```
   ip.src==hostname || ip.dst==hostname
   ```

   You need to replace *hostname* with the hostname that is assigned to the host on which WebSphere Application Server V4.01 for z/OS and OS/390 runs.
3. Click **Capture** -> **Start**.
4. Check all items on the Capture Preferences window (see Figure 6-17 on page 175).
5. If your PC has two or more network interfaces, select the network interface that you want to sniff from the Interface pulldown.



Interface: \Device\Packet_{3D6F8078-

Count: 0 (Infinite)

Filter:

File:

Capture length 65535

Capture packets in promiscuous mode

Update list of packets in real time

Automatic scrolling in live capture

Enable MAC name resolution

Enable network name resolution

Enable transport name resolution

Figure 6-17 The Ethereal Capture Preferences window

6. Press **OK**. TCP/IP packet capturing starts.
7. Trigger the application to trace.
8. Press **Stop** on the Capture window.
9. The result of capturing is displayed on the Ethereal window.
10. Click the line you want to see.

GIOP parser

In the TCP/IP trace you can find the GIOP datastream. You will need to format this data with a GIOP parser to interpret it.

The document “Under the hood: IORs, GIOP and IIOP” describes the appropriate techniques for GIOP parsing. It is available at:

```
http://www-106.ibm.com/developerworks/webservices/library/ws-underhood/
```

Part 3

Appendixes

Integrated and multi-platform scenario sandbox

In this appendix, we describe the hardware, software, and network infrastructure we used to run our test and deployment scenarios.

Environment used for development and deployment

For development and deployment purposes, we used the following software:

- Base operating system Windows 2000 Workstation with fixpack 3
- WebSphere Studio Application Developer V5
- WebSphere Application Server for z/OS System Administration User Interface V4.01.023
- Application Assembly Tool for WebSphere z/OS V4.00.32
- WebSphere Studio Workload Simulator Client, level 02302

For an overview of the physical layout, the mapping of names and addresses, and how these workstations are tied into the network, refer to Figure A-1 on page 180.

WebSphere Application Server distributed test environment

At the distributed site, we used the software components listed here:

- Base operating system Windows 2000 Server with fixpack 3

Table A-1 Different software setup

Server	Usage	Database	HTTP server
tot127	WebSphere HTTP plug-in	N/A	IBM HTTP Server V1.3.26
tot134	WebSphere Application Server Enterprise Edition V4.0.4	IBM DB2 UDB Version 7.2 with fixpack 7	IBM HTTP Server V1.3.19
tot148	WebSphere Application Server Enterprise Edition V5.0.0	IBM DB2 UDB Version 7.2 with fixpack 7	IBM HTTP Server V1.3.26

For an overview of the physical layout, the mapping of names and addresses, and how these servers are tied into the network, refer to Figure A-1 on page 180.

WebSphere on z/OS environment

We used the following software environment on the z/OS side:

- Two LPARs with the z/OS 1.3 base operating system
- WebSphere Application Server V4.01 at build level W401408
- WebSphere Studio Workload Simulator Engine at build level 02280Z

For an overview of the physical layout, the mapping of names and addresses, and how the mainframe is tied into the network, refer to Figure A-1.

Physical network structure

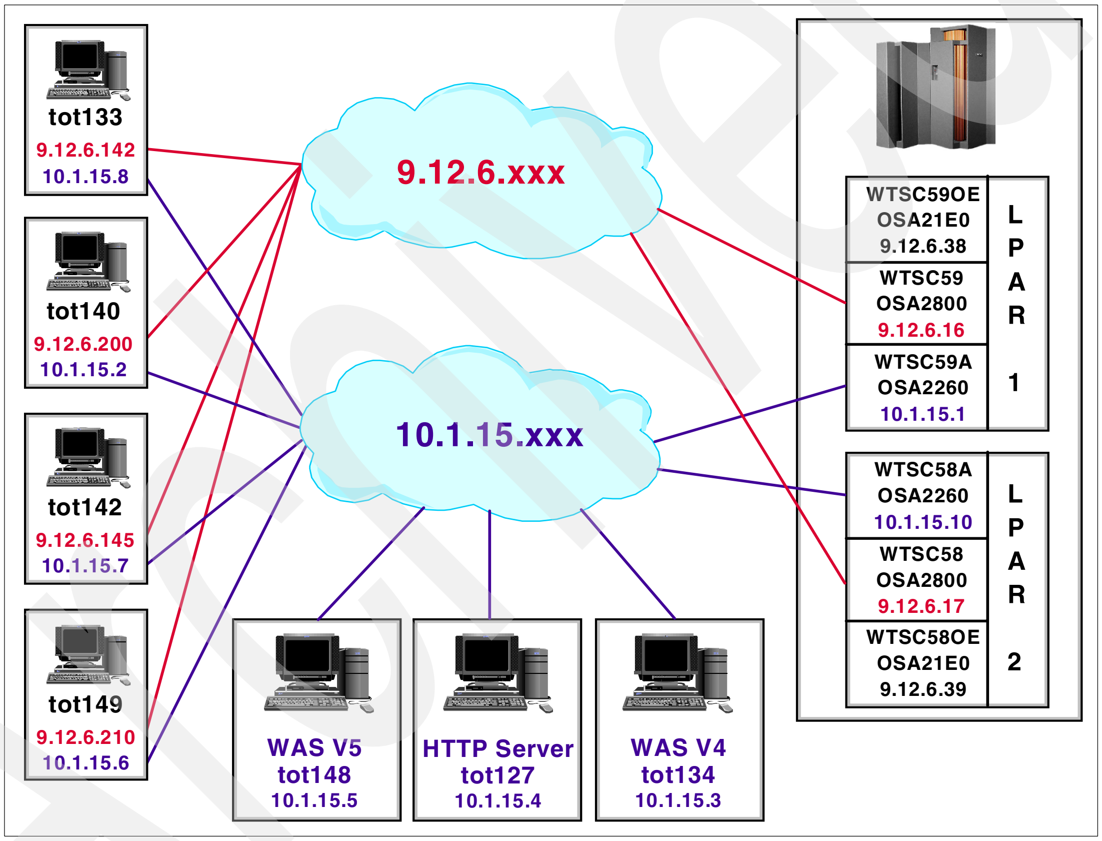


Figure A-1 Physical network structure

6.2.9 Our testing tools

To generate load to our application and to analyze the impact of changes to the application and its environment, we used the IBM WebSphere Studio Workload Simulator. In the following sections, we provide a brief introduction to the Workload Simulator and show the setup of our test environment. We choose Workload Simulator because it is an easy-to-install, easy-to-handle, and highly scalable test tool.

IBM WebSphere Studio Workload Simulator

The Workload Simulator and generates Web traffic to analyze the performance of the Web-serving environment when confronted with production-level loads while simulating large numbers of virtual users. Workload Simulator can also be used to perform quality assurance on Web applications and to test the functionality of Web servers.

Workload Simulator consists of two components: a controller and an engine. The controller is the user interface with the engine, which in turn generates the Web traffic. All test functions are accessible through the controller. This component resides on a Windows workstation and offers a Windows graphical user interface (GUI) for managing all aspects of the load-testing process: test scripts can be created and edited, simulation runs can be set up, executed and monitored, and test results can be analyzed without leaving the Windows GUI.

For further flexibility you can set up a so-called Web Monitor, which is an optional component of Workload Simulator. It allows you to monitor server performance and Workload Simulator engines from a Web browser. This option requires an already installed and functioning Web server on the z/OS system running the Workload Simulator engines.

The load-testing process can be differentiated in three steps, capture, playback, and analysis, as follows:

- **Capture**

  While the user navigates through a Web session, test scripts are automatically generated. The capture function records the session's Web traffic and turns it into a test script ready for immediate playback. If needed, you can add more complexity to the test script with an editor function, in order to have the testing process provide a better simulation of the actions of a group of real users.

- **Playback**

  This function executes the previously captured and edited scripts. For a flexible execution, several runtime parameters can be set: number of repetitions, or should run until manually stopped, or should run a certain period of time; and the number of virtual users to be simulated. Many more options to simulate real-life conditions are possible.

- **Analysis**

  After a test script is defined and the simulation launched, the test can be monitored in real-time during execution. You can monitor a test either through the controller's Windows GUI or through a Web browser. The remote monitoring capability is helpful for extended runs (for example, over a weekend), as it obviates the need for the presence of test personnel at the test site.

  In addition to its real-time monitoring and analysis tools, Workload Simulator also provides a graphing tool to help you create more in-depth analyses of your Web applications. Response time, data throughput, throughput of page elements, and CPU or memory utilization are available for plotting. For in-depth analysis of test results, different levels of logging of the test activity are possible, and the HTTP activity of each simulated client can be traced.

To maintain the same level of security you may have in a production environment, Workload Simulator supports Secure Sockets Layer (SSL) security protocol and handles capture and playback through a SOCKS firewall.

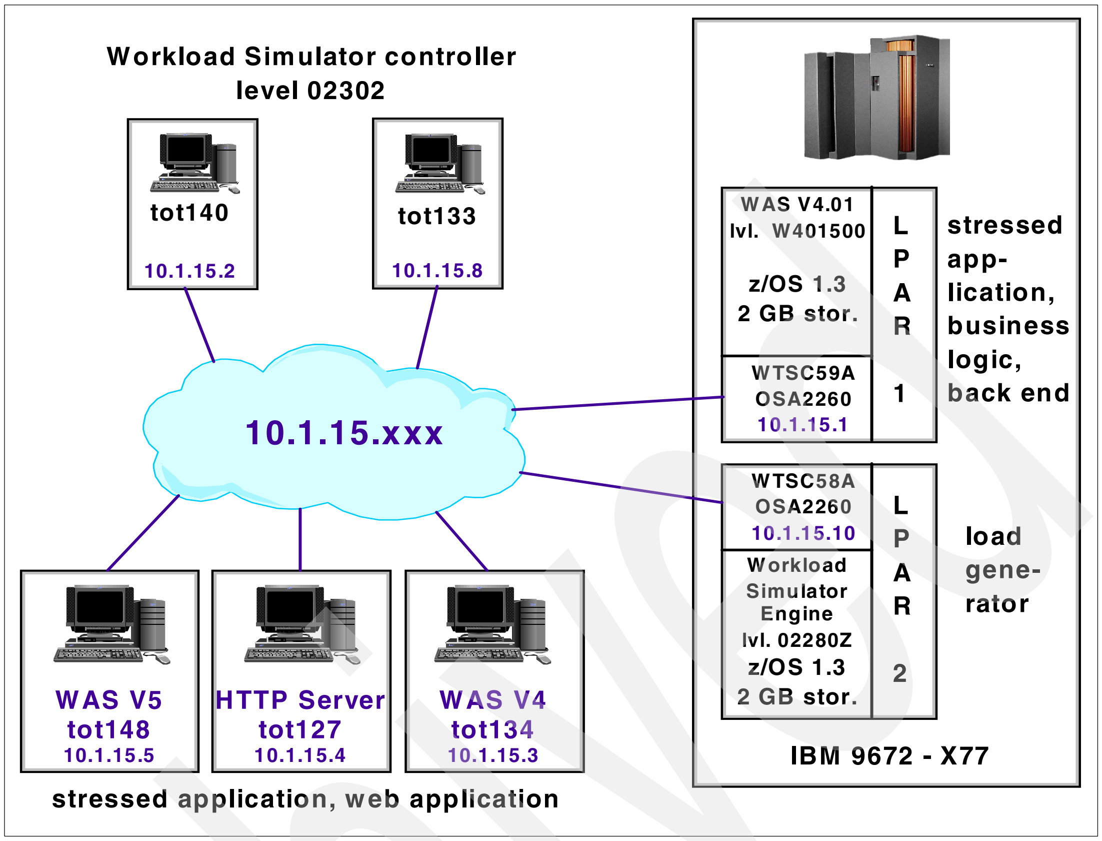


Figure A-2 Workload simulation and performance measurement environment

Our Workload Simulator setup

In order to obtain more realistic performance numbers, we installed Workload Simulator in a different LPAR from where the WebSphere Application Server for z/OS was running. We used a similar environment (both partitions with 2 GB main storage, base operating system z/OS V1.3, except the processors).

To be able to provide a constant and comparable workload, we assigned the two processors as dedicated to the LPAR where the Workload Simulator Engine was running. We assumed that our environment was located behind the second firewall. For further details, refer to Figure A-2.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 184. Note that some of the documents referenced here may be available in softcopy only.

- *CICS Transaction Gateway V5 The WebSphere Connector for CICS,* SG24-6133
- *Java Connectors for CICS Featuring the J2EE Connector Architecture,* SG24-6401
- *From code to deployment: Connecting to CICS from WebSphere for z/OS,* REDP0206
- *z/OS WebSphere and J2EE Security Handbook,* SG24-6846
- *Enabling High Availability e-business on e-server zSeries,* SG24-6850
- *Linux for S/390,* SG24-4987
- *Linux on IBM zSeries and S/390: ISP/ASP Solutions,* SG24-6299
- *IBM WebSphere V4.0 Advanced Edition: Scalability and Availability,* SG24-6192
- *Monitoring WebSphere Application Peformance on z/OS,* SG24-6825
- *z/OS WebSphere and J2EE Security Handbook,* SG24-6846
- *WebSphere Edge Server: New Features and Functions in Version 2,* SG24-6511

Other publications

These publications are also relevant as further information sources:

- *WebSphere Studio Workload Simulator User’s Guide,* SC31-6307
- *WebSphere Studio Workload Simulator Programming Reference,* SC31-6308
- *WebSphere Studio Workload Simulator User’s Guide,* SC31-6307
- *WebSphere Application Server for z/OS and OS/390 v4.0.1: Migration,* GA22-7860
- *WebSphere Application Server for z/OS and OS/390 v4.0.1: Installation and Customization,* GA22-7834
- *Assembling Java™ 2 Platform, Enterprise Edition (J2EE™) Applications,* SA22-7836
- *HTTP Server Planning, Installing and Using,* SC31-8690
- *IBM DB2 Connect Enterprise Edition for UNIX V7 Quick Beginnings,* GC09-2952
- *Understanding HTTP Session Management,* WP100316, TechDocs whitepaper
- *Overcoming a Problem Running Simulated Message Driven Beans,* WP100301, TechDocs whitepaper

Online resources

These Web sites and URLs are also relevant as further information sources:

- TechDocs, a source for technical white papers

  `http://www.ibm.com/support/techdocs/atsmastr.nsf/Web/Techdocs`

- API docs for IBM application servers

  `http://www.ibm.com/software/webservers/appserv/doc/v40/aee/wasa_common/apidocs/index.html`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

`ibm.com/redbooks`

Index

IBM Redbooks

Building Multi-Tier Scenarios for WebSphere Enterprise Applications

(0.2”spine)
0.17”<->0.473”
90<->249 pages

Building Multi-Tier Scenarios for WebSphere Enterprise Applications

Architecting an infrastructure for seamless 3-tier integration

Developing, deploying, and tooling for interoperability

Security, performance, cost, and management views

This IBM Redbook will help you build multi-tier scenarios for WebSphere Enterprise Applications. It applies to WebSphere Application Server V4.01 for z/OS and OS/390.

We cover the aspects of architectural, organizational, and technical issues that you need to consider when selecting an application and runtime design. This book can be used in conjunction with Patterns for e-business when you are faced with making decisions about application patterns and are looking for supporting information.

Because our analysis is done from the perspective of the z/OS platform, we discuss strategies for offloading Web applications from z/OS or from WebSphere for z/OS. We provide an overview of different scenarios and give guidance on platforms, security, deployment, performance, scaling, and EIS integration.

Using this redbook will enable you to architect an infrastructure for seamless three-tier integration by helping you to develop, deploy, and tool the application for interoperability, as well as install and configure the different infrastructures.

SG24-6956-00 ISBN 0738499525