

e-business Cookbook for z/OS Volume I: Technology Introduction

**Planning your software for e-business
enablement on the z/OS platform**

**Functions and features of
e-business software on z/OS**

**Sample scenarios and
technology options**



**Tamas Vilaghy
Frank Pani
Patrick C. Ryan
Bruce Smith**



International Technical Support Organization

**e-business Cookbook for z/OS Volume I:
Technology Introduction**

July 2002

Archived

Take Note! Before using this information and the product it supports, be sure to read the general information in “Notices” on page ix.

Second Edition (July 2002)

This edition applies to Version 4.0.1 of WebSphere Application Server for z/OS, Program Number 5655-F31 for use with the z/OS Operating System.

Comments may be addressed to:
IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2000, 2002

Note to U.S. Government Users – Documentation related to restricted rights – Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xi
Notice	xiii
Comments welcome	xiii
Part 1. Overview	1
Chapter 1. Introduction	3
1.1 Where the Web is going	4
1.2 What e-business is	4
1.3 What you need to do e-business	6
1.3.1 The importance of scalability and availability	6
1.3.2 Performance	7
1.4 The value of zSeries	7
1.4.1 Scalability	8
1.4.2 Security	8
1.4.3 Availability	9
Chapter 2. Technology overviews	11
2.1 Object-oriented technology	12
2.1.1 The procedural model	12
2.1.2 The object-oriented model	13
2.1.3 Object program source code	14
2.1.4 Advantages of object-oriented technology	14
2.2 The Java language	15
2.3 JavaBeans	15
2.3.1 Characteristics	15
2.3.2 Does Java imply object-oriented	17
2.4 The execution environment: Java Virtual Machine	17
2.5 Java 2 Platform	19
2.5.1 Java Standard Edition	19
2.5.2 Java 2 Enterprise Edition	20
2.6 J2EE applications	22
2.6.1 J2EE standard services	23
2.6.2 J2EE containers	26
2.6.3 Application layers	28

2.6.4	Servlets	29
2.6.5	JavaServer Pages (JSPs)	30
2.6.6	Enterprise JavaBeans (EJBs)	30
2.7	Deployment	33
2.7.1	Deployment and deployment descriptors	34
2.7.2	The J2EE packages and their deployment descriptor	35
2.8	CORBA Business Objects	35
2.9	Extensible Markup Language (XML)	36
2.9.1	XML Toolkit for z/OS and OS/390	39
2.9.2	XML Parser for z/OS and OS/390, Java Edition	40
Chapter 3.	The components of e-Business	43
3.1	Introduction	44
3.2	z/OS environment	45
3.2.1	Supporting components	45
3.2.2	TCP/IP	46
3.2.3	z/OS Unix Systems Services (USS)	46
3.2.4	Hierarchical File System (HFS)	47
3.2.5	LDAP/DB2	48
3.2.6	Workload Manager (WLM)	49
3.2.7	Security/SSL	49
3.2.8	LOGGER/RRS	49
3.3	WebSphere Application Server environment	50
3.4	Application development tools	56
3.4.1	Visual Age for Java	56
3.4.2	WebSphere Studio	56
3.4.3	WebSphere Studio Application Developer	57
Chapter 4.	Sample scenarios	59
4.1	Scenario 1: Simple Web serving	61
4.1.1	Variations	62
4.2	Scenario 2: Web-enabling current systems	62
4.2.1	Client accessing the back-end system through a Web server	63
4.2.2	Variations	64
4.2.3	Host on Demand	65
4.3	Scenario 3: New business logic scenario	66
4.3.1	Enterprise JavaBeans	66
4.3.2	Variations	68
4.3.3	Existing transaction server	68
Chapter 5.	Transaction services	69
5.1	Introduction	70
5.2	Transaction servers versus application servers	70
5.3	Transactions in e-business	70

5.4 The Unit of Work process	71
5.4.1 Resource Recovery Services (RRS).....	71
5.4.2 The two-phase commit protocol	71
Chapter 6. Security management	75
6.1 Overview of J2EE security	76
6.1.1 Programmatic security	76
6.1.2 Declarative security.....	77
6.1.3 Role-based authorization	77
6.1.4 Terminology used for J2EE security	79
6.2 Authentication and authorization in J2EE containers	80
6.2.1 Web container authentication and authorization	80
6.2.2 EJB container authentication and authorization	85
6.3 Resource authentication	86
6.4 J2EE security and WebSphere for z/OS	87
6.5 Directory services and LDAP	90
6.6 Firewalls	91
Chapter 7. Managing workload	97
7.1 Overview	98
7.2 z/OS tools	99
7.3 Workload Manager	101
7.3.1 Compatibility mode	101
7.3.2 Goal mode.....	102
7.4 WLM and HTTP Server modes	104
7.4.1 Stand-alone server	104
7.4.2 Scalable server	104
7.4.3 Multiple servers	105
Part 2. The components in detail.....	107
Chapter 8. The IBM HTTP Server	109
8.1 Introduction and role in e-business	110
8.2 Server modes	110
8.3 Ways to access the HTTP server	112
8.4 How the HTTP Server works.....	112
8.5 Passing control from the HTTP server	114
8.6 How the CGI interface works.....	115
8.7 e-business-related functionalities	117
8.7.1 Basic functions	117
8.7.2 Security functions	119
8.7.3 File caching.....	119
Chapter 9. WebSphere Plugin environment	121

9.1 Introduction and role in e-business	122
9.2 Where to run your servlets	122
9.3 How the WebSphere Plugin works	123
9.3.1 Using servlets in the Plugin to talk to backend systems	124
9.3.2 Using servlets in the Plugin to talk to EJBs	125
9.3.3 Accessing servlets in a Web container via the Plugin	126
Chapter 10. J2EE Application Server	127
10.1 Introduction and role in e-business	128
10.2 How it works	129
10.3 Passing Control to a J2EE Application Server	132
10.3.1 Using the WebSphere Plugin	133
10.3.2 Using the HTTP Transport Handler	134
10.4 e-business related functionalities	134
Chapter 11. WebSphere MQ	137
11.1 Introduction and role in e-business	138
11.2 How it works	138
Chapter 12. DB2	141
12.1 Introduction and role in e-business	142
12.2 Accessing DB2	142
12.3 e-business-related functionalities	143
Chapter 13. CICS	147
13.1 Introduction and role in e-business	148
13.2 CICS Web enablement	148
13.2.1 The separation of presentation and business logic	149
13.2.2 CICS Web support	150
13.3 CICS WebServer Plugin	154
13.3.1 CICS Transaction Gateway	157
13.3.2 CICS EJB support	160
Chapter 14. IMS	165
14.1 Introduction and role in e-business	166
14.2 Open Transaction Manager Access	166
14.2.1 IMS Connect	167
14.2.2 IMS Connector for Java	168
Part 3. Application development	171
Chapter 15. Application design	173
15.1 Application design considerations	174
15.2 Your objective	174
15.3 Application functionality	174

15.4 Existing systems	175
15.5 Software levels	175
15.6 Application layers	176
15.7 Object-oriented or traditional language	178
15.8 Skills	180
15.9 Performance	180
15.10 Scalability	180
15.11 Security	181
Chapter 16. Application development	183
16.1 Application development	184
16.2 First Generation - non-Java.	184
16.3 Migration from First Generation to Second and Third Generation	185
16.4 Second Generation - Java.	186
16.4.1 Coding in Java	186
16.4.2 Object-oriented or not	186
16.4.3 Transactional or not	187
16.4.4 Deployment options	187
16.4.5 Developing Java applications	188
16.4.6 Standards	189
16.4.7 Connectors	189
16.4.8 Java Native Interface (JNI)	189
16.4.9 Deploying in WebSphere Application Server on z/OS or OS/390 ..	190
16.5 Migrating from Second Generation to Third Generation	191
16.6 Third Generation - Enterprise JavaBeans	192
16.6.1 Coding in Java	192
16.6.2 Transactional or not	192
16.6.3 Connectors: J2EE Connector Architecture	192
16.6.4 Deployment options	193
Chapter 17. Development tools	195
17.1 WebSphere Application Development Solution.	196
17.1.1 Quick start scenarios.	196
17.2 IBM VisualAge Family	197
17.2.1 VisualAge for Java	198
17.3 IBM WebSphere Studio.	199
17.4 WebSphere Studio Application Developer	200
17.5 Lotus Domino Designer.	202
Related publications	203
IBM Redbooks	203
Other resources	204
Referenced Web sites	204
How to get IBM Redbooks	205

IBM Redbooks collections.....	205
Index	207

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.


This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

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

Redbooks(logo)TM 

AIX[®]

Balance[®]

CICS[®]

CICSplex[®]

DB2[®]

DB2 ConnectTM

DRDA[®]

ES/9000[®]

IBM[®]

IMSTM

IMS/ESA[®]

Language Environment[®]

MQSeries[®]

Multiprise[®]

MVSTM

Net.Data[®]

NetView[®]

OS/2[®]

OS/390[®]

OS/400[®]

Parallel Sysplex[®]

PerformTM

QMFTM

RACF[®]

RMFTM

S/390[®]

SecureWay[®]

SPTTM

System/390[®]

Tivoli[®]

VisualAge[®]

VTAM[®]

WebSphere[®]

z/OSTM

z/VMTM

zSeriesTM

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

Lotus[®]

Lotus Notes[®]

Notes[®]

DominoTM

Word Pro[®]

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare 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.

C-bus is a trademark of Corollary, 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 is the first volume in the “e-business Cookbook for z/OS” series. It provides an overview of e-business and how it relates to zSeries and its z/OS operating system.

The technology portfolio on z/OS in the e-business enablement arena is growing rapidly. Multiple solutions are possible, each requiring a different set of products. In this book we define several scenarios, each using a different “style” of technology. The scenario you adopt is dependent on your strategy, the skills in your organization, and the functionality you need. Within that scenario, various technology options are available; your selections will depend mostly on the subsystems deployed in your organization.

The other volumes in this series are *e-business Cookbook for z/OS, Volume II: Infrastructure*, SG24-5981, and *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980. They address the implementation and configuration of the technology, and describe how to develop solutions.

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.

Tamas Vilaghy is a project leader at the International Technical Support Organization, Poughkeepsie Center. He leads redbook projects dealing with e-business on zSeries servers. Before joining the ITSO, he worked in System Sales Unit and Global Services departments of IBM Hungary. Tamas spent two years in Poughkeepsie from 1998 to 2000 dealing with zSeries marketing and competitive analysis. From 1991 to 1998 he held technical, marketing and sales positions dealing with zSeries.

Patrick C. Ryan is a Senior OS/390 e-business Consultant working for the IBM e-business Center for Technical Competency (eCTC). He has over 28 years of experience in the IT industry. In his current position, he has worked with major IBM customers, installing and configuring WebSphere, both in the US and Europe. He has co-authored several redbooks about WebSphere on the OS/390 and z/OS platforms. In the past he has worked as an MVS Programming Support

Representative, VTAM level 2/3 Specialist, VM designer, LAN Administrator and IT Specialist for the S/390 and z/OS division. He has also presented at Share and Guide meetings and demonstrated new technology at numerous CIO and technical conferences.

Bruce Smith is an IT Specialist at IBM UK. He has been an OS/390 systems programmer for the last ten years, since graduating with a B.Eng in Engineering Electronics from The University of Warwick. He has worked on installation, customization, and ongoing support of commercial e-business running entirely on OS/390. If you have questions regarding this book, or Internet connectivity on OS/390 or z/OS, feel free to contact him via e-mail at smithbj@uk.ibm.com.

Frank Pani is a Technical Services Professional for Host Software Services, IBM Global Services from Canada. Since June, 2000 he has worked on developing DB2 solutions in Java for OS/390. He holds a degree in Combinatorics & Optimization from the University of Waterloo, ON Canada.

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

- ▶ Ivan Joslin
IBM WebSphere for z/OS and OS/390 development lab, Poughkeepsie
- ▶ Alex Louwe Kooijmans, WebSphere and Java specialist, project leader for the first edition of the Cookbook
- ▶ Rich Conway
ITSO, Poughkeepsie
- ▶ Holger Wunderlich
Project leader, ITSO, Poughkeepsie
- ▶ Phil Wakelin
Project leader and CICS specialist, ITSO, Poughkeepsie
- ▶ Mike Cox, Don Bagwell
Washington Systems Center
- ▶ Hilon Potter
IBM eTP Center, Poughkeepsie
- ▶ Thomas Hackett
IBM New Technology Center, Poughkeepsie

Notice

This publication is intended to help customers, business partners, and IBMers to learn the components and their roles in doing e-business on zSeries processors with the z/OS operating system. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere Application Server for z/OS. See the PUBLICATIONS section of the IBM Programming Announcement for WebSphere Application Server for more information about what publications are considered to be product documentation.

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 the address on page ii.



Part 1

Overview

Here we introduce e-business and how it relates to zSeries and its z/OS operating system.

In Chapter 1 we discuss Web evolution and the value of zSeries in e-business.

In Chapter 2 we discuss e-business-related technologies, such as object oriented, Java, CORBA, XML and J2EE technologies.

In Chapter 3 we give a brief overview of the software components of e-business on z/OS. In Part 3 we go into greater detail.

In Chapter 4 we discuss possible scenarios a customer may employ to do e-business.

In Chapter 5, 6, and 7 we discuss three important aspects of e-business: Transactional capabilities, security, and workload management. These three areas are among the strengths of z/OS. They are also among the most important aspects of e-business today.

Introduction

This chapter provides an introduction to e-business on the IBM z/OS operating system running on a zSeries hardware platform.

1.1 Where the Web is going

The Web is changing many aspects of our lives, but no area is undergoing as rapid and significant a change as the way e-businesses operate. As businesses incorporate Internet technology into their core business processes, they start to achieve real business value.

Today, companies large and small are using the Web to communicate with their partners, to connect with their backend data systems, and to transact commerce. This is e-business, where the strength and reliability of traditional information technology meet the Internet.

This new Web plus IT paradigm merges the standards, simplicity, and connectivity of the Internet with the core processes that are the foundation of business. The new applications are interactive, transaction-intensive, and let people do business in more meaningful ways.

With e-business comes the need to connect the new e-business applications to existing applications, data, inventory, and mission-critical transactions.

Fortunately, for z/OS customers there are easy solutions that can be deployed quickly and safely by Web-enabling these existing applications and data while maintaining the necessary qualities of service.

These solutions provide a high return on investment (ROI) because the time-to-market is short, the cost of development and testing is significantly reduced, and risks are low.

Existing assets are battle-proven, fully functional, and have fast response times. z/OS has evolved to become a highly available, secure, and scalable platform, well suited for the modern demands of e-transaction processing.

1.2 What e-business is

e-business is the transformation of key business processes through the use of Internet technologies. In order to remain competitive, improve service to customers, and build new sources of revenue, companies must extend their existing applications and provide new Web-based applications now.

To manage these transitions smoothly, start simple and build on what you have.

The term *e-business* describes a vast and still growing area of enterprise computing. This book describes three areas:

- ▶ **Web serving:** Creating a simple Web site that provides information. The Web site offers static information only, such as company information, or annual reports.

The IBM HTTP Server for z/OS provides this function. For more information see Chapter 8, “The IBM HTTP Server” on page 109.

- ▶ **Web-enabling current systems:** Expanding your Web site to access current systems or databases. The Web site provides dynamic content, such as allowing a customer to ask a question and get an answer.

The IBM WebSphere plugin environment provides this function. For more information see Chapter 9, “WebSphere Plugin environment” on page 121.

You can do both Customer Relationship Management (CRM) and Supply Chain Management (SCM).

CRM enables customers to find out when their orders were shipped, or query their account balances. They can receive answers to their questions immediately.

SCM provides a way to disseminate information and enable two-way communications between companies within a network of suppliers, vendors, and distributors. You can start Web-enabling legacy systems to provide visibility to selected partners, suppliers, and customers, and to integrate the supply chain into your company's other processes.

- ▶ **Developing new business logic:** Creating a Web site that provides “e-transactions”.

This means re-engineering existing applications and providing new components to transform business processes through total business integration. It includes e-commerce, an electronic presentation of goods and services, online order-taking, and handling of online payments and transactions.

The WebSphere Application Server J2EE Application Server environment provides this ability. For more information see Chapter 10, “J2EE Application Server” on page 127.

The move to total business integration and transformation of your business processes is not an overnight process. Enterprises should develop a long-range plan and position themselves for the evolution to component-based development.

Java component-based development allows organizations to be more responsive to new requirements, establish a competitive advantage, and lower the cost of development.

1.3 What you need to do e-business

e-business is about integration and transformation. It is important to understand the requirements for e-business in your installation.

Most large sites face the same challenges, the most significant of which are unpredictable growth and the ability to have solutions ready for unknown problems.

Important: Remember, *you don't control the Internet!*

The skills required to develop components that display information are different from those required to develop components that process transactions.

Failure to optimize graphics, frequent table scans, and joins of multiple tables result in I/O bottlenecks that combine to degrade performance. Site availability can be stressed by unpredictable traffic and inadequate systems management.

The problems you may be facing can be compounded by poor application design and systems that are poorly configured, underpowered, or both.

Important: Remember, *architecture matters.*

1.3.1 The importance of scalability and availability

Internet traffic is unpredictable. *Scalability* refers to a Web site's ability to adapt readily to a greater or lesser intensity of use, volume, or demand while still meeting business objectives.

Understanding the scalability of your system's components and applying appropriate scaling techniques can greatly improve your Web site's availability and performance. Scaling techniques are especially useful in multi-tier architectures when you evaluate components associated with the Web server, the application servers, and the database servers.

Scaling a multi-tiered e-business solution from end to end means managing the performance and capacities of each component within each tier. The total solution is as strong as the weakest component.

The basic objectives of scaling a component/system are to:

- ▶ Increase the capacity or speed
- ▶ Improve the efficiency
- ▶ Shift or reduce the workload

Increasing the scalability of one component may change the dynamics of the site, thereby moving a bottleneck to another component. The scalability of the solution depends on the ability of each component to scale to meet increasing demands.

Availability and reliability are also equally important. You may have a requirement to run your online business 24 hours a day, 7 days a week, 365 days a year. You cannot afford downtime.

1.3.2 Performance

Because of the exposure of the Internet the performance of your IT organization can have enormous visibility and significance to the success of your company's e-business.

You should select Web site components for their ability to meet your needs of today and tomorrow. You require more scalability as you add more work. New Java technologies require more resources than older technologies, because they require more layers of abstraction.

Mature technologies perform better than brand new ones. Normally, function is introduced first, then performance is improved as you see how people use the function.

You need to manage the connections to the Web server, the Web application server, and the databases to minimize the total number of connections needed for an end-to-end system, as well as to eliminate the overhead of setting up connections during normal operations.

To reduce the overhead associated with establishing connections between each layer, a pool of pre-established connections is maintained and shared across multiple requests flowing between the layers.

For example, most application servers enable database connection managers to allow connection reuse.

It is important to note that a session may use multiple connections to accomplish its tasks, or many sessions may share the same connection.

1.4 The value of zSeries

The fastest path to a successful e-business is to leverage your existing, stable IT assets. Statistics say that more than 70% of the world's data and more than 80% of the world's business transactions already reside on IBM servers, with most residing on zSeries systems.

The traditional strengths of the zSeries server are also essential for e-business survival. The zSeries operating system (z/OS) knows how to do transactions and the zSeries hardware sets the standards for availability, scalability, workload management, and low total cost of computing.

The high availability is the result of many years of refinement—and that kind of robustness is not easily achieved. Specific zSeries hardware has been designed over the years to assist the operating system and application subsystems to create a highly reliable and serviceable system.

The zSeries Parallel Enterprise Server families have additional built-in functions that help to reduce planned and unplanned outages.

1.4.1 Scalability

Beyond the new *Capacity Upgrade on Demand* functionality, zSeries Parallel Sysplex clustering technology provides the closest thing to continuous computing available today: a design point of 99.999%. That can mean no more than five minutes of planned or unplanned downtime per year.

zSeries offers virtually unlimited growth. Within the family, it is possible to move from one-way servers to 12-way systems. A zSeries Parallel Sysplex cluster can link up to 32 servers to create a single very large computing resource with almost linear scalability.

Another key to the robust performance of seamless workload balancing. Running all your applications on a single system rather than on a lot of smaller ones means all the resources are available to all the applications, all of the time, with little or no bottlenecks.

With zSeries, you get what is described by some industry consultants as one of the lowest cost-per-user computing environments in the industry.

1.4.2 Security

Security is the cause of much anxiety. Nobody wants to be a victim. Security is more than just putting up a firewall and turning on Secure Sockets Layer (SSL), although those are important elements. Not only does z/OS support its own robust firewall, but the zSeries Cryptographic Coprocessor is standard in the latest families of processors. This processor provides a hardware-assist for SSL, Virtual Private Network connections, and other cryptographic processing.

Of course, everyone has some sort of firewall and SSL support these days. What often goes unmentioned when talking about e-business security is system level security. Security has always been a critical design point for z/OS and its predecessors, OS/390 and MVS.

On a z/OS system, every address space and every task within an address space runs under a security environment. This environment is based on a resource control manager such as the z/OS SecureWay Security Server (formerly known as RACF). Identification and authorization requests are routed to resource control managers that help protect system resources and user databases from unauthorized access. User IDs and passwords can be used to identify Web users, as well as industry standard X.509 digital certificates.

Even more fundamentally, z/OS address spaces separate applications from each other to minimize the risk of one program corrupting another program's private storage/data area.

Hardware storage protection keys prevent user programs from altering system storage.

The Authorized Program Facility (APF) prevents unknown user programs from ever running in “authorized” mode. System level security is ingrained in z/OS.

1.4.3 Availability

In an e-business world, availability is critical. But zSeries people do not think about availability in terms of minimizing the downtime between outages. We think in terms of minimizing outages, period. Again, this goes back to heritage.

Since the beginning, the basic assumption of zSeries was that hundreds or even thousands of users would be depending on it, which is why its error detection and correction systems are so ingrained. Nearly 60% of z/OS software deals in some way with isolating, recovering from, and documenting errors.

The Alternate CPU Recovery (ACR) of z/OS automatically and dynamically isolates a failed processor from service without requiring a system outage. For the ultimate in availability, there is zSeries Parallel Sysplex.

Another aspect of availability is the need for planned outages for data backup and recovery on some platforms.

zSeries does not need to use “triple mirroring” of data to reduce planned outage windows for routine database housekeeping.

Backups, recoveries, and DB2 database reorganizations are done concurrently on z/OS. They are done in the background, on a single instance of the data, while mission-critical applications continue to run with the required response times and service levels. zSeries operations people take this for granted.

Technology overviews

One of the difficulties faced by people skilled in z/OS is understanding the terminology used in the e-business environment. Often we have no idea what the e-business people are talking about! Some of the concepts are well known to us, but new terminology creates a barrier to understanding.

This chapter introduces some of the terminology used, including object-oriented technology, Java, servlets, JSPs, JavaBeans, Enterprise JavaBeans (EJBs) and XML.

In our attempt to be short and simple, we necessarily simplify the discussion. We phrase it in a way that is hopefully meaningful to someone with a z/OS background. If you want more detail, we recommend that you read *Enterprise JavaBeans for z/OS and OS/390 WebSphere Application Server V4.0*, SG24-6283.

To gain an understanding of Java, in general, there are numerous books on the market.

This barrier to understanding is also an issue the other way around. People who work in e-business often have little concept of what the z/OS platform is and can do, and why it is better than other NT and UNIX servers.

2.1 Object-oriented technology

One of the technologies that is most talked about today is object-oriented technology. Usually, it is discussed in the context of Java (and sometimes C++), but the concepts have been in existence since the sixties. Application servers, such as Lotus Notes, are written using it. But what is it? And what is an object?

Simply put, object-oriented technology is a different way to design and code applications. Let us compare it with the procedural model, with which you are probably familiar. We will do so in the context of a simple banking transaction, where we wish to transfer money from one customer's account to an account owned by a different customer.

2.1.1 The procedural model

In the procedural model, our application design and coding would focus on what we have to do to make this happen. We might come up with the steps shown in Figure 2-1.

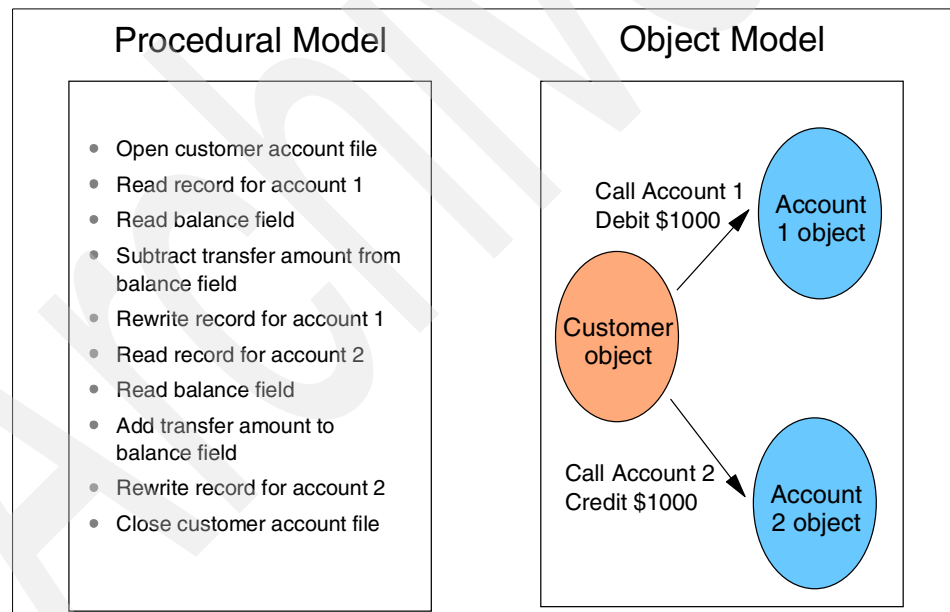


Figure 2-1 Procedural and object models

.As you can see, the programming model is a list of actions (a procedure) that will achieve the desired results.

As we developed applications according to this model, we quickly realized some limitations:

- ▶ It is possible to produce very long programs with complex logic. These are very difficult to write and maintain.
- ▶ We could end up writing the same logic in lots of different programs, as there was no way to share it.
- ▶ If we changed the business model (for example we do not want to allow a customer's account to be debited unless there are sufficient funds and at least ten dollars extra in it), we may have to change many programs.

The result was that we moved to modular programming techniques, writing smaller programs with specific functions, and linking them together to create the whole application. We tried to design the interfaces between the modules to simplify the design and coding.

2.1.2 The object-oriented model

Now we come to object-oriented technology. It tries to accomplish the same goals as modular programming, but it starts from a completely different paradigm. It looks at the objects you have in your business, for example customers, and bases the application coding around those objects.

If we go back to our previous example, we would have an *object* called Customer 1 and another object called Customer 2. These objects are very similar (they are both customers) and so we define a *class* of customers that defines what a customer object is. We would also probably have an object called an account, and in our coding each customer object could be linked to one or more account objects.

In this technology, an object can have associated with it:

- ▶ **Properties**

In our example, each account would have a property called a balance. When we go to that account, we can check what the balance is.

- ▶ **Methods**

A method describes something that we can do to the object. For example, with an account object we could change the balance state by crediting or debiting money. We could also close an account or open a new account.

We associate the properties and methods with the object, and this is the object that we then work with. Think of it as the module in the procedural model.

So now our application design might flow like this:

1. The customer object calls the object for Account 1 with the debit method and passes the transaction amount.
2. At this point, the account 1 object decreases the balance by the transaction amount. This is built into the coding for the account class.
3. The customer object calls the object for Account 2 with the credit method and passes the transaction amount.
4. At this point, the account2 object increases the balance by the transaction amount. This is also built into the coding for the account class.

2.1.3 Object program source code

If you look at a program written in an object-oriented manner, you would see a different method of programming. For example, you may refer to the account balance by coding:

```
Account1.QueryBalance
```

Note that:

- ▶ Account1 is the name of the object for Customer 1's account (it could be an account number for example).
- ▶ QueryBalance is the method that we are requesting from that object.

The object processes that method and returns the result, the account balance. We don't expect you to be able to understand such coding unless you are an application developer using object techniques.

However, you will see that applications are basically made up from programs, and programs contain coding statements. They are known by different names in the object world, but they are not dissimilar to what you are used to.

2.1.4 Advantages of object-oriented technology

Object-oriented technology is a completely different way of looking at your business (in terms of objects, properties, and methods), and designing your applications to work accordingly. By doing so, you will have applications that more closely match the way you do business.

You will find application maintenance easier, since the idea of a customer is unlikely to change fundamentally, and adding a new property or method for a customer only needs to take place in one object.

You will be able to respond to changes in your business more quickly. Once you have an object-oriented design, you will want to code your applications with an object-oriented programming language.

The key is the way you look at your business, not the language you choose.

2.2 The Java language

Why all the fuss about Java? The value of Java is seen in the language and the execution environment implemented by the Java Virtual Machine.

Java is an object-oriented language, so it allows you to easily develop applications according to the object model. The language was developed by Sun, but it has become an industry standard supported by most of the major information technology suppliers, including IBM. These companies help develop the Java standards by proposing new standards to extend the language facilities.

Java is similar to, and was based on, C and C++, so developers in those languages will be able to learn Java quite easily.

However, Java improves on those languages and, in particular, it does not allow you to manipulate pointers to memory. This is a major source of problems in C and C++, since incorrect pointers can cause you to overwrite storage that you should not.

Therefore, the Java language is more robust than its predecessors. However, if Java was just another language, it would not have received the attention it has.

2.3 JavaBeans

So far, we have discussed object technology and Java. The next question is, what is a JavaBean?

2.3.1 Characteristics

When you build an object-oriented application, you want those objects to be reusable. To do that, the objects must have certain characteristics. A Java object becomes a JavaBean when it supports a key set of interfaces:

- **Introspection:** This allows a visual programming tool, such as VisualAge for Java, to dynamically analyze how a JavaBean works. This enables developers to understand a supplied JavaBean and then connect JavaBeans.

Beans that haven't "met" before can learn each other's properties dynamically and act accordingly.

- ▶ **Customization:** This allows developers to customize the appearance and behavior of the JavaBean using a "property sheet" provided by the Bean developer and the visual programming tool.
- ▶ **Events:** These provide a notification mechanism between JavaBeans to announce that something has happened or is about to happen.
- ▶ **Properties:** These are similar to attributes, but they add the ability to broadcast a notification when the property changes.
- ▶ **Persistence, or serialization:** This allows a JavaBean to be customized (for example changing the account balance in our earlier example) and then saved for later use. However, this refers to the JavaBean being saved in memory only.

This is shown in Figure 2-2.

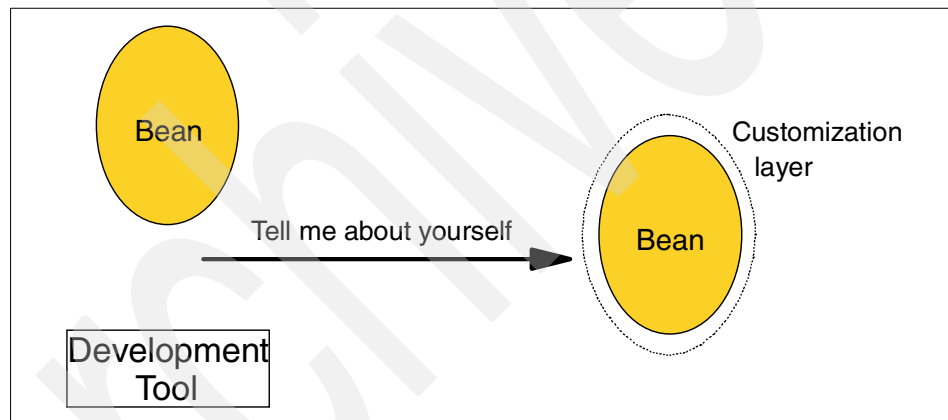


Figure 2-2 JavaBeans

These extensions to a standard object facilitate application assembly and reuse. By writing your Java code to the JavaBean standards, you gain additional benefits in application development. You can easily link together objects (or components) in different combinations to obtain different results.

To build a component with JavaBeans, you write Java language statements and include JavaBeans statements that describe user interface characteristics and events that trigger a Bean to communicate with other Beans.

Do not confuse JavaBeans with Enterprise JavaBeans although the names are similar. JavaBeans are Java objects written in a way that they can be reused and manipulated in Integrated Development Environments (IDE). For example a user interface “button” is a JavaBean that can be manipulated in an IDE and can be customized according to individual requirements without reprogramming (e.g., change the text on the button, change the size or layout).

Like Java, the JavaBeans specification is owned by Sun, but is widely implemented in the industry.

2.3.2 Does Java imply object-oriented

We said previously that Java is an object-oriented language. It provides the constructs to develop objects. However, Java is just a language, and therefore it does not force you to a particular application design.

You can develop applications designed with the procedural model in Java. In fact, if you come from a procedural background, that is always the temptation.

Just because you are using Java does not mean that you are using object-oriented design.

2.4 The execution environment: Java Virtual Machine

The second reason that Java has become so popular is its execution environment. Java code is compiled into bytecode, a representation of the program that uses 2 bytes for each character. The bytecode is stored in a file with an extension of .class or .jar and at runtime it is interpreted by a piece of software called a Java Virtual Machine (JVM). The JVM reads the bytecode and acts upon it. This is shown in Figure 2-3 on page 18.

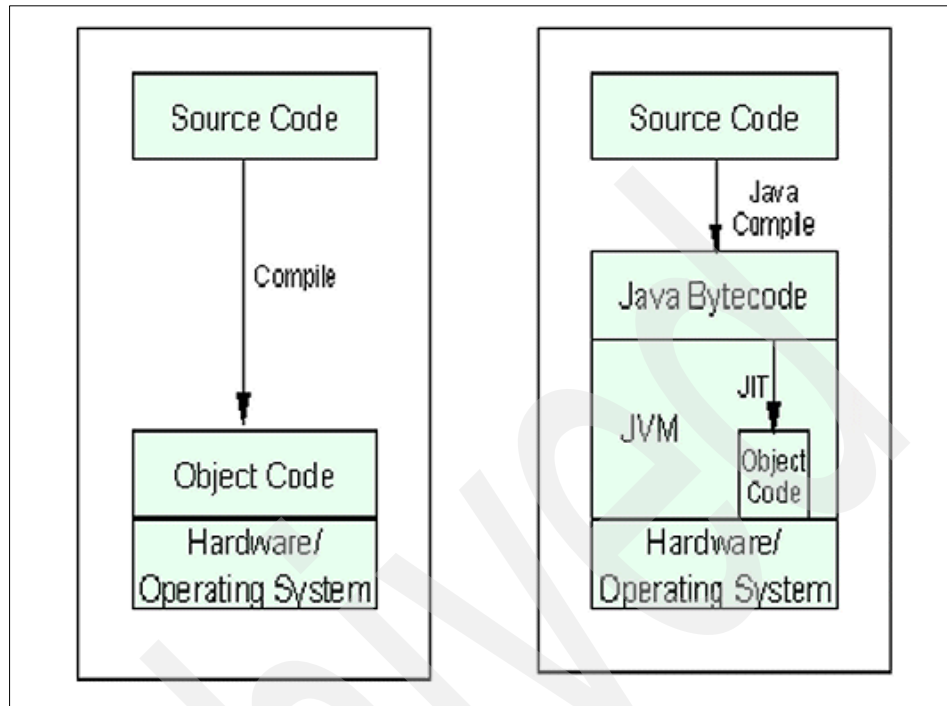


Figure 2-3 Java Virtual Machine

This means that Java code can be run on any machine that has implemented a JVM.

Since all the major platforms in the industry have a JVM, including z/OS, we now have a way to write code once, integrate it easily into other code by basing it on objects, and executing it on any platform.

There is no need to port or recompile code to another platform. The decision on where to run it can be made after the coding and testing is complete.

Java executing options

One issue with interpreting the Java code at runtime is that it takes significantly more processing power than running a piece of precompiled code. There are actually two ways to execute Java code on a platform:

- ▶ Read the Unicode, and interpret it as you go.
- ▶ Just before execution, you can compile the Unicode using a Just-In-Time (JIT) compiler. This increases the start time for the program, but reduces its run time. On z/OS, this is the default way of running Java.

Note: The High Performance Compiler for Java (HPCJ) is no longer available.

2.5 Java 2 Platform

The Java 2 Platform is the foundation on which all of the e-business infrastructure is based. At a high level, the Java Platform is comprised of the Java Standard Edition, Java Enterprise Edition, and for pervasive computing devices Java Micro Edition. For most Java applications the Java 2 Standard Edition (J2SE) will deliver the base function for deploying meaningful enterprise applications. However, when your company demands more sophisticated processing, the Java 2 Enterprise Edition (J2EE) Platform extends the J2SE specification by addressing distributed computing across an enterprise network. And finally, when you need limited function in hand-held type devices you probably want to consider using Java 2 Micro Edition (J2ME), which is a lightweight language with critical minimal language and runtime capabilities.

2.5.1 Java Standard Edition

The Java 2 Platform Standard Edition (J2SE) supports an environment that allows the deployment and development of network-centric applications that span a single user to multiple users across an enterprise environment. Java Development is performed using the Java 2 Software Development Kit (SDK) and the Java 2 Runtime Environment.

Java 2 Software Development Kit (SDK)

The Java 2 SDK Standard Edition includes the tools to develop Java applications. These tools make it possible to develop, test and run applications in a Java environment. In addition to the JRE, development tools such as compilers (javac) and debuggers are included in the kit.

Java Runtime Environment (JRE)

The Java Runtime Environment includes everything to execute Java applications: the Java Virtual Machine (JVM), the core classes for the Java Platform, and the supporting files. The core Java classes are illustrated in Figure 2-4 on page 20.

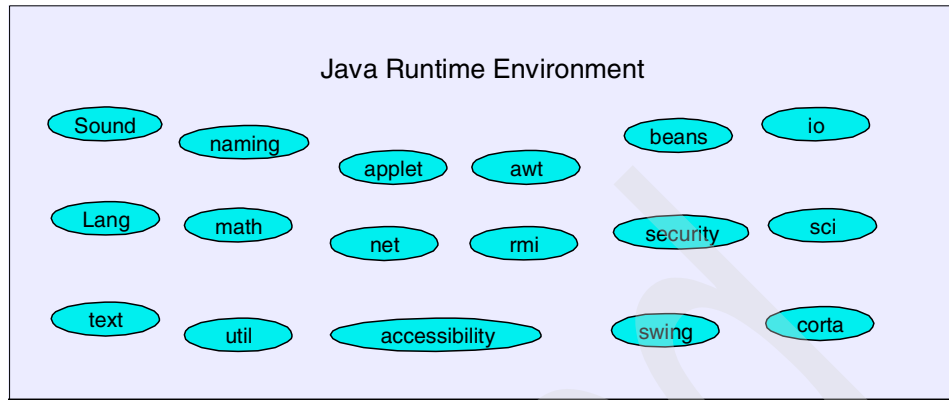


Figure 2-4 Java Platform core classes

These classes are developed for specific computers to make them Java compliant. These classes make it possible for programs written in Java to communicate with that computer's hardware, peripherals and other Java programs residing on this and other physical computer machines.

2.5.2 Java 2 Enterprise Edition

Java 2 Enterprise Edition (J2EE) is an architecture targeted towards multitiered applications. The multi-tiered applications of today are an evolution of the client/server processing model where the presentation and processing were done on one platform and datastores were on another. In most cases, as illustrated in Figure 2-5, business processing was performed inconsistently in databases and client applications.

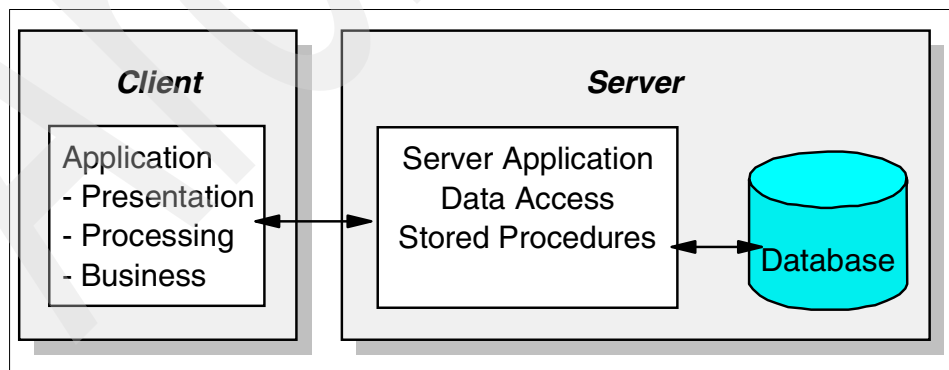


Figure 2-5 Client/server processing

As performance issues arose, a better way of distributing the workload was sought by systems designers and developers. It was recognized that by distributing the workload, the application would perform better than in the client/server configuration. Though this concept was not new, the Java Platform made it easier to develop disconnected software components that could process business logic and maintain data integrity. Figure 2-6 shows a 3-tiered application that has the client tier on a separate machine with the server running data access and business processing in separate components.

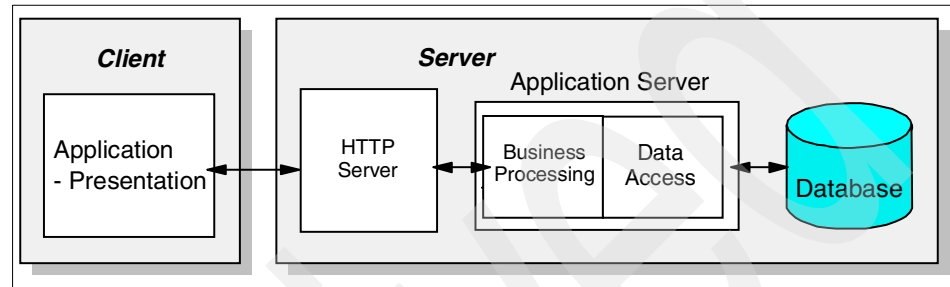


Figure 2-6 3-tiered application

As luck would have it, additional gains had been realized in the evolution of 3-tiered to multi-tiered applications. In multi-tiered applications, each processing unit would be responsible for its own existence as an object. This concept evolved into JavaServer Pages (JSP) to drive dynamic browser presentations, Servlets to handle processing, and Enterprise Java Bean or EJB. In Figure 2-7, we see the evolution of the new components and host integration as well.

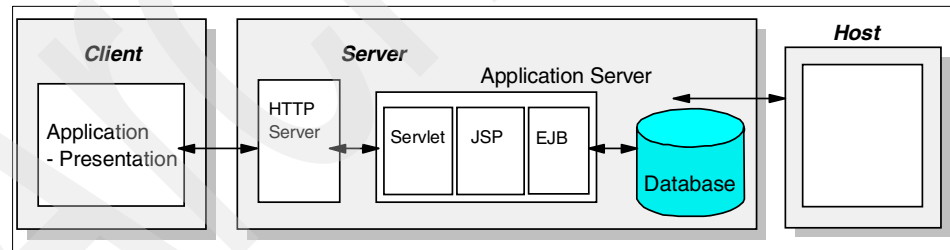


Figure 2-7 Multi-tiered applications

As the complexity of the systems grew, the Java community agreed that a standard needed to be developed to make it easier to develop these applications with distributed components. In loose terms, this is how the J2EE specifications came about.

2.6 J2EE applications

A J2EE product must support application components that perform the bulk of business processing. Each type of component must execute in a container that provides the services used by the application to process in a Java environment. Application components can be packaged in JAR files and include a deployment descriptor, giving platform-independent information to the container that manages them.

Parts of the J2EE specifications describe the use of containers to represent application components that meet specific needs of the target environment. These containers include application components, service components and communications. The application components were the responsibility of the application developers who used the services that enabled the interaction of these components in a distributed environment. Figure 2-8 illustrates the relationship between the components, containers, services, and the Java Platform.

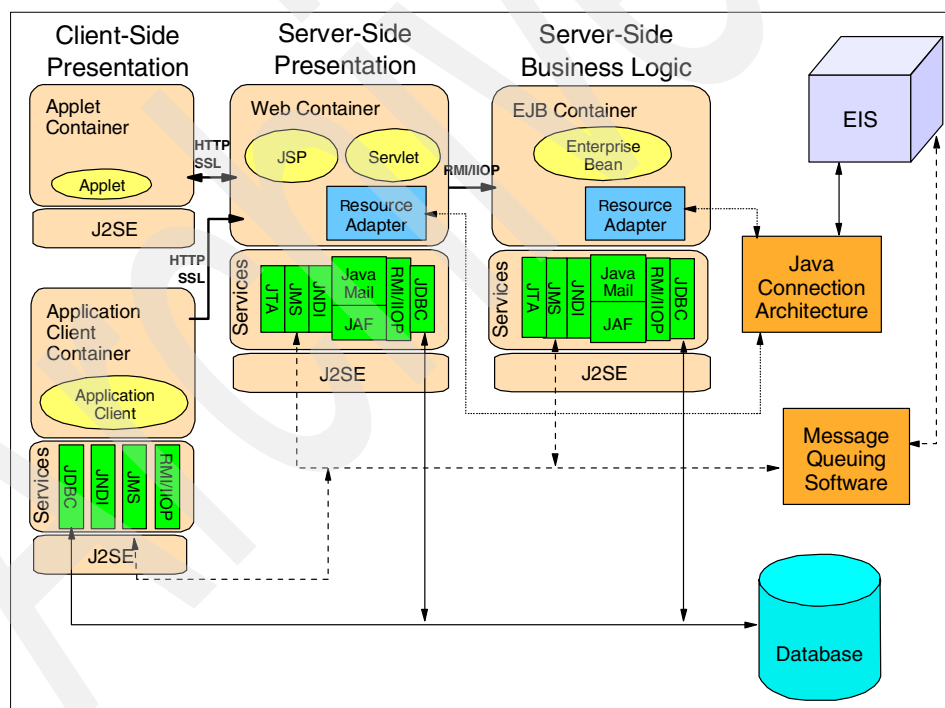


Figure 2-8 J2EE application architecture

J2EE is a multi-tier architecture:

- ▶ Client tier

This is a browser or a Java client, or any other device (e.g. PDA) doing the presentation or *View* of information to the application client.

- ▶ Web component tier

This tier gets client requests (HTTP,HTTPS), analyses the requests and decides to respond with a file (HTML, images) or calls a program (servlet) to do some part of the server-side processing requested by the client. Generally the servlet acts as the *Controller* (controls the whole application flow), then calls a JavaServer Page (JSP) to dynamically generate the HTML response (the presentation or View) to be sent back to the client.

- ▶ EJB tier

The Enterprise JavaBean tier is responsible for the application's business logic or the business *Model*. EJBs represent business logic usually as transactions, such as transfer of money between accounts or ordering a product. EJBs also represent data, so the database is mapped to EJBs for consistent services across the architecture.

- ▶ EIS or database tier

This tier represents either the database itself (e.g. DB2) or an EIS system that executes legacy components of the application. Such examples include the CICS or IMS transaction managers, which can be called from the EJB tier through connectors.

The J2EE architecture enables the realization of the Model-View-Controller architecture. This programming paradigm separates the different layers of the application, namely the presentation of information (View), the application control flow (Controller) and the business logic and data (model). This paradigm makes it easier to develop and maintain the application code.

2.6.1 J2EE standard services

J2EE components may use a set of standard services to interact with each other. Here is a list of these services with a short explanation of the functionality they provide to the components.

HTTP and HTTPS

The HTTP protocol is the protocol used across the Internet to get objects (pages, images, and so on) from remote hosts. HTTP messages are client requests and responses from the server.

A J2EE Platform must implement this protocol—the client-side API is defined by the `java.net` package, which provides the classes for implementing networking applications. The server-side API is defined by the `Servlet` and `JSP` interfaces.

For HTTP layered over the Secure Socket Layer protocol (HTTPS), the same client and server APIs as HTTP are required to support this protocol.

Java Naming and Directory Interface (JNDI)

This API allows the J2EE components to look up other remote objects that they may need to access. The JNDI API was designed to standardize access to a variety of naming and directory interfaces and has two parts:

- ▶ An application-level interface
These APIs are used by the application components to access naming and directory services.
- ▶ A service provider interface
This part of the API is used to attach a provider of a naming and directory service to the J2EE Platform.

Changes to the JNDI interface have been realized and developers must be aware of the changes as their applications are migrated to subsequent versions of the Java Platform. For example, In EJB1.0, the JNDI lookup to find an application named `piggyapp` located in directory `root/com/ibm` was similar to:

```
initialContext.lookup("com/ibm/piggyapp");
```

In EJB1.1, using JNDI, the lookup to find the application becomes:

```
initialContext.lookup("java:comp/env/ejb/com/ibm/piggyapp");
```

Java Database Connectivity (JDBC)

JDBC is the standard API that enables connectivity with database systems. An application's components can use JDBC to handle data from relational databases and other repositories.

The JDBC API can also be divided into two parts: an application-level interface used by the components for accessing databases and a service provider interface to attach a provider to the platform.

Java Messaging Service (JMS)

This API defines a standard mechanism for the components to send and receive messages. WebSphere for z/OS supports the use of the JMS API by the following types of J2EE application components: `Servlets`, `JavaServer Pages (JSPs)`, and `Enterprise JavaBeans`. It enables the use of enterprise messaging systems (IBM MQSeries, for example).

JavaMail and JavaBeans Activation Framework (JAF)

The JavaMail API allows an application to send e-mail notifications. It is divided into two parts: an application-level interface used by the application to send mail, and a service provider interface to attach a provider to the platform.

JavaMail includes the JavaBeans Activation Framework API (JAF).

Java Transaction API (JTA and JTS)

The Java Transaction API (JTA) provides a way for J2EE components and clients to manage their own transactions. It also allows multiple components to participate in a single transaction. JTA is the API, and the Java Transaction Service (JTS) is the implementation of this service.

The application-level interface of the API is used by the container and the application components to demarcate transaction boundaries. The second part of the API defines an interface between the transaction manager and the resource manager used.

Remote Method Invocation — Internet Inter-ORB Protocol

This API supports the use of the RMI APIs with the IIOP protocol. The Remote Method Invocation (RMI) is a technology that allows an object running in one Java Virtual Machine to access another object running in a different Java Virtual Machine.

The Internet Inter-ORB (Object Request Broker) Protocol (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 allows J2EE applications to access CORBA services defined by components living outside the J2EE product. This API will also be used to access Enterprise JavaBeans components in an application.

Java IDL

IDL stands for Interface Definition Language. This API allows J2EE application components to invoke external CORBA objects using the IIOP protocol. These CORBA objects are completely independent of the J2EE application (they may not be Java objects) and may be used by other applications.

2.6.2 J2EE containers

In the J2EE architecture the application components are executed in *containers*. The containers provide execution environment with transactional, security, availability and persistence services, just to name a few, so the application developer need not deal with these issues, but focus on business logic instead.

There must be one container for each application client type in a J2EE application. The reason for having a container between the J2EE application components and the set of services is to provide a federated view of the APIs for the application components.

There are four types of containers in the J2EE architecture:

- ▶ Applet containers, which provide runtime for applets in a browser
- ▶ Application Client containers, which provide runtime for J2EE application clients running on a client device (e.g. PC)
- ▶ Web Component containers, which provide the runtime environment for servlets and JSPs
- ▶ EJB containers, which provide the execution environment for Enterprise JavaBeans

Application client container

Application clients are Java programs, such as a Graphical User Interface (GUI) that executes on a desktop computer. They have access to all of the facilities of J2EE when they run in a J2EE client container. This container supports application client components. It must provide access to the set of services required by J2EE but is not required to manage transactions. Application clients have access to the Java API and are packaged in a JAR file.

An application client is packaged in a JAR file and is typically used to run the presentation logic of an application. Stand-alone applications or programs that do not fit in any of the other categories can be placed in this container.

Applet container

An applet is a component that typically executes in a Web browser. It is a Java class that can also run in a variety of other applications or devices. The applet must be loaded, initialized and run by a Web browser; it can be used to process presentation logic, and it provides a powerful user interface for J2EE applications. However, simple HTML pages may also be used to process presentation logic as well.

Applets embedded in an HTML page are deployed and managed on a J2EE server, although they run on the client machine. They are considered as belonging to the HTML page and an HTML page is managed by a J2EE Server.

The applet container includes support for the applet programming model. Typically a J2SE (Java 2 Platform, Standard Edition) 1.2 compatible applet execution environment acts as a container. The Java plug-in may be added to the browser to obtain such a container. Applets communicate over HTTP if they run in a browser, but they can also communicate using serialized objects.

Web container

The Web container is used to run servlets and JSPs (referred to as “Web components”). It is responsible for providing a runtime environment with a set of services.

Important: A Web container is not a Web server, but part of a Web Application Server!

A Web container provides the runtime environment for Web components (a Web component is a servlet or a JSP) and services for these components: security, concurrency, life cycle management, transaction, deployment, etc.

By “Web server” we usually mean “HTTP server”.

EJB container

The EJB container on the application server is where all deployed EJBs reside. The EJB specification defines a container contract for remote access to the bean, as well as specific container interaction, which may be implemented differently by various system vendors. In addition to the container contract, the EJB specification also defines EJB composition and deployment requirements. Each of these functional characteristics is supported by one or more APIs in J2EE.

The Enterprise Java Bean architecture described in the EJB specification defines the contract enabling Java code written by multiple vendors to interoperate in a distributed programming environment. The reusable code components are referred to as Enterprise Java Beans. EJBs add the scalability and reliability aspects inherent to industrial strength server applications.

2.6.3 Application layers

As applications are developed, the components can be arranged in components that have specific functions for the applications. Figure 2-9 shows an overview of the application components in their respective containers along with the types of logic they usually handle in an enterprise application. We can see the evolution of the J2EE architecture from client/server to multi-tiers that have specific responsibilities divided into layers of presentation, business logic and data stores (databases and EIS).

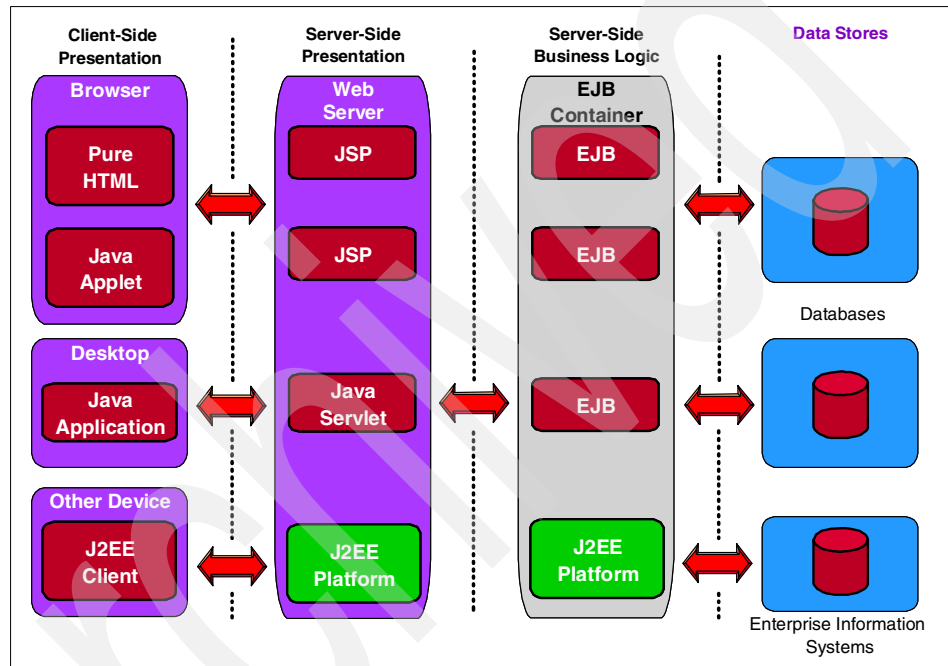


Figure 2-9 Application layers

Presentation layer

The presentation layer is responsible for formatting and displaying graphics and text to the end user of the application. Given today's technologies, the presentation layer can be realized using different medium ranging from a desktop workstation to pervasive computing devices. The containers that make up the presentation layer are described in 2.6.2, "J2EE containers" on page 26. Depending on the needs of the application, and the requirements of the business, the presentation layer can make use of a variety of technologies ranging from browsers to Java applications.

Business logic layer

The Business logic layer is where the integration of the business model and the technology comes together. This layer is responsible for representing the business process, business rules, and interaction with other systems to resolve any business processing issues for the target application. The containers that perform these functions are defined in “J2EE containers” on page 26.

Data stores

Data stores represent enterprise data that is used by the J2EE application. In traditional host applications, data stores were IMS and CICS systems that stored data in a variety of media controlled by their respective environments. With the evolution of databases and stored procedures, some business logic has been written in these environments making it hard to create a true break in data store, business process and business logic. As a result, strategies in Web applications have evolved that allow the cohabitation of business logic in some data store environments and the business logic layer. In general, navigation and new business logic is incorporated in the business logic layer with minimal introduction of new business logic in the data stores layer.

2.6.4 Servlets

A servlet is basically a Java application written according to the servlet programming model. The servlet code is executed on the application server.

Typically, a servlet is used to get requests from clients, build a response to be shipped back to the browser in HTML format. However, a servlet can do anything within the boundaries of the JDK functionality, including using the Java Database Connection (JDBC) to access databases and talking to Enterprise Java Beans (EJBs). See 2.6.6, “Enterprise JavaBeans (EJBs)” on page 30 for more information concerning EJBs.

The servlet functionality allows you to separate the formatting of the response from the actual processing logic in the EJBs. This should be one of your goals when using the object oriented programming methodology.

It allows you to make changes to formatting logic, in response to changes to your internet presence, without impacting your business logic. Programmers whose responsibility is the graphical end user interfaces can concentrate on what they best do.

In the mean time, those programmers responsible for the business logic can limit their changes to just the essential programming logic and not have to worry about how the data is going to be formatted.

Servlets execute in the Web containers.

2.6.5 JavaServer Pages (JSPs)

JSPs are HTML pages with special tags used to insert Java code. Eventually they wind up being compiled into a servlet, but you don't have to worry about that.

JavaServer Pages are a way to create HTML pages more dynamically. JSPs can almost be written as normal HTML pages, but they are converted into a server-side Java program (servlet) at deployment time. So, the HTML is actually built on the server and then sent to the browser. This mechanism, combined with the possibility to mix in Java code or invoke JavaBeans or other Java servlets, makes a JSP so dynamic.

The point of using JSPs, as opposed to simple servlets, is that they have a more strict separation between output formatting (HTML) and application logic (Java) used to generate the formatting and also talk to EJBs.

JSPs have to be written according to the JSP specification and also execute in the Web containers.

2.6.6 Enterprise JavaBeans (EJBs)

The Enterprise JavaBeans (EJB) programming model has become a very attractive application programming model largely due to the perceived benefits of being an open industry standard, which is based on the notion of building highly portable and extensible object-oriented software components. The ability to develop and test atomic software components rapidly on any platform, and then deploy them unchanged on yet a different platform, is the goal of development teams. Such is the promise of component software technology.

As you may guess from the name, Enterprise JavaBeans (EJBs) are an extension of JavaBeans. To explain why they came about, let us consider what happens when we try to write a significant application (what you may call an enterprise application) from JavaBeans.

The limitations of JavaBeans

The JavaBeans container provides a limited set of facilities. If you need any of the following facilities, you need to write code to implement them in your application:

- ▶ Managing the *life cycle* of an object, such as process allocation, thread management, object activation, and object destruction.
- ▶ Saving and restoring a conversational *object state* between method calls.
- ▶ *Security* functions such as authenticating users and checking authorization levels.

- ▶ Defining the scope of a *transaction* between different transaction managers, and making sure that all the managers start, commit, or rollback transactions together.
- ▶ Explicitly retrieve and store *persistent* object data from a database.

This is shown in Figure 2-10.

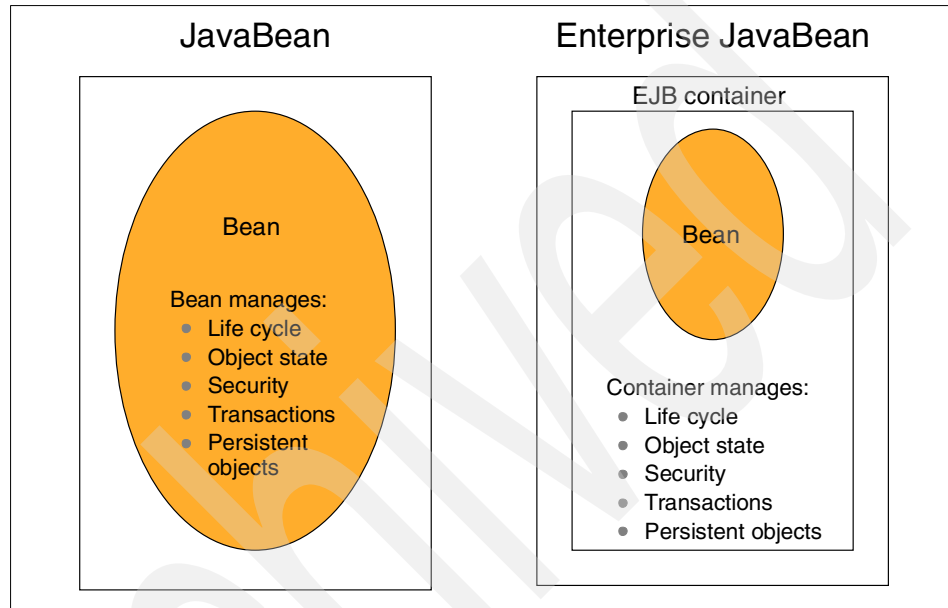


Figure 2-10 Enterprise JavaBeans

You will immediately notice that a transaction manager such as CICS and IMS on z/OS has been providing these functions for many years. It makes no sense to require every application to code their own versions of these. Rather, they should be provided as standard facilities by the application server.

The value of Enterprise Java Beans (EJBs)

This is where EJBs come in. EJB is a specification for a set of services provided by an EJB server. Since these services are fundamental to most applications, the idea is that the EJB specification will become the standard for the Java programmer. In fact, it simplifies the coding that they need to do. Instead of coding mundane things such as reading and writing to files, they can concentrate on coding the business logic.

Other benefits of EJBs are:

- ▶ Java applications can be written to be industrial strength, using the types of facilities that CICS and IMS developers have used for many years.
- ▶ Applications written to the EJB specification can be easily combined and run on any platform. You do not need to know how to access data or manage threads on a specific platform, or how to connect to a particular database or transaction manager.
- ▶ You can easily separate the roles in developing and deploying applications:
 - The Enterprise JavaBean developer (or Application Component Provider) creates the Beans, defines their interfaces, descriptors and properties, packages them in a jar file and provides or sells them to the Application Assembler.
 - The Application Assembler takes Beans from the component developer and combines them into a J2EE application. The application is put into an EAR file.
 - The Bean deployer installs the EJBs in containers to run the application on a target machine.
 - The container provider provides the runtime environment to support EJBs and the tools to manage the Beans.
- ▶ Each person can work independently of the others, and there should be no (or minimal) dependencies between them.

Types of beans

It is important to remember that there are two basic types of EJBs: Session Beans and Entity Beans.

▶ Session Beans

Session Beans are created for each client connection and destroyed afterwards. Session beans provide the business logic of applications, e.g., money transfer.

▶ Entity Beans

Entity Beans are meant to represent something in the system (mostly data) that has to be there all the time (persistent). Figure 2-11 on page 33 shows the relationship between Session Beans and Entity Beans.

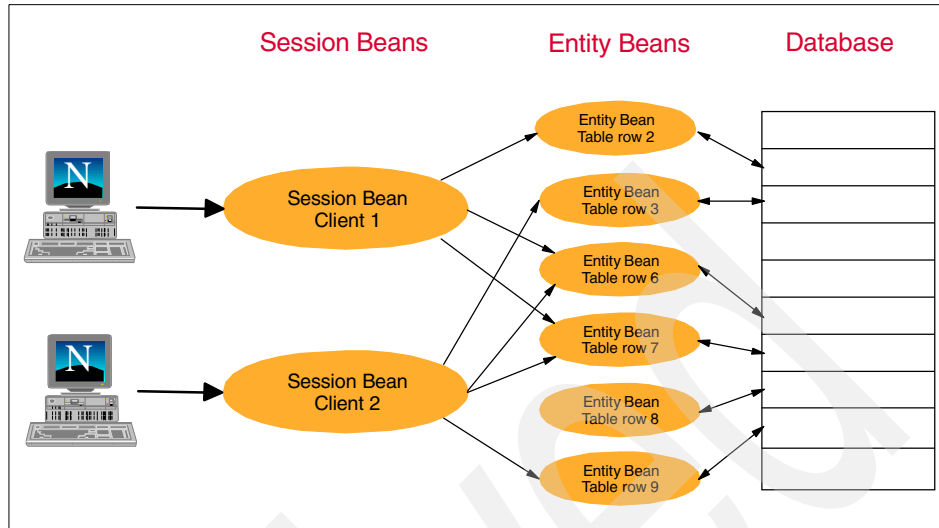


Figure 2-11 Session Beans and Entity Beans

IBM provides two application servers on WebSphere Application Server for z/OS, which includes a J2EE application server and a Managed Object Framework (MOFW) application server which is the name of the CORBA application server.

For more information on the EJB specification, see the Web site at:

<http://java.sun.com/products/ejb/index.html>

2.7 Deployment

The J2EE specification gives instructions for packaging and deploying components to their runtime environment. It introduces the concept of deployment descriptors and specifies the content that can be grouped together. Figure 2-12 on page 34 illustrates the deployment descriptors described below.

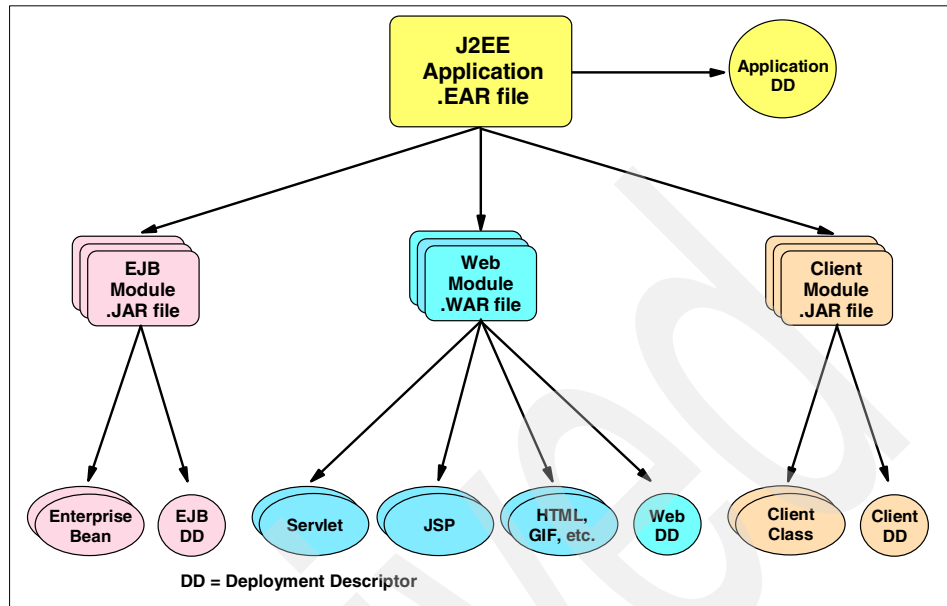


Figure 2-12 Deployment descriptors

2.7.1 Deployment and deployment descriptors

Application components can be packaged so that they can be deployed in many environments. Application components can be packaged independently, whereas elements of an application client are packaged in a single JAR file. Web components may be packaged in a WAR (Web Archive) file with other Web resources. Enterprise Beans can be packaged together as well. These packages will be used for deployment in the container dedicated to each application component.

Application components can also be packaged together to construct a J2EE application, that is to say, an application containing Enterprise Beans, Web components and eventually, an application client. The intent of the application is to provide for the possibility of deployment to several operating environments.

The deployment task generally consists of three steps: installation of the components on the server (or container), followed by configuration and execution of the application. Deployment descriptors assist in this deployment task.

A deployment descriptor is an XML file describing how to assemble and deploy a unit into a specific environment. It is provided with the application and specifies the required external resources, as well as the security requirements and environment parameters. The purpose of a deployment descriptor is to enable developers to create reusable components. These components can be customized to run in several environments.

2.7.2 The J2EE packages and their deployment descriptor

Figure 2-12 on page 34 shows the different packages that can be built for deployment; each of these contains its own deployment descriptor (DD). JAR files are the standard way of packaging modules and applications, but other formats may be used.

There are four deployment modules:

- ▶ EJB

EJBs are packaged in a J2EE module (a JAR file, for example). The EJB specification defines the DTD (Document Type Definition) of the deployment descriptor associated with this module.

- ▶ Web application

A Web application also represents a J2EE module, containing servlets, JSPs and other resources. The Java servlet specification defines the DTD of the deployment descriptor for a Web application.

- ▶ Application client

An application client represents a J2EE module. It is packaged in a JAR file and contains a deployment descriptor.

- ▶ J2EE application

A J2EE application is a set of J2EE modules with a J2EE deployment descriptor. It is packaged using the JAR file format into an Enterprise ARchive (EAR) file. This file has the .ear extension. It contains a deployment descriptor and one or more J2EE modules, each of which contains its own deployment descriptor.

2.8 CORBA Business Objects

Now that we've discussed J2EE, it's time to move on to a discussion of CORBA Business Objects. In the world of CORBA you are not limited to using Java as a programming language. Both C and C++ are also supported. You can write platform-specific Business Objects that can communicate with other Business Objects running on different platforms and written in another language.

While this does not live up to the Java promise of “write your business logic once” and then “run it anywhere”, it is nevertheless a very powerful concept.

CORBA applications can take advantage of a given platform’s strengths and exploit them. It can communicate with other business objects via a common communications path.

2.9 Extensible Markup Language (XML)

One of the main concerns in e-business is universality. We want all that we do in e-business to be universal, that is, usable and accessible from anywhere. The Internet has provided a sort of universal communication (it may not be perfect, but is certainly accessible to almost anybody). Browsers have provided a universal user interface with the standard usage of HTML, although Java has provided a universal language, a truly “write once, execute everywhere”.

But what about the data that goes from one place to another in this universal environment? The truth is that everyone stores data the way they need it, which means that if you transfer it from one place to another, some sort of “reformatting” always has to take place. What XML gives you is a universal data format for text files that can be shared and “understood” by anybody.

For example, imagine the following situation. Many libraries offer their catalogs over the Web. Usually there is a simple Web form where you enter a title, name or subject, and you are presented with some search results. If you wanted to search several libraries, you would need to go to each of their Web sites.

It would be useful and convenient if there was a single Web page that could search many libraries. To build that today would require extracting the data (title, author, ISBN, and so forth) from each search results page. However, each search results page is formatted differently, and the data is mixed in with presentation information. To collate the results of many searches would require complex programming for each library’s Web site, separating the data from the presentation.

Suppose instead that there was a single format for returning search results from libraries. Let us call it A Book Catalog Markup Language or ABCML. ABCML would define tags for the author, title, and so on, thus making it easy for a computer to extract the data. Building the meta-catalog suddenly becomes easier. ABCML would also help the libraries, because they could re-use each other’s software. Also, when a new book came in, the publisher could provide the book’s catalog information in ABCML, and save the librarian the effort of typing it into the catalog.

The main benefits of using XML are:

- ▶ Re-use of data
- ▶ Separation of data and display

With the rise of pervasive computing, each device will have a particular way to display data. If they all understand XML, this is much easier.
- ▶ Extensibility

New types of information can be created and added in old documents without affecting the old data or format.
- ▶ Semantic information

This is probably one of the major possibilities of XML, since it allows data to be interpreted with a meaning, not only as a set of characters.

However, although XML is very useful in an Internet environment, it is not only about the Web. You can use XML wherever you wish inside your business and exploit all its advantages without going to the Web.

XML data format

What does an XML format look like? Take this data, for example, which could be in any of your databases:

X-One P185/70R15Michelin 620 A B265.78

In this format, it is difficult to understand what it is all about. Now, in XML format, it could look like this:

Example 2-1

```
<?XML version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TIRE SYSTEM "http://www.aaa.com/tirespec.dtd"/>
<TIRE MODEL="X-One" SIZE="P185/70R15">
<MANUFACTURER NAME="Michelin" HREF="http://www.michelin.com"/>
<DESCRIPTION>The all-season T speed-rated touring tire.</DESCRIPTION>
<UTQC TREADWEAR="620" TRACTION="A" TEMPERATURE="B"/>
<PRICE UNIT="Dollars">265.78</PRICE>
</TIRE>
```

Now this is a much more understandable way of formatting the data, and anybody who knows about tires can read it. You might also think that it looks like HTML because of the use of tags. However, HTML has many disadvantages when compared to XML. HTML has these characteristics:

- ▶ It is all form and no substance.
- ▶ It is not extensible.
- ▶ It is display-centric.

- ▶ It is not directly reusable.
- ▶ It provides only one view of data.
- ▶ It has little or no semantic structure.
- ▶ It is too simple.

But would a browser understand an XML file like the one in the example? The answer is no. Browsers only understand HTML. However, XML is not only the formatting structure. There are three other (optional) pieces to XML:

- ▶ Document Type Definition files (DTD)

DTDs specify the logical structure of the XML formatted file; it defines a relationship between the document elements and their attributes. It enables an XML parser to validate that the file is well formatted. DTDs can be public, private, industry-specific or customer-built. This is what makes sure that you can understand an XML file when you get it.
- ▶ Extensible Style Language (XSL)

XSL is just a template that can be used (using the XML file and the DTD file) to create an HTML version of the document so that a browser can display it. But this is only one use of it. You can create another XML file instead of an HTML one.
- ▶ Extensible Link Language (XLL)

XLL defines ways of using links to describe relationships between data, including one-to-many and indirection (which isolates end users from the familiar “link not found” messages when a resource is relocated).

Uses for XML

As a technology, XML will be showing up in many different places, such as:

- ▶ VisualTeam Connection as a metadata language
- ▶ Component Broker or WebSphere Application Server Enterprise Edition for object persistence and meta data
- ▶ Tivoli as metadata for systems management
- ▶ WebSphere Application Server for server-side programming
- ▶ DB2 for improved search and retrieval, document and data creation, and for exchanging database schema.

For example, do you want to build a subset of a relational database for local use? Just issue a DB2 select query from your Web browser, and ask for the data to be returned in XML schema. The browser detects the arrival of database XML schema and calls a plug-in that automatically builds a table in the database with all the correct column names and definitions and stores the data. It couldn't be simpler!

2.9.1 XML Toolkit for z/OS and OS/390

In November 1999, IBM announced that it was donating its Java (XML4J) and C++ (XML4C) parser technologies to the Apache XML project. The Apache XML project team renamed the parser technologies “Xerces.” Information about the Apache XML project is documented at their Web site at:

<http://xml.apache.org/index.html>

The XML parsers in the XML Toolkit for z/OS and OS/390 are based on the Apache Xerces code base. The Java parser is based on the Apache version 1.2.1 code base, while the C++ parser is based on Apache version 1.3.0 and ICU 1.6 (International Components for Unicode).

The toolkit includes public APIs and experimental APIs. Public APIs are those that have reached W3C recommendation status. Public APIs are considered stable, meaning that their interface is not expected to change. Fixes will be made available for Severity 1 or 2 problems with public APIs.

Experimental APIs are those that have not reached W3C recommendation status. They are subject to change and are not supported. Consequently, fixes may not be made available for these experimental APIs.

The XML Toolkit for z/OS and OS/390 V1R2 includes:

- ▶ XML Parser for z/OS and OS/390, Java Edition
- ▶ XML Parser for z/OS and OS/390, C++ Edition

Each parser included in the toolkit includes public and experimental APIs, as follows:

- ▶ Public APIs
 - DOM Level 1
 - SAX Level1
- ▶ Experimental APIs
 - DOM Level 2
 - SAX Level 2
 - Java Parser partial April 7 W3C Schema specification

In October 2001, the XML Toolkit for z/OS and OS/390 Version 1 Release 3 has been announced. This new release provides updates to the XML Java and C++ parsers, based on Apache Xerces-C 1.5.0 and Xerces-J 1.4.2, including SAX 2.0 and DOM 2.0 API support as well as Namespace, Schema and JAXP support. The release also includes Java and C++ XSLT Processor support, called LotusXSL, based on the Apache Xalan_C 1.2 and Xalan_J 2.2 processors.

For more information, see the XML Toolkit Web page at:

<http://www.ibm.com/servers/eservers/zseries/software/xml>

2.9.2 XML Parser for z/OS and OS/390, Java Edition

The z/OS and OS/390 technologies that can be integrated with the Java XML parser are listed in Table 2-1.

Table 2-1 Java XML Parser operating environments

Parser	CICS	WebSphere Application Server	IMS	Batch
Java Parser	Yes	Yes	Yes	Yes

CICS TS

There are two ways to use XML with CICS:

1. The XML Java parser can be used with CICS TS version 1.3 and above. The Java program must run under the Java Virtual Machine (JVM). Each CICS transaction running under a JVM runs on a separate TCB.

Java programs within CICS using the Java XML APIs must run in a JVM in order to prevent operating system waits from affecting other CICS transactions.

2. The XML Java parser can be used in a Java program that runs outside of the CICS environment; it converts the parsed data into a COMMAREA, and communicates with CICS.

WebSphere Application Server

There are no restrictions when using XML Java parser APIs with WebSphere Application Server. However, WebSphere Application Server also provides its own set of XML Document Structure Services, which contains an XML parser.

IMS Transaction Manager

There are two ways to use XML with IMS:

1. For existing IMS transactions, parsing and transformation servlets convert an XML stream to and from an IMS transaction message data structure.
2. New or modified IMS applications can use the Java parser APIs. Java IMS application programs need to be compiled using the High Performance Compiler option in the VisualAge for Java. The reason that Java programs within IMS must be compiled with the High Performance Compiler option in

the VisualAge for Java is because IMS does not use JVMs. Instead, each IMS transaction runs under its own TCB.

Batch

Batch Java programs that invoke the XML Java parser must run in a JVM.

Archived



The components of e-Business

The z/OS Operating System provides a robust environment in which you can enable a successful Web presence today.

You can build on what you have and Web-enable existing applications. You can tie into your backend systems, build new Java applications, and use the WebSphere family of products to run them.

3.1 Introduction

In order to develop and run an e-business on z/OS, there are several major components or environments that you need to be concerned about. Each of these environments contains several subcomponents, each of which plays an important part in a successful e-business.

The z/OS environment

- TCP/IP
- Unix Systems Services
- HFS
- LDAP/DB2
- WLM
- Security/SSL
- RRS/LOGGER

The WebSphere Application Server environment

- HTTP Server (although the IBM HTTP Server is not part of WebSphere Application Server it is considered part of the Web-serving environment)
- WebSphere Plugin Environment (the plugin runs in the HTTP Server)
- HTTP Protocol Handler
- Application Servers
 - J2EE
 - MOFW/CORBA
 - Base Environment

Connectors to backend systems

- CICS
- IMS
- DB2

The application development environment

- WebSphere Studio
- Visual Age for JAVA
- WebSphere Studio Application Developer

3.2 z/OS environment

The z/OS environment provides the foundation and services needed for your e-business to operate. It also provides the traditional environment of the backend databases and transactional servers that your e-business will connect to.

3.2.1 Supporting components

The following supporting components are an integral part of the z/OS environment. They represent mature technologies that have evolved to handle IBM customers' business needs.

The use of these z/OS components by the WebSphere Application Server provides a stable platform, with built-in recovery and security, for both traditional and e-business environments.

TCP/IP	Communications vehicle
Unix Systems Services	Platform support
HFS	File support
LDAP/DB2	Directory services
WLM	Work Load Management
Security/SSL	Security
RRS/LOGGER	Recovery

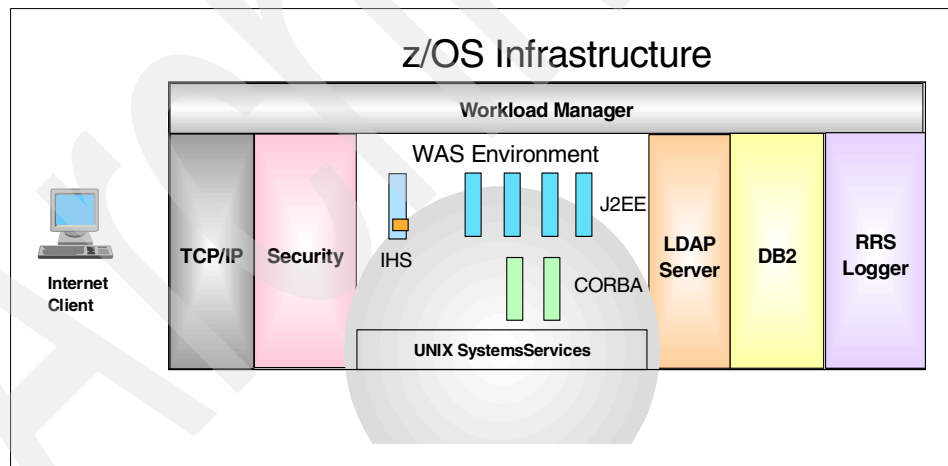


Figure 3-1 z/OS infrastructure

3.2.2 TCP/IP

The TCP/IP protocol is used as the basis for communication between the clients and the HTTP server.

It is also used to communicate between other portions of the WebSphere Application Server environment such the System Management Server and the System Management User Interface. See “Systems Management User Interface (SMUI)” on page 131 for more information about these components.

HiperSockets

HiperSockets, a new feature unique to the zSeries, provides a “TCP/IP network in the server” that offers high-speed any-to-any connectivity among virtual servers (TCP/IP images) within a z900 without any physical cabling. HiperSockets minimizes network latency and maximizes bandwidth between combinations of Linux, z/OS and z/VM virtual servers.

These OS images can be first-level (directly under an LPAR), or second-level images (under z/VM). With up to four HiperSockets per LPAR connection, one could separate traffic to different HiperSockets for security (separation of LAN traffic, no external wire-tapping, monitoring) and performance and management reasons (separate sysplex traffic from Linux or non-sysplex LPAR traffic).

Since HiperSockets does not use an external network, it can free up system and network resources, eliminating attachment cost while improving availability and performance. HiperSockets can have significant value in server consolidation, for example, by connecting LPARs of multiple Linux virtual servers under z/VM to z/OS machines.

Furthermore, HiperSockets will be utilized by TCP/IP in place of XCF for sysplex connectivity between images which exist in the same server, thus z/OS uses HiperSockets for connectivity between sysplex images in the same server and uses XCF for connectivity between images in different servers. Management and administration cost reductions over existing configurations are possible.

3.2.3 z/OS Unix Systems Services (USS)

z/OS Unix Systems Services, the IBM version of UNIX, provides some services needed to run the WebSphere Application Server environment and the Java applications that run within it.

This robust and mature version of UNIX is fully functional and integrated into the z/OS system. It takes advantage of zSeries hardware and the z/OS operating system to provide optimum performance for e-business applications.

3.2.4 Hierarchical File System (HFS)

HFS is used to store Java code, static files, and the configuration files needed for running the application servers.

One of the advantages of running on zSeries hardware is the availability of FastWrite DASD. Storing your e-business files on high performance storage can alleviate I/O bottlenecks that exist on other platforms

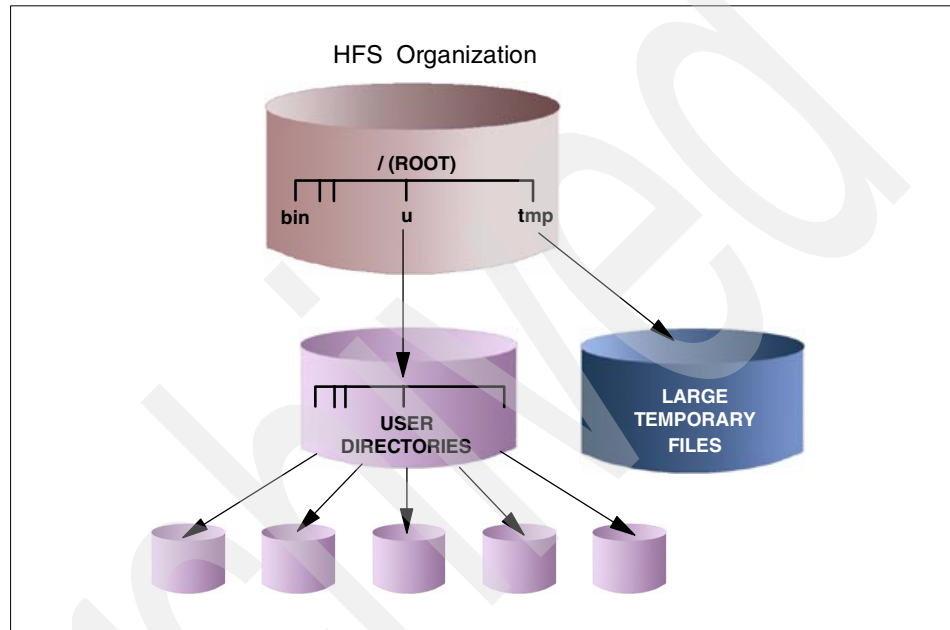


Figure 3-2 Hierarchical File System

3.2.5 LDAP/DB2

The z/OS Lightweight Directory Access Protocol (LDAP) server, part of the Security Server for z/OS, is based on a client/server model that provides client access to an LDAP server.

LDAP is a directory service protocol that runs over TCP/IP. Refer to *SecureWay Security Server LDAP Server Administration and Use*, SC24-5923 for more information.

An LDAP directory provides an easy way to maintain directory information in a central location for storage, update, retrieval, and exchange.

A directory is like a database, but tends to contain more descriptive, attribute-based information. The information in a directory is generally read much more often than it is written. As a consequence, directories do not usually implement the complicated transaction or rollback schemes regular databases use for doing high-volume complex updates.

In the WebSphere Application Server 4.0.x environment LDAP Directory services are used to provide a method for finding Java objects (EJBs) and CORBA Business Objects.

DB2

The data that is used by LDAP is stored in DB2 tables. This provides a persistent storage medium for the LDAP data.

The LDAP is used by both the “system servers”, such as BBOSMS and BBOIR, and the J2EE and MOFW application servers, but in different ways. See “Base Environment” on page 129 for more information about these system servers.

- ▶ The SMS and IR regions include DLL code to directly exploit naming information in their part of the LDAP database.
- ▶ J2EE application servers access the LDAP database through the LDAP server with the usage of the client/server Java Naming Directory Interface (JNDI) interface. The name service is mapped to the X.500 data model in the Lightweight Directory Access Protocol (LDAP) server on z/OS.

Note that the LDAP server is built on the data sharing implementation of DB2. This makes the Name Service implementation on z/OS fully transactional and recoverable. The Name Server also provides the residence for the lifecycle-related objects, such as Factory Finders and location objects.

3.2.6 Workload Manager (WLM)

The Workload Manager is used to automatically scale the number of available servers and balance the workload between those servers. This functionality is provided by the z/OS operating system and gives increased flexibility to WebSphere when running on the zSeries platform. See 7.3, “Workload Manager” on page 101 for more information about WLM.

WLM tuning provides the ability to automatically balance development with e-business running on the same platform. Performance goals, queue managers, and queue servers can all be further defined in this type of model.

Since no one controls the Internet, the ability to automatically scale your e-business horizontally to meet increased demand is an important capability.

3.2.7 Security/SSL

Security is needed in today’s Web-enabled environment. The IBM zSeries hardware and the z/OS operating system work together to lessen the impact of providing security.

Security is the corner stone of the z/OS environment. It extends across the entire operating system and protects all of your business resources on a 24/ 7 basis.

The zSeries “hardware storage keys” protect data that resides in memory from being read or modified by any unauthorized programs. Meanwhile, the z/OS SecureWay Security Server protects the rest of your resources from being tampered with or compromised by hackers.

In addition, the z/OS operating system software exploits the zSeries hardware Cryptographic Co-processors for SSL to provide security with a minimum of overhead.

See 6.1, “Overview of J2EE security” on page 76 for more detailed information concerning security.

3.2.8 LOGGER/RRS

System logger is a z/OS component that allows an application to log data from a sysplex. You can log data from one system in a sysplex or from multiple systems across the sysplex. A system logger application writes log data into a log stream (coupling facility or DASD). A coupling facility log stream can contain data from multiple systems, allowing a system logger application to merge data from systems across the sysplex. In a DASD-only log stream, interim storage for log data is contained in local storage buffers on the system.

Resource Recovery Services consists of the protocols and program interfaces that allow an application program to make consistent changes to multiple protected resources.

z/OS, when requested, can coordinate changes to one or more protected resources, which can be accessed through different resource managers and reside on different systems. z/OS ensures that all changes are made or no changes are made.

The Object Transaction Service (OTS) manages transactions and provides the CORBA OTS interface to z/OS Resource Recovery Services (RRS) which coordinates resource recovery across several Resource Managers. On the Server Region side, this is a very thin layer that delegates requests to the Control Region. The Object Transaction Service (OTS) requires existence of RRS. RRS in its turn requires the definitions of logstreams in a Couple dataset.

The IMS, CICS and DB2 Resource Managers have also implemented an interface to RRS so that, in the case of a WebSphere client application, resource coordination between the various Resource Managers can be driven by RRS. When the WebSphere client application completes its work and issues a commit, then the Control Region notifies RRS of this event and the two phase commit process begins. The client is notified about the outcome and may then continue with the next transactional Unit of Work (UOW).

The activation of RRS implies the use of the RRS Attachment Facility (RRSAF) when using DB2.

3.3 WebSphere Application Server environment

The WebSphere Application Server environment is where the heart of your e-business resides on z/OS. Depending on the needs of your business you'll have need of some or all of the following subcomponents of the WebSphere Application Server (Figure 3-3 on page 51).

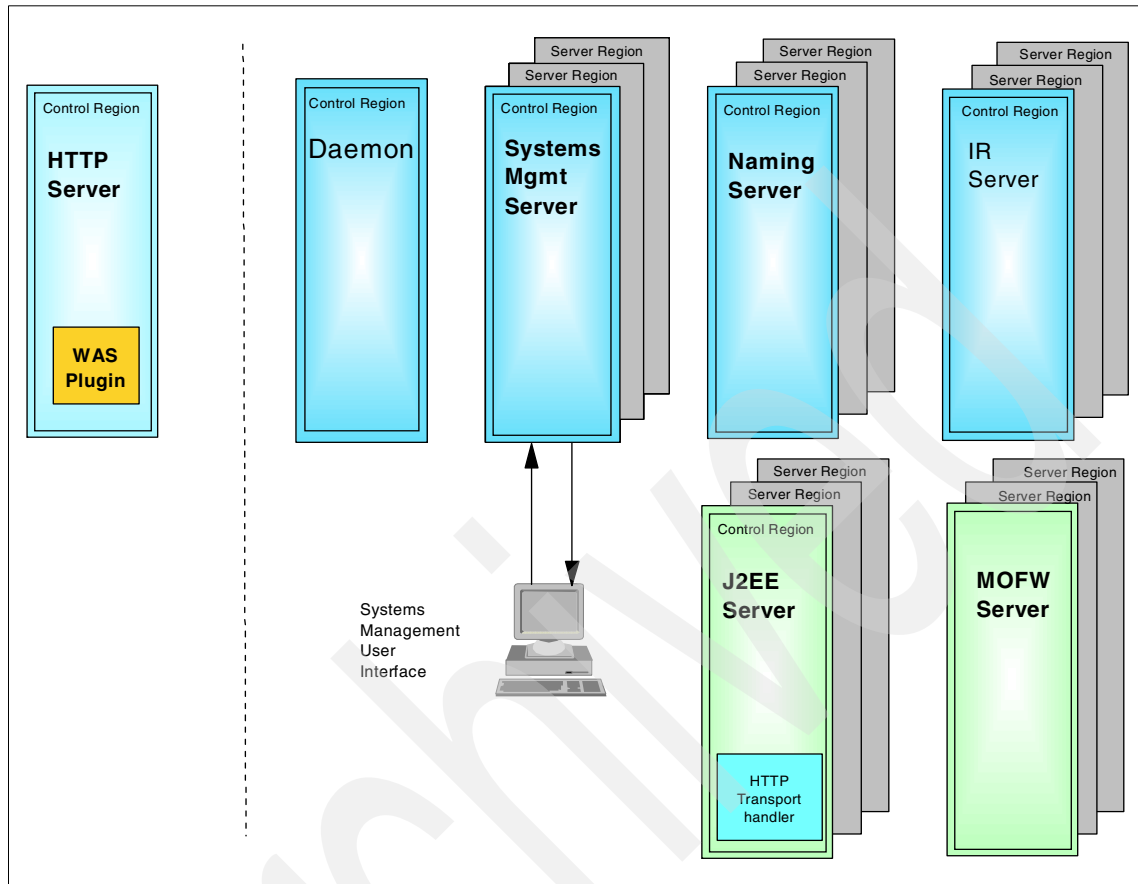


Figure 3-3 WebSphere Application Server environment

Each of the preceding subcomponents provides support for different pieces of your e-business on z/OS. For those readers familiar with the z/OS environment, each is a separate address space.

The z/OS WorkLoad Manager (WLM) is used to start multiple copies, depending upon the workload for several but not all of the subcomponents.

The following discussion is meant to give z/OS personnel a high-level overview of these subcomponents—a sort of quick “here are all the pieces of the puzzle” viewpoint. For more detailed information, refer to the appropriate chapters referenced in the overview.

HTTP Server

The HTTP Server is primarily used to serve static pages and provide an interface for CGI programs. As mentioned in 1.3, “What you need to do e-business” on page 6, the HTTP Server allows you to provide simple Web serving and an e-business presence on the Internet, but remember the technology is limited.

For more information about the HTTP Server, including CGI programs, see Chapter 8, “The IBM HTTP Server” on page 109.

The HTTP Server also provides an environment for the WebSphere Application Server Plugin environment. This plugin environment expands your ability to do business on the Internet.

WebSphere Plugin environment

The WebSphere Plugin is used primarily to route servlet and JSP requests to the WebSphere Application Server. It can also be used to run Servlets and JSPs and to communicate between servlets running in the WebSphere Plugin and the EJBs running in a J2EE Application Server. For more information about these technologies, see 2.5, “Java 2 Platform” on page 19.

For more information about the WebSphere Plugin, see Chapter 9, “WebSphere Plugin environment” on page 121.

HTTP Transport Handler

This component can handle HTTP(S) requests from clients. The protocol handler can speed up HTTP processing significantly, although it is not designed to deal with browser clients, so cannot handle the differences between different browsers. It assumes that there is an HTTP server in front of it that handles browser nuances (e.g., WebSphere Edge Server).

Application Server Environment

The Application Server Environment consists of a base environment that provides support for application servers, and the application servers themselves.

Currently there are two types of application servers that run in this environment: the J2EE application server and the CORBA(MOFW) server.

The J2EE application server is used to serve servlets and EJBs, while the CORBA application server is used for serving CORBA Business Objects. Both of these technologies can talk to each other in this environment. At this point it's important to understand the distinction between the two types of application servers.

J2EE

J2EE application servers are based on the Java language only and allow you to run J2EE components, such as servlets or EJBs on any platform that supports the JVM.

Important: The emphasis for a J2EE Application Server is platform neutrality.

The ability to write your business objects once and run them on any J2EE application server.

For more information concerning J2EE application servers, see Chapter 10, “J2EE Application Server” on page 127.

CORBA (MOFW)

CORBA servers, on the other hand, provide the ability to write your CORBA business objects in a number of different languages for use on different platforms and then network them together so that they function as a single entity.

Important: The emphasis for a CORBA application server is language neutrality.

The ability to share business objects between many different systems regardless of the language used to create the object.

Both of the technologies support object-oriented programming and object reuse. This means your investment in business logic can be shared across your entire enterprise.

They both also encapsulate and separate business logic from presentation logic. By using servlets and JSPs to build the HTML and then EJBs and CORBA objects to talk to backend systems, you allow changes to either interface without disrupting the other.

Base environment

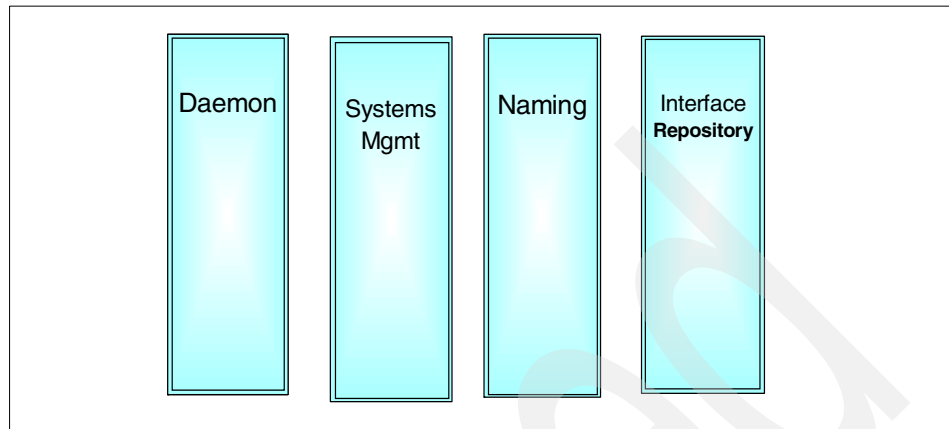


Figure 3-4 Base environment for application servers

The basic environment for the application servers consists of the following subcomponents. Each of these subcomponents is a separate address space in an z/OS system.

- ▶ **Daemon**
This WebSphere Application Server address space is started first; the Daemon is responsible for starting the SMS, Naming and IR Control regions.
- ▶ **SMS**
The System Management Server manages configuration data for all servers. It is the operations interface for installing and configuring servers.
- ▶ **Naming**
The Naming Server is responsible for providing names for object references.
- ▶ **Interface Repository**
Interfaces with the LDAP server.

For more information concerning the base environment for application servers, see “Base Environment” on page 129.

Systems Management User Interface (SMUI)

This application runs on a client machine and allows you to add and manage new Application Servers and applications via a graphical user interface

Connectors to backend systems

In addition to the supporting components, the z/OS system provides access to the traditional mainframe backend databases and transaction servers such as the following:

CICS	Customer Information Control System
IMS	Information Management System
DB2	Database2

You can tie into your backend systems with connectors, described in these categories:

- ▶ First generation connectors

This type of connector tends to have less standard tools, and less portability.

It is less focused on application re-engineering or technology upgrades and more focused on optimizing Web enablement to an existing application and data. They are not supported by WebSphere Application Server.

- ▶ Second generation (Java) connectors

These connectors are characterized by the use of Java and state-of-the-art tools.

For relational databases, Java DataBase Connectivity (JDBC) and Static Query Language Java (SQLJ) are provided. CICS and IMS have developed the CICS Transaction Gateway (CTG) and the IMS Connector for Java.

Applications for those connectors can be developed with a very high level of abstraction using the Common Connector Framework.

Second generation connectors are more focused on standards, tools, portability and beginning the process of application re-engineering using Java technology.

- ▶ Third generation (Enterprise Java) connectors

These connectors can be used as a bridge between Enterprise JavaBeans (EJBs) and existing traditional data stores and transactions.

The connectors are based on the Java Connector Architecture (JCA). The new CICS Transaction Gateway and the IMS Connect use this type of connector.

The third generation connectors are an evolution of the second generation with enhanced transactional and security context at the thread level, and data persistence built into the application programming model.

The z/OS platform's QoS (Qualities of Service) are provided to the application transparently. Examples of these key QoS advantages are Workload Management, RRS (Recoverable Resource Services) for distributed transactions and Parallel Sysplex for clustering and data sharing.

3.4 Application development tools

A robust set of application development tools is available for Web applications on z/OS. Several run on a workstation running one of the Windows operating systems (2000, 98 or NT), helping you create new Web-enabled applications by re-using key components of your existing applications.

They include VisualAge for Java Enterprise Edition, and WebSphere Studio and the new tool, WebSphere Studio Application Developer. You can develop on the workstation and deploy on z/OS.

3.4.1 Visual Age for Java

VisualAge for Java is a visual tool for building servlet or EJB applications in Java.

It automatically takes care of the linkage code required between the various components of the application, and leaves you to just develop the specific business logic that you need.

It supports many platforms for development, including Windows NT/2000. It supports many platforms for testing and debugging, including z/OS, OS/400, AIX, and Windows.

For more information concerning Visual Age for Java, see 17.2, “IBM VisualAge Family” on page 197.

3.4.2 WebSphere Studio

WebSphere Studio is a visual tool for building servlets, JSPs, database access, JavaBeans and HTML pages.

For more information concerning WebSphere Studio, see 17.3, “IBM WebSphere Studio” on page 199.

3.4.3 WebSphere Studio Application Developer

WebSphere Studio Application Developer is an integrated development environment that encompasses all of the previous development components. It provides a sound base for an integrated development environment, where different vendors can provide different components of the IDE. WebSphere Studio Application Developer is the future of integrated J2EE development.

Sample scenarios

In this chapter we look at three sample e-business enablement scenarios using an OS/390 server. We looked at these from three business requirements:

- ▶ If you want to serve static information on the Web, you should look at the *Web serving* scenario. This scenario applies to fairly simple applications, but will probably be your starting point in e-business.
- ▶ If you want to provide Web access to data or other applications, you should consider whether that data and those applications already exist today. If they do, you should look at the *Web-enabling current systems* scenario. In this scenario, we assume little or no new application logic.
- ▶ If you do need new data and application logic to deliver your e-business solution, you may find the *New business logic* scenario relevant to your needs.

These scenarios are high level, but choosing one of them already sets the stage for the possible technical solution alternatives. The scenarios and their solution types are summarized in Table 4-1 on page 60.

Table 4-1 e-business enablement scenarios and solution types

Requirement	Solution type	Main components of the solution	AD tooling
Web serving Serving static information No access to applications or data	An HTTP-capable Web server, such as WebSphere and Domino, provides access to the information.	IBM HTTP Server for OS/390 No connectors required.	WebSphere Studio, WebSphere Homepage Builder, NetObjects, or any other Web site development tool.
Web-enabling current systems Enabling current systems to be accessed from the Web	Clients connect to back-end applications and data through a Web Application server, such as WebSphere. This server may reside on z/OS or on a middle-tier server.	IBM HTTP Server for OS/390 and, additionally, WebSphere Application Server in case Java servlets or JSPs are used. Lotus Domino may be used as an alternative. One or more connectors are needed.	WebSphere Studio, NetObjects or any other Web site development tool. For development of Java servlets or Beans, VisualAge for Java or any other Java IDE (Integrated Development Environment). Domino Designer (in case Lotus Domino is used as the Web server).
	Web browser to 3270 interface	Host on-Demand and IBM HTTP Server for OS/390.	None required.
	Clients connect directly to back-end application in a 2-tier model.	Multi-tier connector required. (No HTTP server required).	Depends on solution.
New Business Logic	Based on EJBs	WebSphere EJB connectors are required in case back-end systems need to be accessed.	VisualAge for Java and/or WebSphere Studio Application Developer
Developing new business logic, preferably transactional	Based on traditional programming languages	Transactional code in IMS, CICS or DB2. Connectors are required to get from the Web to those new traditional transactions.	VisualAge family (depending on the language) WebSphere Studio Application Developer
	Packaged solution	For example, WebSphere Commerce Suite (WCS)	Tools usually integrated in the solution.

We now look at each of these requirements and solutions in more detail.

4.1 Scenario 1: Simple Web serving

In this scenario, the objective is to serve static information. We define static information as information that cannot be changed by application logic. Static information can be HTML text files, graphic files, audio files, and so on. Typically, there are no application programs involved and there is no access to databases. The only thing the client does is retrieve files with information from the server and use those files in a browser. While this may be an initial starting point, it is likely that you will want to move to one of the other scenarios eventually.

The HTTP server is key in this scenario, as shown in Figure 4-1.

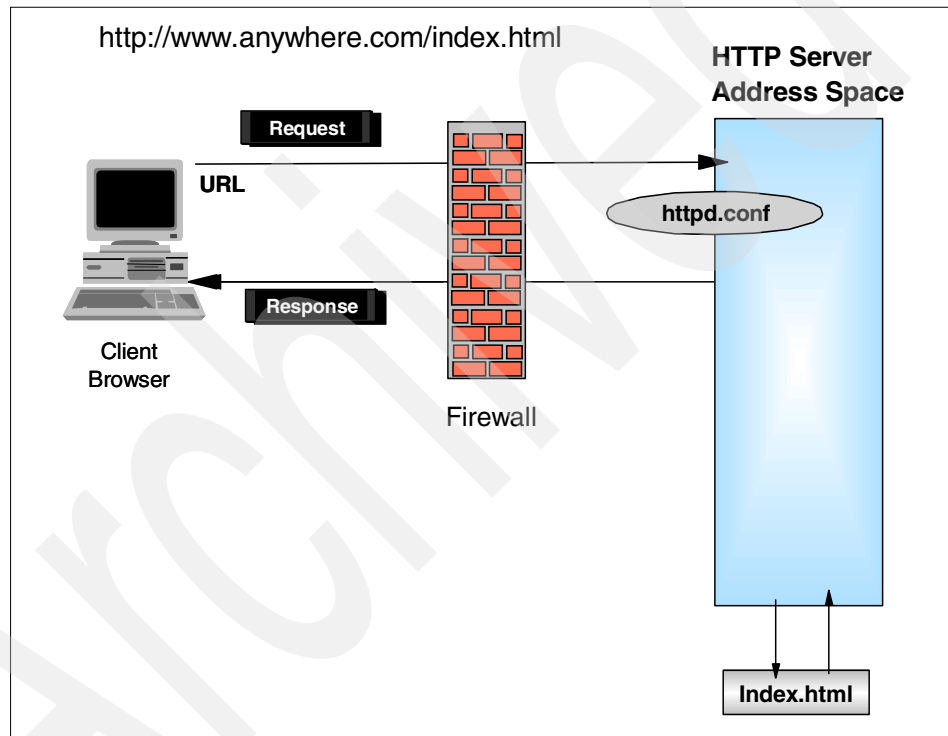


Figure 4-1 Web serving scenario

The following aspects are key in this type of solution:

- The client sends requests to the HTTP Server using the HTTP protocol. This connection can easily be encrypted using the built-in SSL support of the browser and the HTTP server.

- ▶ The HTTP server reads the requested files from the Hierarchical File System (HFS) on OS/390 and sends them back to the client. The HTTP server knows where to find the specific files based on configuration files.
- ▶ There are no connections from the HTTP server to any other subsystem on OS/390.
- ▶ The content stored on OS/390 can be developed with tools like WebSphere Studio, WebSphere Homepage Builder, NetObjects Fusion, or any other Web content development tool.
- ▶ Users can be authenticated by the HTTP server, using a security database on OS/390, such as RACF.

4.1.1 Variations

A variation on this solution is to use the Lotus Domino server to serve information from Lotus Notes databases as well as from UNIX files. Either the IBM HTTP Server for OS/390 or the integrated Lotus Domino HTTP server can be used. Domino Designer would be used for development. User authentication can be accomplished by Domino using its own directory.

4.2 Scenario 2: Web-enabling current systems

In this scenario, we assume that you have one or more of the following systems:

- ▶ A transaction server (CICS and/or IMS)
- ▶ A database server (DB2 or any other relational database supported on OS/390)
- ▶ A messaging server (MQSeries)
- ▶ One or more file systems, such as VSAM

The requirement is to put a Web presentation layer onto an existing application, in many cases without changing the existing application logic. We assume no (or only a minimal) requirement for new application logic.

There are three types of solutions for this requirement:

- ▶ The client accesses the back-end system through a Web server. The Web server could be the IBM HTTP Server for z/OS only, or a combination of the IBM HTTP Server for z/OS and WebSphere Application Server in case the connection to the back-end system is Java-based.
- ▶ The client starts up a browser with a Host on Demand applet emulating a 3270 interface.

- The client makes a direct connection to the back-end system and uses a *gateway*.

4.2.1 Client accessing the back-end system through a Web server

In this solution, the client connects to a Web server using the HTTP protocol. Typically, the client request includes the invocation of a “server-side” program to be executed by the Web server. The server-side program is capable of initializing, maintaining and closing a connection with another subsystem. For example, the program is capable of sending an SQL query to DB2 and receiving the results. Once the results are received, the Web server forwards the formatted results to the client who made the request. This process is shown in Figure 4-2.

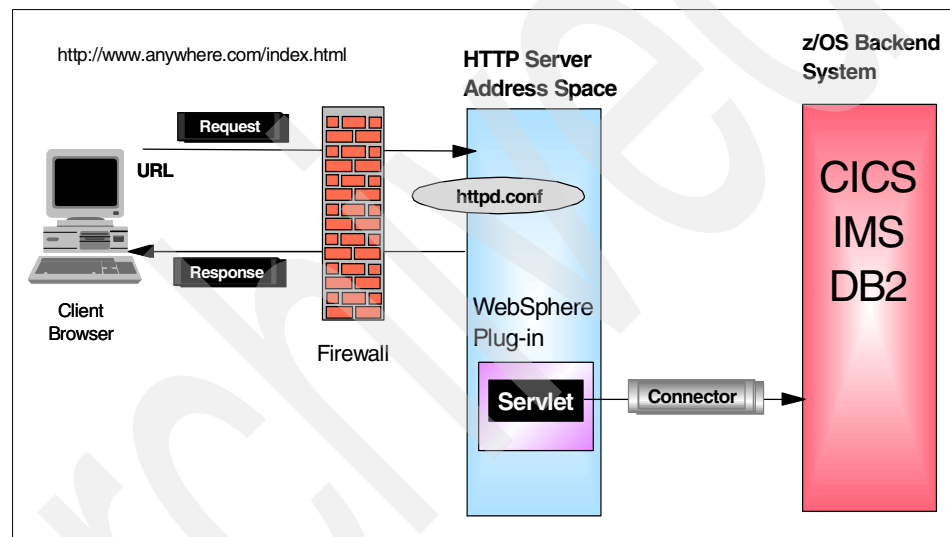


Figure 4-2 Web-enabling legacy systems

The following aspects are key in this solution:

- The client accesses the back-end system through either the IBM HTTP Server for OS/390 only or a combination of the IBM HTTP Server for OS/390 and WebSphere Application Server.
- A server-side program is used to establish the connection between the Web server and the back-end system. If a CGI script or a C/C++ program is used, there is no need to use the WebSphere Application Server component.

See “CGI Interface” on page 114 for more information concerning CGI application programs

- ▶ If a Java-based program is used, the WebSphere Application Server plug-in runs this program on behalf of the IBM HTTP Server. We assume in this scenario that the strategy is to use Java-based connections.
- ▶ WebSphere Application Server is actually run as a plug-in to the IBM HTTP Server. It manages and runs servlets and JavaServer Pages (JSPs) containing the presentation logic to format the data coming from the back-end systems. They format the result into HTML for a browser (other formats are possible for other HTTP clients).

For more information concerning the WebSphere plug-in, see Chapter 9, “WebSphere Plugin environment” on page 121.

- ▶ For the connection between the Web server and the back-end system, a connector can be used or you can manually code a plug-in using a native protocol, such as Advanced Program-to-Program Communication (APPC). A connector does not require that the developer of the plug-in know the technical details of the protocol to access a specific back-end system.
- ▶ To develop this solution, you would use WebSphere Studio to develop the *Web components*, which are the Web pages, JavaServer Pages, and Java servlets. VisualAge for Java is the appropriate tool for developing business logic components, which are currently JavaBeans and Enterprise JavaBeans. Users can be authenticated by the Web server, using the RACF security database.

Note: Throughout the remainder of this book, we assume that the use of Java is strategic in developing server-side logic to be run on the Web server. From now on, when we talk about plug-ins, we target servlets and JavaServer Pages for presentation logic and Enterprise JavaBeans for business logic.

4.2.2 Variations

Some possible variations for accessing the back-end system through a Web server include:

- ▶ Using the Lotus Domino server instead of WebSphere Application Server to make connections to back-end systems.

Either the HTTP Server for OS/390 or the integrated HTTP server can be used as the HTTP server component. In the case of using Lotus Domino, *agents* running in Domino would replace the servlets and JSPs. The agents can include data from Notes databases in the response to the client. Connectors are also available from Domino to the back-end systems. Domino Designer can be used for development. User authentication can be done by Domino using its own directory.

- ▶ Using a different plug-in to the IBM HTTP Server rather than WebSphere Application Server.
For example, Net.Data is a plug-in to the IBM HTTP Server and it connects to DB2. In this variation you do not need WebSphere Application Server.
- ▶ Writing your own logic to access a back-end system based on one of the available communication protocols, if there is no connector available that meets your needs.
You may have to use the Java Native Interface (JNI) to use a native communication protocol such as APPC or Open Transaction Manager Access (OTMA) from a Java servlet or JSP.

4.2.3 Host on Demand

IBM SecureWay Host on Demand (HOD) provides access to 3270-type applications from a Web browser. This solution is shown in Figure 4-3.

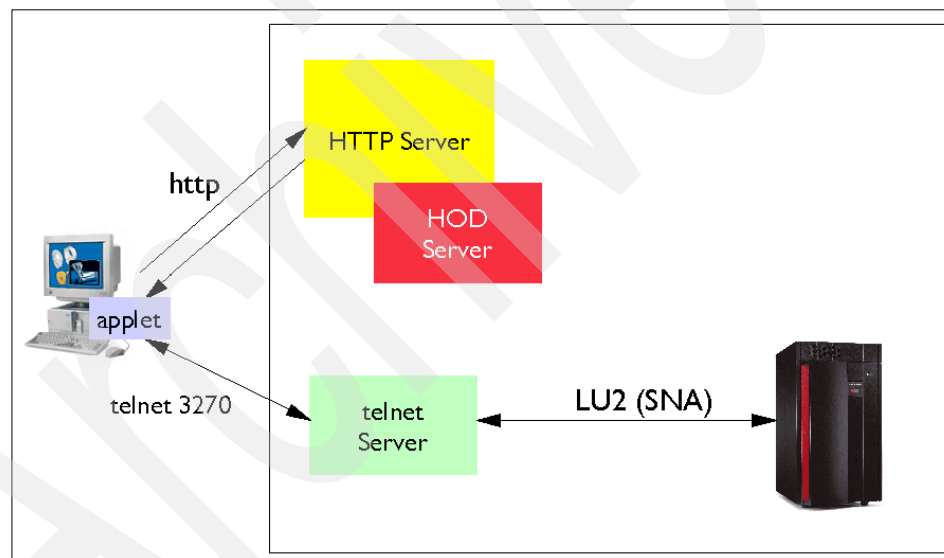


Figure 4-3 Host On Demand

The following aspects are key in this solution:

- ▶ The Web browser connects to the HTTP server using the HTTP protocol. The HTTP server reads the HOD client applet from the UNIX file system and sends it to the Web browser.
- ▶ The browser loads the applet.

- ▶ The applet makes a TCP/IP connection to a Telnet 3270 server running on OS/390. Alternatively, the applet can connect to an HOD server that will redirect the request to a Telnet 3270 server.
- ▶ The Telnet 3270 daemon connects to the back-end system.
- ▶ The connection from the applet to the Telnet 3270 server can be encrypted using SSL.
- ▶ No development of code is required. You can customize the interfaces if you want.
- ▶ User authentication is done by the back-end system.
- ▶ If you want to use a firewall, it needs to allow Telnet sessions.

For more information, see *IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing*, SG24-2149.

4.3 Scenario 3: New business logic scenario

In this scenario, we assume that the objective is to write a significant portion of new business logic. In case some or all of the new components require transactional properties, we recommend that you use an application server with transactional capabilities. Refer to Chapter 5, “Transaction services” on page 69, for a better understanding of transactions.

We also assume that you need to access back-end systems, either now or in the future. Even if you are starting a brand new application, it is likely that you will need to connect to other applications at some time.

There are several solutions for this requirement:

- ▶ Write the new business logic in Enterprise Java Beans and deploy it in an EJB container. A CORBA solution could also be used.
- ▶ Write the business logic using traditional programming languages and deploy them in an existing transaction server such as CICS or IMS.
- ▶ Implement a packaged solution, such as WebSphere Commerce Suite for e-commerce.

4.3.1 Enterprise JavaBeans

Enterprise JavaBeans (EJBs) provide support for transactions and other facilities needed by mission-critical enterprise applications.

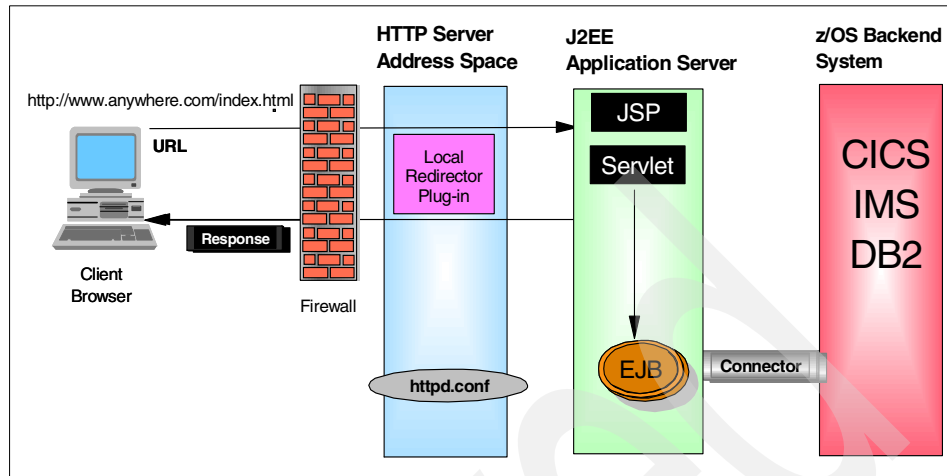


Figure 4-4 Future architecture with WebSphere Application Server Enterprise Edition and EJBs

The following aspects are key in this solution:

- ▶ The client sends a request to the HTTP Server on OS/390 using the HTTP protocol. The request typically includes the invocation of a servlet or a JSP
- ▶ The request to run a servlet or JSP is passed on to one of the WebSphere address spaces.
- ▶ WebSphere manages and runs servlets and JSPs that contain the presentation logic to format the data coming from the back-end systems.
- ▶ WebSphere will provide a container to run Enterprise Java Beans (EJBs). This container provides transactional and other services.
- ▶ The servlets or JSPs invoke the EJBs. The EJBs contain the new, transactional business logic, and the servlets/JSPs should only contain presentation logic.
- ▶ The EJBs can connect to back-end systems using connectors.
- ▶ To develop this solution, you would use WebSphere Studio to develop the Web pages and JSPs and VisualAge for Java to develop servlets, JavaBeans used in servlets and JSPs, and EJBs. You may also use WebSphere Studio Application Developer.
- ▶ Users can be authenticated by the HTTP Server using the RACF security database.

4.3.2 Variations

Some possible variations are:

- ▶ A Java client application can invoke the EJBs directly using the Remote Method Invocation/Internet Inter-ORB Protocol (RMI/IIOP).
- ▶ You could also code applications in C++ using the CORBA interfaces. This also provides transaction and other capabilities similar to EJBs and is supported in WebSphere Application Server.
- ▶ CICS also provides an EJB container, so the new application logic could be written in EJBs but deployed within CICS rather than WebSphere for zOS.

4.3.3 Existing transaction server

Existing transaction servers, such as CICS and IMS, already provide the services that you need for running transactions. That way, you can develop new business logic in those systems and access it using any of the solutions in 4.2, “Scenario 2: Web-enabling current systems” on page 62.

Transaction services

In this section, we discuss a group of products that are key in e-business and are seen of more significance on z/OS than on other platforms. Since the beginning of the 1970s, online application programs on z/OS are typically run through a transaction processor.

The well-known Information Management System (IMS) and Customer Information Control System (CICS) are leading in this area and have an outstanding record of availability and reliability. The transaction concept goes hand in hand with z/OS, and the combination of z/OS and either IMS or CICS is an extremely reliable environment for mission-critical applications.

5.1 Introduction

This section is not intended to be a thorough discussion of the transaction process, but just a picture to understand the importance of transaction processing in e-business and the main transaction servers on the z/OS platform.

5.2 Transaction servers versus application servers

In the core of any computer-based business system, there must be a component that executes business applications according to certain calls or requests. This subsystem is what we usually call an application server or application manager (for instance, WebSphere, IMS, CICS, Lotus Domino). In many cases, the applications running in one of these application servers need to access data managed by other servers (data managers), or call other applications located in other application servers.

When an application server running a certain application accesses other resources, either data managers or other application managers, it is very likely that all changes made by each server have to be coordinated. This concept is not new. In traditional environments like IMS or CICS, this is what we call a transaction.

Nobody would expect an IMS transaction to end successfully if some database access didn't work, while the rest of the process had finished perfectly well. The capability of rolling back a whole set of dependent changes if one of them is not successful, is what makes the difference between a simple application server and a transaction server.

WebSphere Application Server is an application server with transactional capabilities.

5.3 Transactions in e-business

In the e-business environment, transactions become even more important since, almost by definition, e-business has to be thought of in terms of pervasive computing. That is, resources can be spread across several computers, and many different types of servers can be accessed in a single request. It is, therefore, very important to have an architecture capable of “tying up” all those related changes and reverting them all if one of them did not complete successfully.

In the development of your new e-business applications, you have to consider the importance of having a good transactional environment because it may simplify the development a lot. Even if you are planning to make existing applications accessible through the Web, and believe that you do not need that much sophistication, it is likely that in the future you will need more e-business capabilities. Then, a transactional type of development will be needed.

Since e-business implies a lot of “talk” between different computers and probably different platforms, it is important that they all agree on the language they use to achieve a true transactional environment. Therefore, standardization of functionalities, protocols and services is crucial.

5.4 The Unit of Work process

Let's call a Unit of Work (UOW) any set of dependent changes. A UOW could consist of some transactions (one to CICS, another to IMS) and some database access (for example, to DB2). It is up to you to decide when all those changes make up a single atomic unit that either finishes successfully for any involved component, or has to be rolled back as if it had never been initiated.

We know that transaction managers (or data managers) can take care of their part of the process, but who takes care of coordinating all the transaction managers involved in a UOW? There needs to be a way to detect when a transaction involved in a UOW didn't complete, and inform all the other components that they have to roll back their changes, even if they successfully completed their transactions.

5.4.1 Resource Recovery Services (RRS)

On the z/OS platform, this is done by a system component called Resource Recovery Services (RRS). RRS is a sync point coordinator. This means that it can talk to all the transaction managers (and data managers) involved in a Unit of Work and detect when one of them did not successfully finish its part of the work. Any transaction manager or data manager that needs to talk to RRS must first be registered. Otherwise, how would RRS know they exist?

5.4.2 The two-phase commit protocol

The two-phase commit protocol is the technique used by RRS and the transaction coordinator to achieve the commit or rollback of all the components involved in a UOW:

1. When the global transaction controller arrives at the end of the UOW, it uses RRS to issue a *prepare to commit* statement to the other transaction

managers or data managers. They then do whatever is necessary to ensure that they can either make the changes permanent or roll them back. When this is done, they acknowledge the prepare to commit statement either positively or negatively. RRS collects all the “votes”.

2. If all the votes are positive, RRS asks all the resource managers to commit their changes. If any of the votes are negative, RRS asks all resource managers involved in the UOW to roll back. At this point, RRS forgets about that UOW and it is up to each resource manager to complete the commit or roll-back for their part.

Figure 5-1 summarizes the two-phase commit protocol concept.

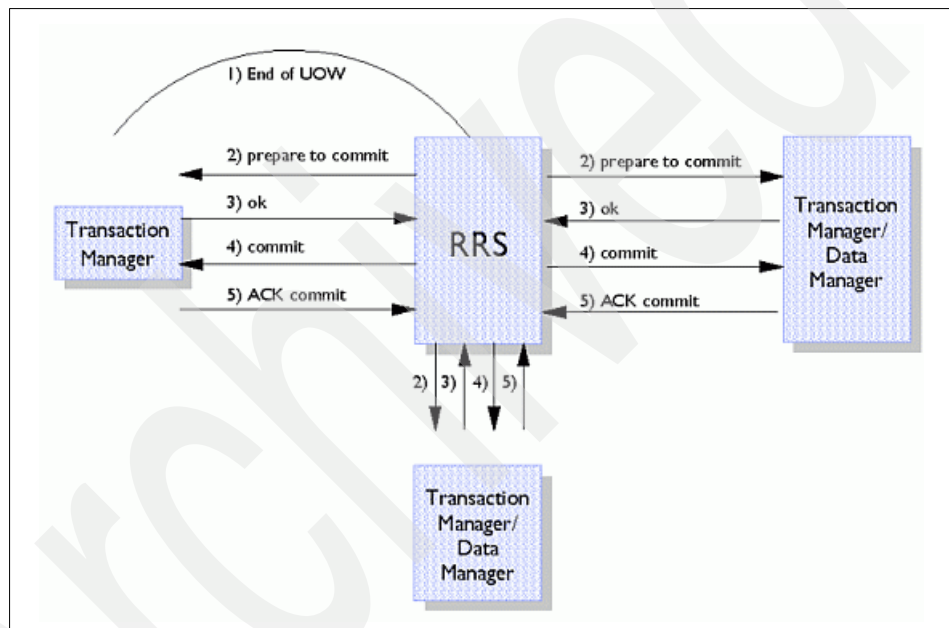


Figure 5-1 Two-phase commit protocol used by Resource Recover Services (RRS) on z/OS

The flow shown in Figure 5-1 is explained here:

1. The transaction coordinator, which could be any transaction manager, tells RRS that the UOW has finished.
2. RRS sends a “prepare to commit” message to all involved resource managers.
3. Resource managers answer positively or negatively to the “prepare to commit”.
4. If all the votes are positive, RRS tells the resource managers to commit.

The resource managers acknowledge the commit. From now on, RRS forgets about these resource managers for this UOW. It is up to them to finish the commit process.

Archived

Security management

In this chapter we discuss security aspects of e-business.

We first provide an overview of the J2EE security model and how it relates to WebSphere for z/OS. Most of the security information is stored in directory services, like the Lightweight Directory Access Protocol (LDAP), so we deal with that next. Finally, we discuss firewalls and how they relate to z/OS.

6.1 Overview of J2EE security

The notion of security includes several aspects. *Identification* is the process by which users or programs that communicate with each other provide identifying information. *Authentication* is the process used to validate the identity information provided by the communication partner. *Authorization* is the process by which a program determines whether a given identity is permitted to access a resource, such as a file or application component. The J2EE specifications describe the concepts to be used for these processes.

Although data integrity, confidentiality, non-repudiation and auditing are also important aspects of security, the J2EE specifications do not address them in any detail.

J2EE specifies a component programming model for security that provides both application programming interfaces (APIs) and declared properties. The security APIs used in the logic of application programs are referred to as *programmatic security*. The declared security properties, called *declarative security*, are found in components' deployment descriptors. One objective of the J2EE programming model is to encourage the use of declarative security, which is enforced by the container. This removes much of the responsibility for security from the application developer.

6.1.1 Programmatic security

It is desirable to use declarative security wherever possible in order to relieve the application programmer from the task of having to add security-relevant code to an application.

In cases where this is not feasible, or where the program logic requires security to be handled by the program, programmatic security can be used, as defined by the J2EE specification: programmatic security refers to security decisions made by security-aware applications. Programmatic security is useful when declarative security alone is not sufficient to express the security model of the application.

An API is available with two methods each for the Web container and the EJB container.

The methods to be used in Web applications are:

- ▶ `getUserPrincipal`
- ▶ `isUserInRole`

In EJB applications, the following methods are to be used:

- ▶ `getCallerPrincipal`
- ▶ `isCallerInRole`

Methods `getUserPrincipal` and `getCallerPrincipal`, respectively, are used to retrieve the userid associated with the current Web application (Servlet or Java Server Page) or EJB.

Methods `isUserInRole` and `isCallerInRole`, respectively, can be used to check if the current user has been given access to a certain role. Based on the result, the program can make decisions in its processing path.

6.1.2 Declarative security

The J2EE specification defines declarative security as the means of expressing an application's security structure, including security roles, access control, and authentication requirements in a form external to the application. The deployment descriptor is the primary vehicle for declarative security in the J2EE platform.

In the deployment descriptor of both Web applications and EJB applications, security constraints can be defined that need to be satisfied for the application to be allowed to run. These constraints can be defined on the level of individual methods.

J2EE does not provide a security policy. J2EE provides APIs and security properties. J2EE products, such as WebSphere Application Server for z/OS and OS/390, can implement the APIs and security properties in a way that defines a security policy.

6.1.3 Role-based authorization

For both programmatic and declarative security, J2EE uses the concept of *security roles*. Security roles are used to delineate permissions based on business rules.

For example, in a bank, there might be a business rule that allows tellers to accept and process withdrawals and deposits for customers, but might require that wire transfers to other financial institutions be processed only by supervisors. In order to implement this rule in a J2EE application program, we could define security roles called "teller" and "supervisor" and use APIs and security declarations to control access to program logic based on these security roles.

A role-based model may be contrasted with a more traditional security model, sometimes called permission-based. In a traditional security model, resources such as programs, files, or devices are associated with users or groups of users who are given various levels of permission with respect to those resources. For example, a user or group might have permission to read a file, but not to update it. Another group might have permission to execute a file (presumably containing program code), but not to browse it with an editor so as to discover its inner logic.

Typically, both the user registry and the resources are defined in a security database. Users and groups are associated with resources in some way, such as with an access control list (ACL). The maintenance of the user registry, resource profiles and ACLs requires a security administrator who must have enough information about each user and each resource to make the proper decisions.

In the J2EE role-based model, users and groups of users are still stored in a user registry. A mapping must also be provided to specify the assignment of users and groups to security roles. This can also be contained in a user registry, or it can exist in a separate file. J2EE applications, however, have their own role-based security constraints, so that the applications themselves do not have to be defined as protected resources by a security administrator or defined in a registry or security database.

J2EE applications, moreover, display considerably more flexibility than traditional applications with respect to security constraints. Typically, traditional applications as a whole are protected, if at all, by an ACL that is global with respect to the program. A user or group of users may or may not have permission to execute the program as a whole, but there is no means of restricting parts of the program including authorization logic within the application code itself. J2EE applications have a declarative means of protecting each individual method of each component by specifying, outside the program logic, which security roles may execute it.

Of course, there may be drawbacks to not having a central registry where all programs and their access controls are maintained. For instance, the J2EE role-based security model may make third-party authentication more difficult to implement.

To implement both programmatic and declarative security, application developers (including what J2EE refers to as “component providers” and “assemblers”) declare role names in the component’s deployment descriptor. In addition, for declarative security, the application assembler associates security roles with “method-permission” elements in the deployment descriptor. A user may invoke a method only if a role to which the user’s identity is mapped is listed on that method’s method-permission element.

Deployers make J2EE components and applications ready for particular operational environments. Part of this deployment process includes editing the deployment descriptor (usually through a graphical tool) to map security roles to users or user groups that are understood by the security implementation of the environment.

As depicted in Figure 6-1, application developers (component providers) may use J2EE roles both programmatically and declaratively. The application assembler can link the role names used by application components from multiple sources. The security administrator or deployer provides role names that meet the requirements of the system where the application is being deployed. The security administrator links or permits individual users or groups to the role names.

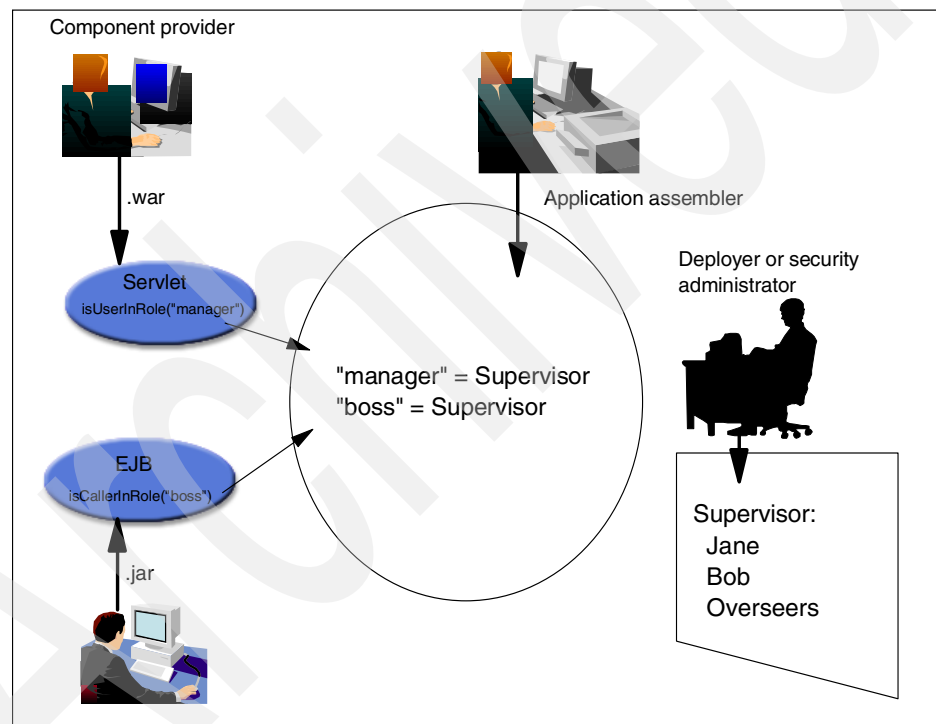


Figure 6-1 J2EE security roles

6.1.4 Terminology used for J2EE security

There are a number of terms that have a particular meaning in J2EE.

Principal	An entity that can be identified and authenticated (for example, the initiator of a request, such as a user or requestor).
Principal name	Identity of a principal (a userid, for example).
Credential	Reference information, such as a password or certificate, that can be used to authenticate a principal.
Subject	A set of principals and their credentials associated with a thread of execution.
Authentication	The process by which an entity, such as a server, verifies the identity presented by a user. This process usually requires some form of credential information such as a password known only to the user.
Authorization	The process of determining what type of access (if any) the security policy allows to a resource (for example, a file or method) by a principal.
Security role	A logical grouping of users who share a level of access permissions.
Security domain	A scope that defines where a set of security policies are maintained and enforced. (This is also referred to as a “security policy domain” or “realm.”)

6.2 Authentication and authorization in J2EE containers

WebSphere Application Server for z/OS has separate containers for Web applications and Enterprise JavaBeans (EJBs). Each container supports different APIs and properties. These are discussed in 6.2.1, “Web container authentication and authorization” on page 80 and 6.2.2, “EJB container authentication and authorization” on page 85. Both containers use a similar model for authorization, however.

6.2.1 Web container authentication and authorization

Web containers implement J2EE-specified methods of authentication and authorization.

Web container authentication

Web containers support the following methods of authentication:

- ▶ HTTP basic authentication
- ▶ HTTPS basic authentication
- ▶ Form-based authentication
- ▶ HTTP digest authentication
- ▶ Client certificates

HTTP basic authentication

HTTP basic authentication is based on a userid and password. When processing a request for a protected resource, the container looks in the HTTP message for a userid and password. If these are found, the container then consults the operating platform's security mechanism to verify that the password is correct. If the password matches, the userid is authenticated and a principal is established.

If the userid and password are not found, or if the password does not match what is expected, the container returns an HTTP 401 response. This response causes the browser to prompt the end user for a userid and password. The application developer or deployer can configure a realm name in the deployment descriptor. The container passes this realm name with the 401 response, and the browser displays it in the prompt.

Figure 6-2 on page 82 graphically depicts the flow of basic authentication.

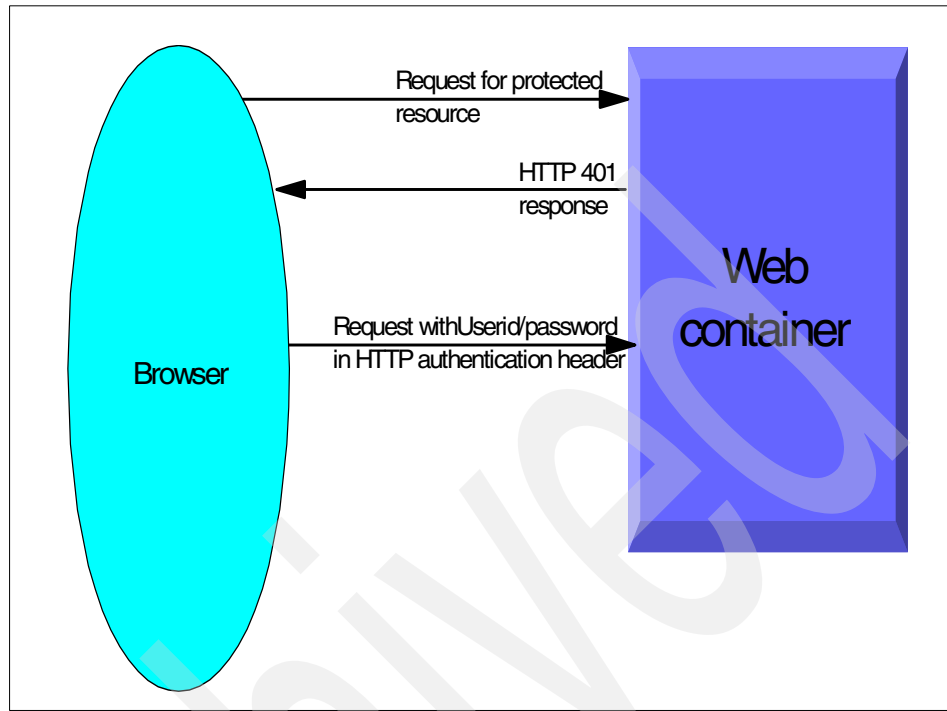


Figure 6-2 HTTP basic authentication

HTTP basic authentication is not considered to be secure because the userid and password are included, in a simple base64 encoding, with each request. In addition, the userid and password are part of the HTTP header, available to the application program itself.

HTTPS basic authentication

The container behaves similarly with HTTPS basic authentication. HTTPS uses Secure Socket Layer (SSL) to establish a secure, encrypted session between the browser and the HTTP server. If someone on the Internet intercepts the session, the userid and password are not easily discovered.

Form-based authentication

Form-based authentication may take place over an unencrypted or encrypted session. In either case, it offers several clear advantages over basic authentication.

Under the form-based authentication protocol, the application provides two files. One is called the *login screen* and is used to solicit a userid and password from the end user if none is passed in the HTTP header.

The other is an *error screen* that is returned to the user if the userid and password do not meet the requirements of the security system. These screens provide a much richer opportunity for the application developer to tailor what the end user sees during the authentication process.

Figure 6-3 shows that when the Web container receives a request for a protected resource, it returns the login screen defined by the application developer. The login screen must contain an HTTP form with the action being `j_security_check`. The form must solicit a userid and password from the end user.

When the user clicks the action button, the `j_security_check` method (which is defined by the Web container vendor, not the application developer) is invoked to authenticate the userid and password. If the authentication is unsuccessful, the error screen is returned. Otherwise, the original request is processed if the authenticated user is authorized (see “Web container authorization” on page 85).

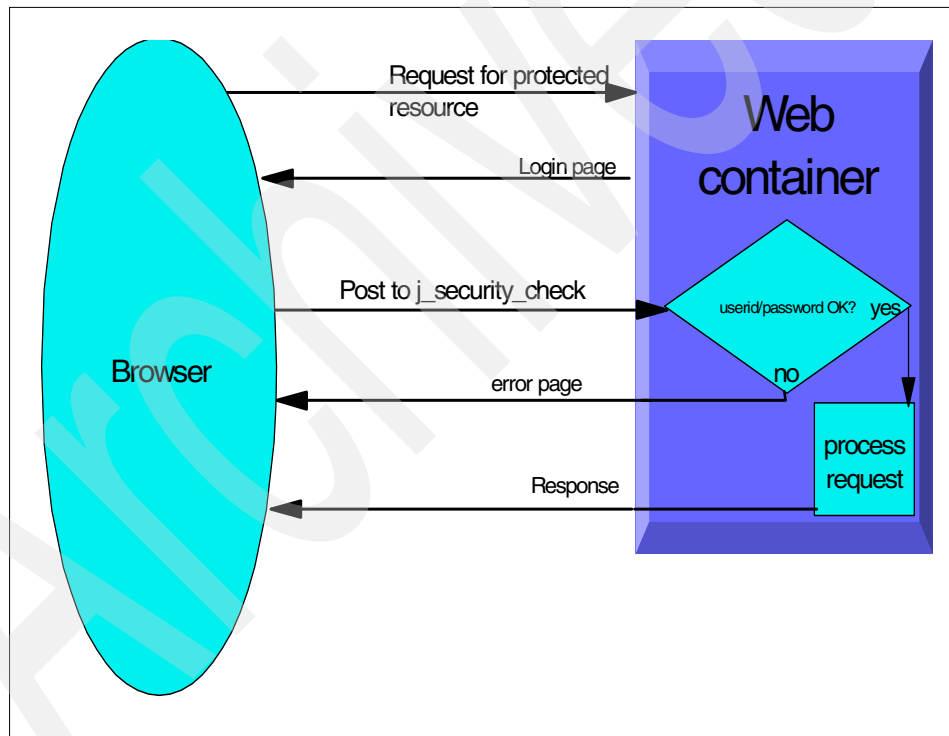


Figure 6-3 Form-based authentication

Furthermore, there is no need for the userid and password to flow with each request. When the container authenticates the userid and password, it creates a login session containing the principal and credentials. It can then return a token

that is meaningful only in the container's operating environment. The browser can return this token instead of a userid and password on subsequent requests.

If the login session is shared within a security domain, sometimes called a "realm," it allows a single signon capability within that domain. The J2EE specification requires vendors to support single signon, but does not define the scope of a security domain.

HTTP digest authentication

RFC 2617 "HTTP Authentication: Basic and Digest Access Authentication" specifies a challenge-response authentication mechanism called Digest Authentication. This protocol was created to overcome the biggest drawback of basic authentication—the sending of the user's userid and password in the clear.

With digest authentication, the password is not sent over the communications channel during the authentication exchange at all. The server challenges the user with a "nonce", an unencrypted random value. The user responds with a checksum (typically, an MD5 hash) of the userid, the password, the nonce, and some additional data. The server creates the same checksum from the user parameters available in the user registry. If the two checksums match, it is assumed that the user knows the correct password.

Support for digest authentication in J2EE is optional. With RACF as user registry, the user's password is not available in clear text, so the checksum cannot be constructed from the password stored in the RACF database. WebSphere Application Server does not support digest authentication (on any platform).

HTTP digest authentication is mentioned but not defined in the J2EE specification. It is permitted, but not required for Web containers.

Client certificates

The J2EE specification also requires that containers support the use of client certificates for authentication. The J2EE specification refers to this as "SSL mutual authentication."

Specifying the authentication method to be used

An application developer specifies what authentication method is to be used for an application by providing a *login-config* element in the application's deployment descriptor. The login-config element specifies the type of authentication to be used and any associated data, such as login and error pages for form-based authentication.

Web container authorization

The Web container supports two APIs for programmatic authorization. They belong to the `javax.servlet.http.HttpServletRequest` interface. The application programmer can use `getUserPrincipal()` to determine the identity of the authenticated user. The application logic can then make authorization decisions based on this identity. This method takes no arguments and returns a `java.security.Principal` object.

`isUserInRole()` takes a role name in the form of a `java.lang.String` as an argument and returns a boolean result. The result will be true if the current authenticated user has been authorized to the role name passed on the call. This is useful when a servlet method may be accessed by users in more than one role. It gives the application programmer a way to determine if the current user has been authorized to a specific role of the one or more that may be authorized to access the servlet method.

An application programmer who uses `isUserInRole()` must define the role names separately in the deployment descriptor file. This is done through the deployment descriptor's *security-role-ref* element. This element is contained within the *servlet* element, so that the roles defined in one servlet may be independent of those defined in other servlets. The application programmer provides a description of each role as a means of communicating its significance to the application assembler or deployer.

The application assembler defines *security-constraint* elements to specify which role or roles are authorized to each method. The security-constraint starts with a *web-resource-collection* element which defines one or more *url-patterns* and *http-methods* to which the constraint applies.

The security-constraint also contains an *auth-constraint* which lists one or more role names that are to be authorized to access the URLs and methods listed in the web-resource collections.

Finally, the security-constraint includes a *user-data-constraint* element to specify whether data flowing between the client and the container must be protected.

6.2.2 EJB container authentication and authorization

The Enterprise JavaBean (EJB) container supports different authentication and authorization methods from the Web container.

EJB container authentication

Whereas the J2EE specification has detailed information on the authentication mechanisms that must be implemented for the Web container, it does not specify authentication for the EJB container. It merely states that the EJB container must

have a means for the authentication of principals. In many cases this is of little consequence, since EJBs are typically invoked by servlet code (where the caller has been authenticated by the Web container) and not by end users directly.

EJB container authorization

The EJB container also supports two APIs for programmatic authorization. These come from the `javax.ejb.EJBContext` interface. They are similar to those supported by the Web container. `getCallerPrincipal()` is similar to the `getUserPrincipal()` of the `HttpServletRequest` interface.

`isCallerInRole()`, which takes a role name in the form of a `java.lang.String` as an argument, is similar to the `isUserInRole()` method of the `HttpServletRequest` interface.

The declaration of roles and authorization to methods are similar in concept to those found in the Web container, although the elements and tags have different syntax.

6.3 Resource authentication

In addition to authentication by Web and EJB containers, we must consider how authentication is to be accomplished for something called “resource factories.” A resource factory is an object that is used to create “resources.” Resources are means of accessing facilities (databases, for example) outside the realm of J2EE objects. The most common resources in use today are Java Database Connectivity (JDBC) connections to relational databases, J2EE Connectors to enterprise information systems such as transaction managers, and Java Messaging Service (JMS) connections to messaging and queueing systems.

When an application uses a resource factory, there usually must be a means of specifying to the container how authenticating information is to be passed to the resource factory and to the resources that it creates. J2EE handles this through a deployment descriptor element, *res-auth*. The *res-auth* element can take on two values. A value of `application` indicates that the application itself will provide authenticating information. This may be done, for example, by coding a `getConnection(userid,password)` call, where the authenticating information consists of a `userid` and `password` that the resource will be able to authenticate.

The other possible value, `container`, indicates that the container will provide authenticating information. In this case, it is the responsibility of the container to sign on to the resource using information provided by the deployer. How this information is provided is not within the scope of the J2EE specification, since it differs for each container’s implementation.

6.4 J2EE security and WebSphere for z/OS

In order to understand how J2EE security concepts are implemented, it may be useful to review the WebSphere Application Server for z/OS and OS/390 architecture.

On z/OS, WebSphere Application Server is designed to execute in a Parallel Sysplex. A WebSphere administrator defines one or more application servers into which J2EE applications are installed. An application server has a sysplex-wide scope. That is, instances of the application server may exist on one or more systems within a Parallel Sysplex. These instances are identical to one another, each one serving an identical suite of applications. Each application server instance consists of one started task called the *control region*, and one or more started tasks known as *server regions*.

As shown in Figure 6-4 on page 88, J2EE components, executing in Web containers and Enterprise JavaBeans™ (EJB) containers, reside in server regions. Each server region is started with an identity, or *userid*, that is defined in the System Authorization Facility (SAF) registry. During the life of the server region, this identity is contained in an Access Control Environment Element (ACEE).

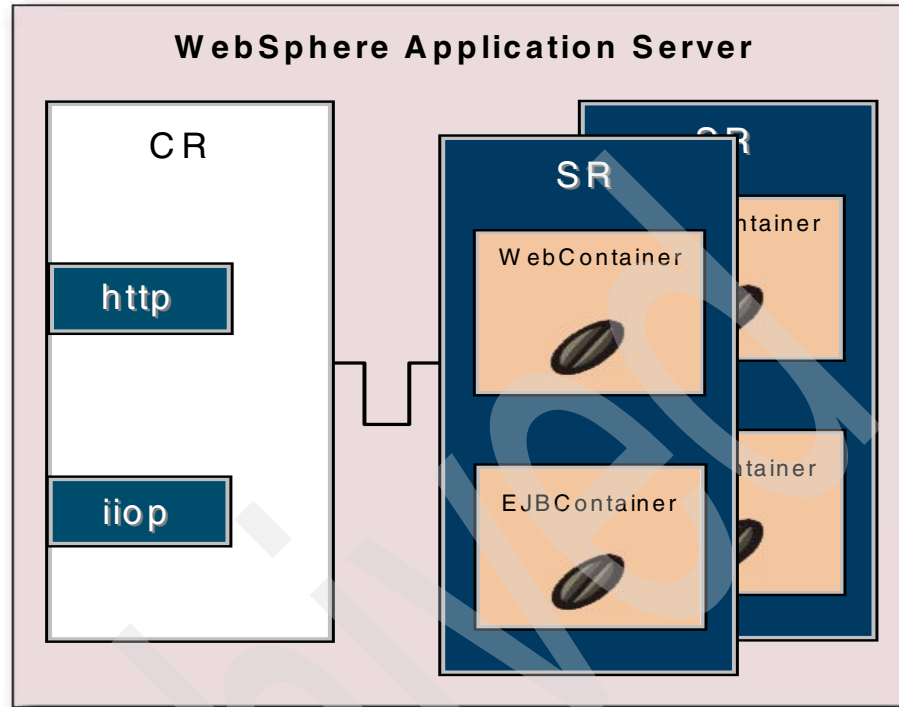


Figure 6-4 WebSphere server instance

The implementation of J2EE security in WebSphere Application Server for z/OS and OS/390 is built on the SAF. SAF is an interface defined by z/OS to allow programs to use a security product, such as z/OS SecureWay Security Server (RACF), to control access to resources.

As well as running in a server region, every request is associated with a Java principal whose principal name is either an SAF userid or a name that can be associated with an SAF userid. This identity is obtained either as a configured default or as the result of J2EE authentication. As Figure 6-5 on page 89 shows, there are always two identities present in the processing of a J2EE request by WebSphere Application Server for z/OS and OS/390: the identity contained in the ACEE, and the identity contained in the Java principal.

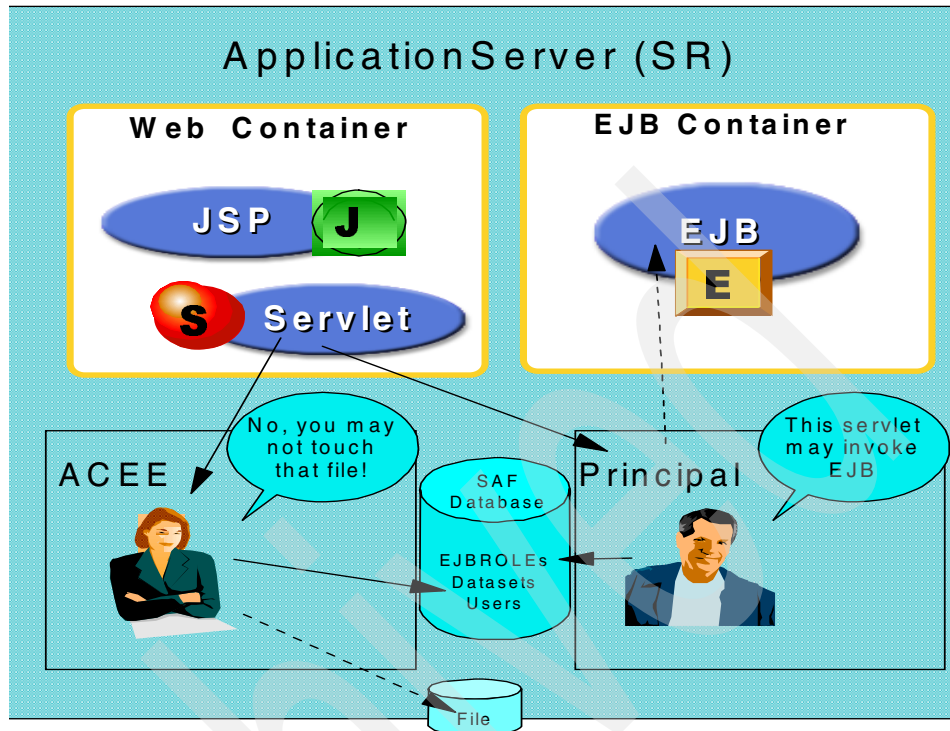


Figure 6-5 Two identities for every J2EE request

The Java principal is used in authorization checks that are performed by the J2EE container. For example, when one of the methods of a component is about to be invoked, the container determines whether the user represented by the identity belongs to one of the security roles allowed to invoke the method.

The z/OS operating system continues to use the ACEE when checking for authorization to resources that it manages, such as files and TCP/IP sockets.

This brings us to the implementation of roles in WebSphere Application Server V4.0.1 for z/OS and OS/390. Roles, as defined by J2EE, are not inherent in the SAF model. In order to implement J2EE roles, a new SAF class, EJBROLE, was created. (Do not be confused by the name EJBROLE. When this class was defined, it was thought that it would be useful only for EJBs. It is used, however, for J2EE roles in both EJBs and Web applications.)

When an application deployer uses a role in a component's deployment descriptor, the role name must be identical to the name of an EJBROLE profile. A security administrator defines EJBROLE profiles and permits SAF users or groups to the profiles. In order to be considered as eligible for a role, a user must

have read access to the EJBROLE profile or must be connected to an SAF group that has read access.

6.5 Directory services and LDAP

In our discussion, we have assumed that a user is accessing the system and that we need to authenticate them against a directory. A directory in this case is a list of user names and passwords. There are different directories available for authentication.

For traditional z/OS applications, RACF provides the directory facilities. Some applications, such as Lotus Domino, provide their own directory facilities.

An industry standard directory also could be used. As we have seen, today's e-business applications contain many components. If we allow each component to build and use its own directory, we will have to define the same information in many, possibly incompatible, places. To avoid that, the industry has come up with a common directory called the Lightweight Directory Access Protocol (LDAP). The details of LDAP V2 are defined in Internet Engineering Task Force (IETF) Request for Comments (RFC) 1777, and the details of LDAP V3 are defined in the set of IETF RFCs 2251 - 2256.

LDAP is being implemented in many directory products across the industry. It defines a communications protocol for a requestor (or client) to ask a directory server for information. A *directory* is a specialized type of database that can be used for looking up user names, or for any other information such as phone numbers, or books in a library. LDAP directories can be centralized on one server, or can be distributed in various ways across multiple servers.

From an infrastructure viewpoint, you need to understand which directories you require for user authentication. The general direction is to use LDAP for authentication, although not all products have gotten to that point.

The SecureWay Security Server for z/OS, a standard part of z/OS, contains an LDAP server. This server stores its directory data in a DB2 database, but it can also access data in the RACF database.

You should plan at which points in your solution you need to authenticate users and the directory options supported by the various products involved. Then see how you can minimize the number of different types of directories that you need to use. RACF, together with the LDAP server on OS/390, provides a good directory because both of them sit on the highly secure S/390 platform, and also because they probably already contain a lot of directory information.

The basic concepts of LDAP and their implementation on OS/390 are explained well in *Ready for e-business: OS/390 Security Server Enhancements*, SG24-5158.

6.6 Firewalls

A *firewall* is one or more computers you use to separate a safe network from a not-so-safe network, as shown in Figure 6-6. It is a defense mechanism between your internal secure network and the external non-secure Internet.

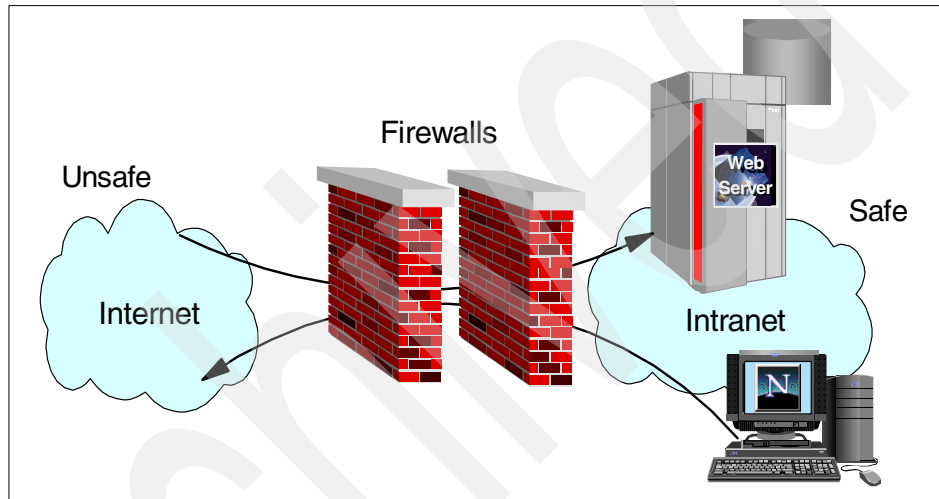


Figure 6-6 Firewalls

If you allow unprotected access from the Internet to your servers, you allow anyone in the world to send any data they like to any of your servers. This makes it very easy for potential intruders to understand your server topology, therefore making it easier to find a way into your systems. In addition, that approach requires you to secure every possible entry point on every possible server, and that is difficult to achieve and maintain.

The purpose of the firewall is to provide a first level of control over access into your computer systems. The firewall sits between the external network and your internal network, and limits what external users can do. Initially you would allow no traffic to come through. Then you would open up only those communication routes that you need. This limits the entry points on the internal systems that you need to control particularly carefully.

A firewall can protect you against a number of security exposures:

- ▶ It can allow internal users to freely access the non-secure network, while controlling external users' access to the secure network.
- ▶ It can stop network traffic that issues commands to understand the internal network, or tries to disable servers by overloading them with a large number of messages.
- ▶ It can convert network addresses so that the true IP addresses for your internal servers are not used in the non-secure network.
- ▶ It can restrict external traffic to only access a very limited number of servers and services in the secure network.

While we have talked of a single firewall, your firewall may consist of a number of servers, each protecting one or more gateways between your secure network and the Internet. In this case, you should think of the multiple firewall servers as one logical firewall. They should all implement the same company security policy.

Firewall types

There are many firewall technologies available, but they can be grouped into two major categories:

- ▶ Those that allow IP packets to be routed directly between two or more networks. These use techniques such as:
 - *IP filtering*, to check messages and either permit or deny the traffic based on source or target IP address, direction, protocol, port, and message content.
 - *Network Address Translation (NAT)*, to translate network addresses and thus hide internal addresses from the outside world.
 - *Virtual Private Network (VPN)*, to provide a secure network connection (or encrypted IP tunnel) between your secure networks, using encryption as the messages flow across the non-secure Internet.
- ▶ Those that disable IP routing, but relay data through specialized application programs. Messages cannot route directly. Instead, they are intercepted by the application program, may be changed, and are then passed on through a different connection. Examples of such applications are:
 - Proxy servers

These are specific to an application, such as FTP, Telnet or Web serving. They can reroute the traffic transparently to the client. Because a proxy is application-protocol aware, it is able to actively participate in the application protocol and, for example, request that the client end user enter a valid user ID and password for the machine on which the proxy executes. Only if the client end user authenticates successfully on the proxy machine is it allowed to pass through to the other network.

- Socks servers

These are application protocol-independent and generally work with all application protocols. Clients need to be aware of the socks server to support the initial handshaking with it. To avoid the need to put application code in all clients, that support can be built into the library used to compile and link the client application, or into the TCP/IP protocol stack. Such components are said to be *socksified*.

For more information on these topics, see *Stay Cool on OS/390: Installing Firewall Technology*, SG24-2046. Also see *Firewall Technologies Guide and Reference*, SC24-5835.

Firewall technology on z/OS

Firewall technologies were originally integrated into OS/390 Version 2 Release 5 and later as part of the OS/3909 Security Server and the eNetwork Communications Server. They were also made available on OS/390 Version 2 Release 4 with the OS/390 Firewall Technology Toolkit.

On z/OS they are provided in three separate ways: as part of the z/OS base code, part of the z/OS Communication Server, and also as part of the z/OS SecureWay Security Server.

The z/OS base code provides FTP proxy support and socks server (daemon) support. The z/OS Communication Server security features include IP packet filtering, IP Security and VPN, and Network Address Translation (NAT). The z/OS SecureWay Security Server provides Internet Security Association Key Management Protocol (ISAKMP) support, Command Line Configuration/Administration, and GUI Configuration/Administration.

The ISAKMP Server provides the capability to dynamically establish VPNs, negotiate VPN attributes, and dynamically manage VPN encryption keys. The ISAKMP server uses industry standard protocols to automatically exchange encryption keys, and therefore should operate smoothly with similar servers from other vendors. This gives you VPN capability without time-consuming, error-prone key management.

The hardware requirements for the firewall on z/OS are two network interfaces:

- ▶ One connecting to the secure (internal) network that the firewall protects
- ▶ One connecting to the non-secure, outside network or Internet

The network interface can be any communication hardware supported by the TCP/IP stack to make the network connection, such as any of the currently available OSA cards, CTC, or XCF.

You should research, and plan in detail, to match the needs of each firewall with the firewall technologies that are available to select the best fit.

Logical firewall structure

First you need to design your firewall structure. If your company has already connected servers to the outside world, you will almost certainly have a firewall in place, with well thought-out security policies. In that case, you can use that firewall structure for additional services. As you do so, you will need to review those security policies for any changes that may be necessary, and make sure the combination of the firewall and the new services provides adequate protection.

So far, we have only considered the Internet to be the unsafe place, while your internal network has been considered the safe place. However, that may be an oversimplification in some situations. For example, consider a research department that works with highly confidential information. In such an environment, you may want to protect that research department from your regular users by implementing a firewall between your regular internal network and the network in your research department, in this case considering your regular users to be the potential “bad guys”. This way multiple firewalls can be implemented with different levels of protection.

The most common use of multiple firewalls is the structure known as a Demilitarized Zone (DMZ), as shown in Figure 6-7 on page 95.

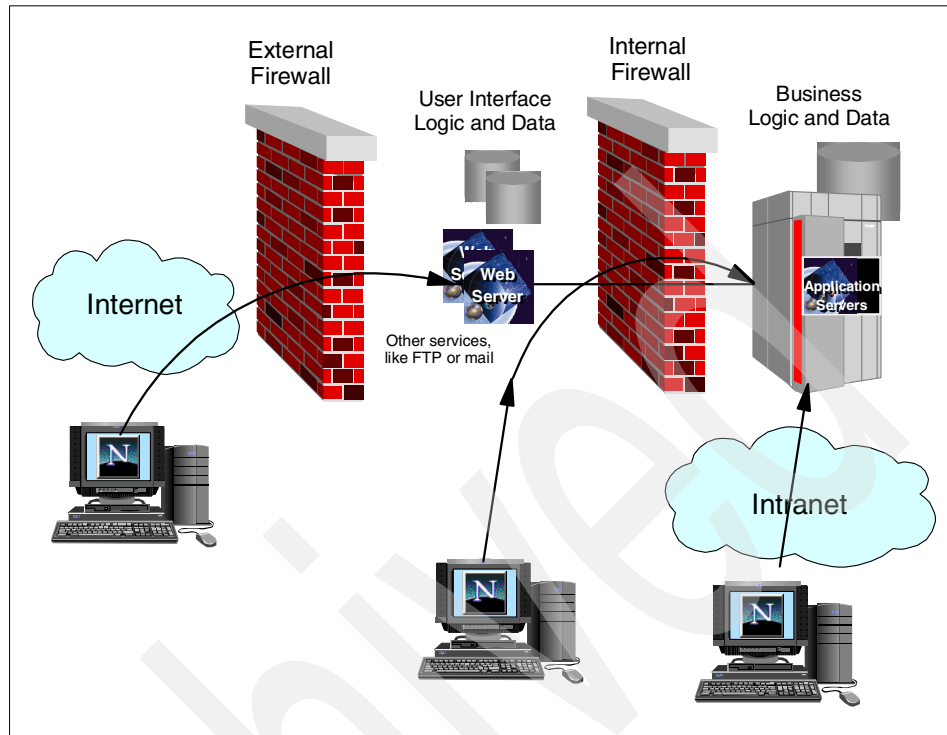


Figure 6-7 Firewall structure with a Demilitarized Zone (DMZ)

In the DMZ design, Web servers (providing presentation services and other static content) is placed behind an *outer* (external) firewall, and the other servers (supporting business logic) are placed behind a second, *inner* (internal) firewall.

The Web server sits alone in the DMZ, handling requests from the Web and passing them along to the secure intranet network. The Web server contains the logic and data necessary to construct the user interface of the e-business applications. The Web application server and internal business systems behind the inner firewall contain all the remaining business logic and data of the application.

The benefit of the DMZ approach is that your key business systems are protected behind *two* firewalls, and all traffic in the DMZ is controlled by the Web server. This provides an additional level of control and security.

This separation of roles has implications for how you structure your applications and how the various parts communicate, especially when using objects. You must also consider trade-offs in your design between the performance of your application and risk.

One important trade-off to evaluate is the degree to which you will exploit the performance benefit you can attain by caching frequently requested data inside the DMZ, rather than retrieving it from back-end systems each time it is requested. In any case, machines in the DMZ are known to be at higher risk and are managed accordingly.

Managing workload

In this chapter, we discuss an important issue of e-business, namely management of workload. This is an area in which z/OS has a long and outstanding history among the operating systems on the e-business market today.

7.1 Overview

There are many cases where a company's first foray into e-business was of such disastrous proportions that it made headlines in the evening news. The cause is often a lack of planning and tools. Planning: how are we prepared to detect problems and how are we prepared to deal with them. Tools: the gadgets that let us know that what we have built will work, and continue to work.

There are any number of solutions available which can take a business to the Web. In many cases, a complete solution is not available for a single platform. This results in a mixture of servers and operating systems all talking their own language and providing performance information in various cryptic forms. There is nothing to relate a piece of data from one server to a piece of information from another server.

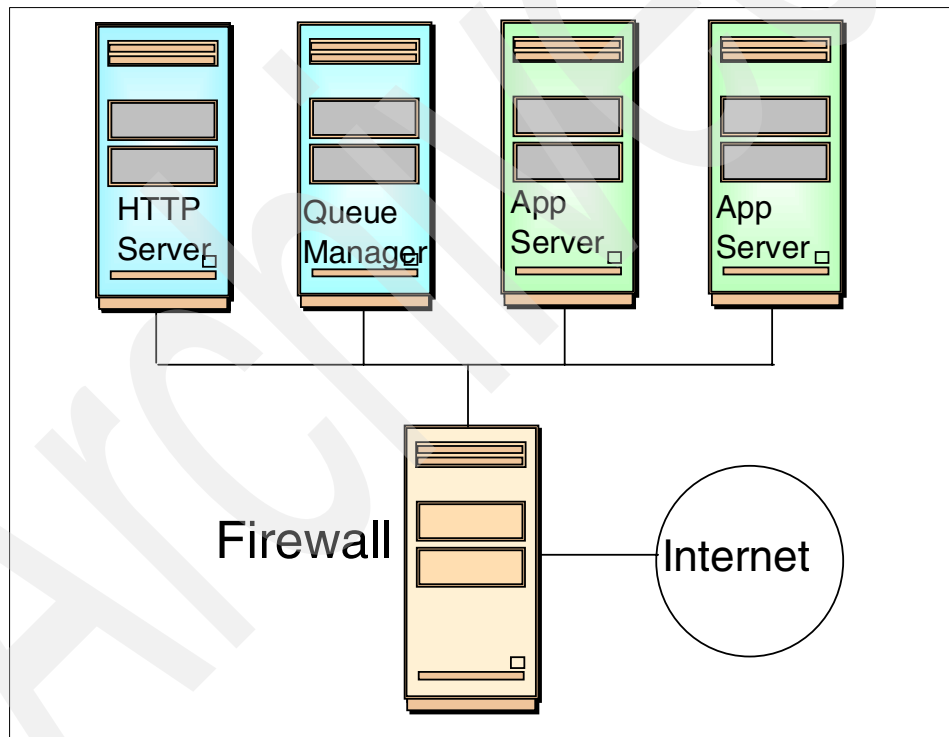


Figure 7-1 Network solution, a planning dilemma

Most e-business enablement solutions exist on the model of a server/processor on a network performing a single function. One function, one server. The upgrade path in this arrangement is hit and miss. When a particular server impacts the system, more memory is tried, more disk cache, a faster hard disk, and then the entire machine is upgraded. As business grows, we chase more and more of these failures around the network until, finally, the network is the problem.

Add to all this: Hackers, denial of service attacks, and the legal privacy obligations inherent to e-business—and there is a need to limit the availability of information to unwanted visitors. Firewalls are the standard for the first line of defense; after that, things tend to get hazy. There are many cases where firewalls are used in front of every server in a network. Every solution implements security in a manner deemed appropriate for the product. There is little opportunity to add barriers that exist in a consistent fashion across the myriad of networked components.

7.2 z/OS tools

z/OS provides the tools and the functionality to address many of the issues normally not considered when planning e-business. The ability to quantify the relationship between one service and another and to make adjustments to their performance is supported through Resource Management Facility (RMF), System Management Facilities (SMF), and Workload Manager (WLM).

RMF, through its various reports and vast data collection facilities, can help identify and resolve contention between components. It also provides real-time monitoring of the system with graphic measures and alerts.

SMF provides a detailed historic view of activity on a task by task basis.

WLM provides the ability to adjust service on the fly, based on predefined rules and service policies.

The ability to measure systems in real terms such as storage, CPU, and I/O provides a simpler upgrade path. There is little sense in monitoring response time if we do not understand the components that make up response time. Measurement should occur when what is being measured can be bought. There is no ability to walk into a store and say you need two thousand transactions an hour. That's not what they sell. Faster or more CPUs can be bought, memory can be bought, and disk drives can be bought. If a measurement tool does not provide information in terms of what can be purchased, it has little value. With

z/OS and WLM balancing, the entire workload across a single platform, growth in storage, CPU and I/O can be tracked and the right upgrades added before a problem occurs. This is preferable to chasing the weakest link in a network arrangement.

With some or all of the server functions running under z/OS, communications between the components are no longer dependent upon a network eliminating the need for entire network infrastructure upgrades. They also remove the barrier of the network being a component of throughput. Servers talk to each other at memory-to-memory speeds. Eliminating the network layer also removes a data exposure level that is very attractive to less desirable Web visitors.

Security is covered in Chapter 6, “Security management” on page 75. However, z/OS running on zSeries processors can offer barriers against hackers, which do not exist in other architectures.

A company’s entry into the e-business community can be controlled through the use of stand-alone processors, LPARs, single image, and Parallel Sysplex. Each one can serve a need for security by providing various levels of isolation and growth.

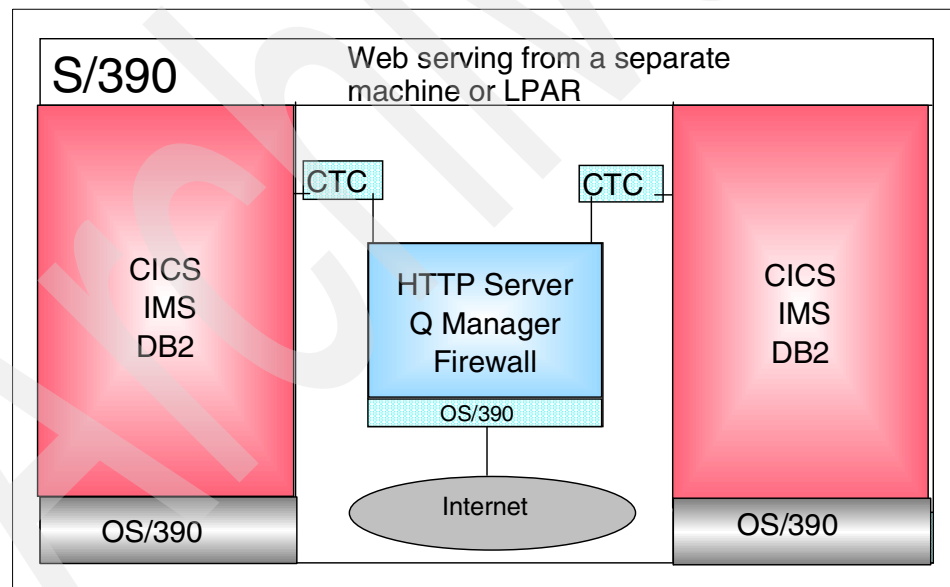


Figure 7-2 Isolating the Internet with LPAR or separate machines

The stand-alone processor and LPAR options enable an organization to control the exposure of sensitive information. By limiting the amount of shared data and connectivity to the background systems, architectural boundaries can be built that ensure a level of comfort. There are several other advantages to this arrangement besides security. Different software levels and maintenance may exist between the connected systems. In the early days of testing the e-business products, modifications to the LPAR or system running the products will not have any impact upon existing production applications.

Single-image and Parallel Sysplex usually take the shape of adding an e-business solution to existing platforms. Security can depend heavily on the interface between UNIX services and the MVS component that hosts the servers. The firewall, the lack of network connectivity between server components, and the introduction of a consistent security methodology across the entire platform all reduce the potential for undesirable attacks.

7.3 Workload Manager

WLM is the component of z/OS that provides the tuning knobs and buttons. Performance is adjusted in one of two modes: *compatibility* or *goal*.

7.3.1 Compatibility mode

This mode of tuning is traditional in a sense. Before z/OS became z/OS, MVS and OS/390 depended on the use of CPU, IOC, SRB and MSO along with a myriad of parameters and options requiring a highly skilled group of individuals to manipulate them. Translating business requirements into these values often resulted in errors and questionable experimentation. The best that could be accomplished was “close”.

As business and machines grew, so did the mixture of products necessary to support them. Compatibility mode tuning became an array of parameters that tried to define how an application was suppose to run instead of defining how fast it should run. Overcommitting storage to a CPU-starved subsystem was a typical “shot in the dark” type of tuning exercise.

Whenever a new subsystem or task was added to the system, its impact could not be measured, resulting in “best guess” approaches to its introduction into the melee. Even seemingly minor changes could have devastating impacts on the overall performance of the system.

With the introduction of new subsystems came the limitations of compatibility mode. With the system broken down into only three major components (Batch, TSO, and Started Tasks), the ability to isolate and tune an individual component is increasingly more difficult.

Many products have their own tuning knobs and dials that work from within. The ability to break down work and categorize it within the context of the larger program cannot be addressed in compatibility mode. Internal tuning can also conflict with external tuning parameters, making the resolution of a performance issue even more difficult.

In systems where multiple processors are available, there is no ability to distribute the workload evenly across the systems. Compatibility mode is limited to the OS in which it runs. It doesn't have the ability to look across a sysplex for resources when the system it is on is under stress.

7.3.2 Goal mode

Introduced way back in MVS/ESA 5.0, goal tuning changed the framework by which we define our performance needs. Translating the business need "ASAP" into computing is accomplished by the term "velocity". If one type of work is more important to the business than another in real-time terms, the speed of the two tasks can be tuned accordingly. WLM permits the dynamic updating of tuning values to react when the desired balance is not being accomplished.

WLM uses fewer knobs and buttons to accomplish the objectives set by business needs. Paging, swapping, storage, IO, and CPU are all dynamically balanced by WLM in order to facilitate a task making its performance objective. A task with high storage demands verses a task with high IO impact can be balanced according to real needs, not fixed values as in compatibility mode.

High availability transaction processors such as CICS, IMS and DB2 provide internal performance reporting information to WLM. With the ability to classify smaller units of work within the larger server, workflow can be managed at an increasingly granular level.

In many cases, particularly e-business, WLM's ability to balance response goals with throughput presents the opportunity for services to be started and stopped as they become necessary. The WebSphere Application Server launches application server address spaces according to the application environment. WLM can launch additional application servers based on workload and throughput objectives. In compatibility mode, system operators would need to start tasks manually, through the console.

For both performance and security purposes, each request type of work in the system is assigned an application environment. The application environment defined when installing a transaction (or a group of transactions) relates directly to an Application Environment name in WLM. Application Environments can also be defined in RACF with the same definition. This combines performance with security for a consistent understanding of an application's role throughout the system.

WLM provides a single tool for balancing workload across a Parallel Sysplex by using global performance policies. WebSphere queue servers can be dispatched across all the parallel processors in a sysplex, increasing availability and providing growth options not available in compatibility mode. Performance can be distributed on a basis of need with no barrier at the upper capacity of a single system in the sysplex.

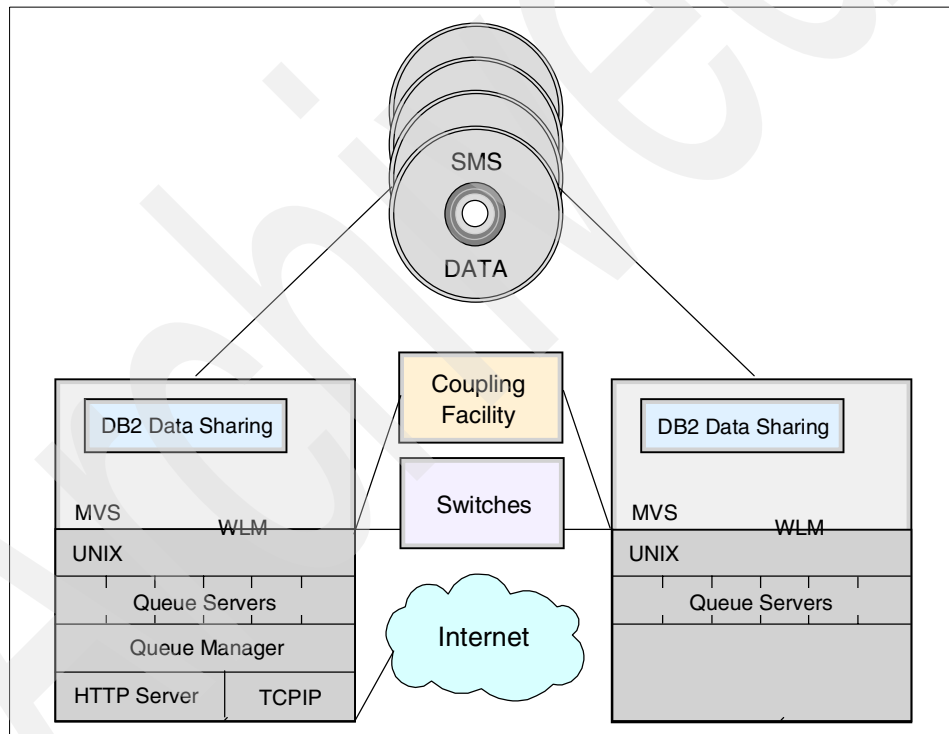


Figure 7-3 Multiple servers started by WLM across a sysplex

The introduction of new subsystems beyond the Batch, TSO, and STC classifications of compatibility mode creates opportunities to further sift the confusion out of performance management. Compatibility mode performance subsystem groupings are predefined within WLM. Role-your-own categories,

which may be introduced beyond the predefined groupings, provide support for an ever increasing, scalable model. Moving from a slower processor to a faster one, changing the weight values for an LPAR, or implementing a sysplex can be accomplished without the need to redefine the performance objectives of the overall system.

7.4 WLM and HTTP Server modes

There are three approaches that can be taken to enabling e-business on z/OS. Combined with WLM, each approach can fit a particular business need.

7.4.1 Stand-alone server

This mode is typically for HTTP server-only implementations. Whether it exists on a separate machine, within its own LPAR, or as part of a larger operating system, its main role is to provide a limited exposure to the internet. It is often used to provide some informative pages about the business along with a few contact names and phone numbers, and this is all that is necessary.

WLM handles this environment in both compatibility and goal mode. In compatibility mode, the HTTP server would be tuned under the SYSSTC workload grouping. By using the fixed values specified in compatibility mode, an increase in response time would not be reported in terms of needs not being met. Constraints could be hit and miss to diagnose. The impact of one workload on another would be difficult to quantify.

In goal mode, WLM would not be used to classify requests by application environment specifications. A response time goal in terms of a time target and a velocity would balance all of the quantifiable elements such as CPU, storage, and I/O. In an LPAR, changes to weighting factors and storage allocations can be calculated as a direct result of RMF/WLM reports.

7.4.2 Scalable server

In order for a system to scale, a queue manager with the ability to quantify work in terms that WLM understands, and to distribute the work in the same fashion, is necessary. WebSphere application server is designed with this particular goal in mind.

WebSphere is started manually or by automation as the queue manager. Queue servers are spawned as a result of different Application Environment specifications or WLM adjusting workloads to accomplish throughput objectives. By using the application environment, work can be grouped by similar data

access requirements, performance needs, or just the opposite, separated to isolate workloads from each other. If requests for a certain application environment are infrequent, WLM may allow a spawned queue server to shut down, permitting a future, infrequent request to bypass the queuing process.

WLM in Goal mode is a prerequisite for WebSphere application server version 4. In goal mode, WLM and z/OS take a more dynamic stance. The WebSphere Control Regions receive requests, which it classifies, and queues to WLM. WLM then manages queuing the work to Server Regions. This also involves starting up extra Server Regions if required, and shutting down underutilized ones.

7.4.3 Multiple servers

The combination of a stand-alone server and a scalable server, or even multiple scalable servers, can work together to improve scalability and security throughout the system.

An HTTP stand-alone server functioning as the gateway to scalable servers creates a barrier at which user authentication can be handled without exposing the entire system to unwanted eyes. The stand-alone server can provide URL rerouting (links) to other servers listening on different ports in the same TCP/IP stack, or on different stacks. User authentication at this point in the connection path permits the differentiation between Internet and intranet/extranet authorization and maintains the separation throughout the life of a connection. Although there is not true intranet/extranet implementation because there is no net, the same boundaries which are normally defined for these environments still exist in WLM and RACF.

When designing a scalable system of any sort, it is best to keep the number of application environments to a minimum. Starting too many queue servers as a result of many application environments can have a negative impact on the system. Defining minor application environments with terms like fast, medium, and slow along with a separation of the major application between production and test is often a good place to start. By scaling the velocity of each category to accomplish a specified business goal, the impact is driven strictly by demand. A production or fast goal may cause the spawning of two or three queue servers for one application environment whereas test/fast may never have the need to spawn one. Setting reasonable limits for accomplishing goals is made easier by clearly-defined workload categories.



Part 2

The components in detail

In Chapter 3, “The components of e-Business” on page 43, we briefly touched the components of e-business on the z/OS operating system. In this part we discuss the individual components and their roles in detail.



The IBM HTTP Server

This chapter explains the most important concepts related to the HTTP Server on z/OS and its features.

8.1 Introduction and role in e-business

The HTTP server plays a central role in almost any e-business environment. This process receives the requests from clients and either serves a file itself, or redirects the request to an external program (CGI), or to the WebSphere Application Server plugin and perhaps eventually to an application server such as J2EE or MOFW.

When a client connects to the HTTP server, via a browser on a work station, the Universal Resource Locator (URL) is used to locate both the server and the specified page to be served:

```
http://www.anywhere.com/index.html
```

Although the primary use of the HTTP server is to send HTML pages to a browser, either statically or dynamically generated, the file being sent could be anything, as long as the client uses the HTTP protocol to communicate with the HTTP server, and, obviously, as long as the client knows what is being received.

8.2 Server modes

There are three modes to run the IBM HTTP server in z/OS:

- ▶ Standalone Server mode
- ▶ Scalable Server mode
- ▶ Multiple Server mode

See 7.3, “Workload Manager” on page 101 for more detailed information concerning how the WorkLoad Manager (WLM) interacts with the WebSphere Application Server environment. WLM also interacts with the HTTP server to provide scalability.

It is important to understand these execution modes. They are discussed here so you will have an overview of the characteristics of these modes and how workload management, simple network management protocol, and system management facilities work.

Stand-alone server

This mode is typically for simple HTTP server-only implementations.

It is often used for simple Web sites that provide static information about a business and can also be used for a corporate intranet with limited function.

Its main role is to provide a limited exposure to the Internet. Since it has a finite capacity based on resource definitions, it may not be able to adequately respond to changes in demand.

For more information, see 7.4.1, “Stand-alone server” on page 104.

Scalable server

This mode is typically for interactive Web sites where the traffic volume can decline and increase dynamically and the ability to react is needed.

It is also meant to be used in a more sophisticated environment where servlets and JSPs are invoked and performance demands have to be met. The WebSphere Application Server Environment is designed with this particular goal in mind.

The HTTP Server can be started as a queue manager (Control Region). Queue servers (Server Regions) are spawned as a result of WLM adjusting workloads to accomplish throughput objectives.

For more information, see 7.4.2, “Scalable server” on page 104.

Multiple servers

The combination of a stand-alone server and a scalable server, or even multiple scalable servers, can improve scalability and security throughout the system.

An HTTP stand-alone server functioning as the gateway to scalable servers creates a barrier at which user authentication can be handled without exposing the entire system to unwanted eyes. The stand-alone server can then provide URL rerouting (links) to other servers listening on different ports in the same TCP/IP stack, or on different stacks.

For more information, see 7.4.3, “Multiple servers” on page 105.

8.3 Ways to access the HTTP server

All HTTP servers are meant to be connected to the Internet or an intranet. Therefore, it is logical that the way to access an HTTP server is always TCP/IP, which forms the backbone of Internet addressing.

Over TCP/IP, though, there can be other protocols. The most common one is the HyperText Transfer Protocol (HTTP).

Communication protocols

The following protocols are supported by all HTTP servers:

- ▶ HyperText Transfer Protocol (HTTP)

HTTP is an application-level protocol. It is a generic, stateless protocol that can be used for many tasks beyond its use for Hypertext, although this is its more extended and general use. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

Although connectionless, the HTTP 1.1 specification defines a persistent connection concept that makes the HTTP server hold a connection for a while, therefore making it more efficient if more requests are coming from the same client.

- ▶ HyperText Transfer Protocol Secure (HTTPS)

HTTPS is a Web protocol that encrypts and decrypts user page requests as well as the pages that are returned by the HTTP server.

HTTPS uses the Secure Socket Layer (SSL) as a sub layer under the regular HTTP application layer. By using powerful encryption algorithms, SSL makes possible the secure transport of sensitive data.

HTTPS uses port 443 instead of HTTP port 80 in its interactions with the lower layer, TCP/IP. The use of two different ports allows for the use of a single IP address for a given e-business Web site and makes the switching between secure and non-secure modes fairly simple.

8.4 How the HTTP Server works

Although its original function was mainly to serve static HTML pages (or files) to a browser, today many of the HTML pages that it sends to the client are dynamically built by other components.

The process for serving static files (a) can be summarized in Figure 8-1. The figure also shows an overview of the processes for both dynamic files “servlets and JSPs” (b) and for CGI applications (c).

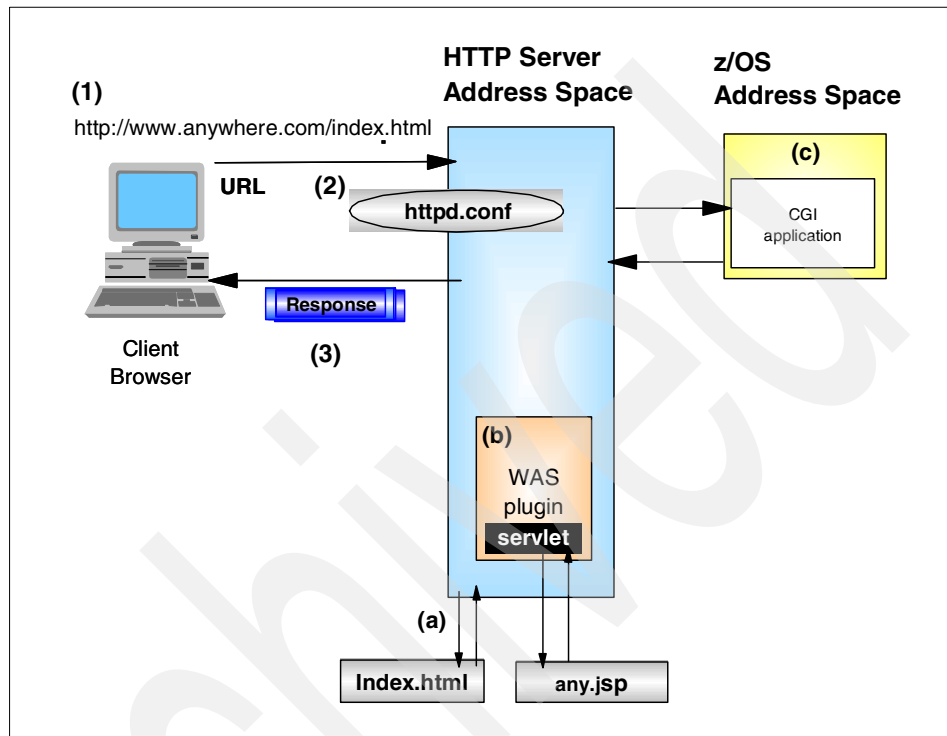


Figure 8-1 How the HTTP server works

1. The HTTP server receives the client request (usually after passing a firewall).
2. By comparing the incoming URL with directives in the `httpd.conf` file, the HTTP server routes the request in one of the following three ways:
 - a. If the request is for a static file (as above) the HTTP server will return the static file to the requestor.
 - b. If the request is not related to serving a static file, but is output to be created dynamically (servlet or JSP), the HTTP server passes the request to the WebSphere Application Server plugin.
 - c. If the request is for a CGI application, the request is passed to the application.
3. All output and imbedded files are returned to the browser client by the HTTP server.

8.5 Passing control from the HTTP server

There are two ways in which the IBM HTTP Server for z/OS can pass control to another program: the Common Gateway Interface (CGI) programs or plug-ins.

CGI Interface

The Common Gateway Interface (CGI) is a standard way for a Web server to pass a Web user's request to an application program and to receive data back to forward to the user.

When the user requests a Web page (for example, by clicking on a highlighted word or entering a Web site address), the server sends back the requested page.

However, when a user fills out a form on a Web page and sends it, this usually needs to be processed by an application program. The Web server typically passes the form information to an application program that processes the data and may send back a confirmation message.

This method or convention for passing data back and forth between the server and the application is called the Common Gateway Interface (CGI). It is part of the HTTP protocol.

Although still used by many applications and very easy to use, it doesn't fit very well into the e-business model that your business should try to build for the future.

One of the reasons for this is that, for each CGI application started, a separate z/OS address space has to be created. This can be extremely costly in terms of CPU utilization.

WebSphere plug-in

Plug-ins are programs that run in threads inside the HTTP server process. This is the basic difference from CGIs and is why they are much faster.

Each plug-in program goes through the following processes:

ServerInit	An initialization process - only the first time it is called
Service	A service process - for each new request passed
ServerTerm	A termination process - when it has to be removed from the HTTP server

All this happens inside the HTTP server address space, and unlike CGI there is no overhead of building a new address space for each request.

To write plug-ins for the IBM HTTP Server for z/OS, you must use a set of HTTP server APIs called *Go Web Server API (GWAPI)*.

Several products come as a plug-in to the IBM HTTP Server for z/OS, but there is one that is especially important — the IBM z/OS WebSphere Application Server.

Both the WebSphere 3.5 and 4.0.x plug-ins can be used to run servlets and JSPs in the HTTP Server address space. The WebSphere 4.0.x plug-in can also be used to connect to servlets and JSPs that are running in a Web container in a J2EE Application Server address space.

For more information about the WebSphere plug-ins, see Chapter 9, “WebSphere Plugin environment” on page 121.

The plug-in environment provides a bridge to the future and the use of EJBs and Application Servers. It allows you to use your current e-business environment as a basis for expansion, migrating existing servlets and JSPs to Web containers that reside in the same address space as your EJB containers.

8.6 How the CGI interface works

The server program can be written in any language supported by the server platform. Although the CGI interface is common to most server platforms, portability of executable code and source is dependent upon the compatibility of the server platforms. A CGI written to run on a UNIX server may not execute on an NT or z/OS platform.

A separate CGI address space is required on the server for each type of browser request. This creates an efficiency and maintenance dilemma in environments with a rich variety of services and processes.

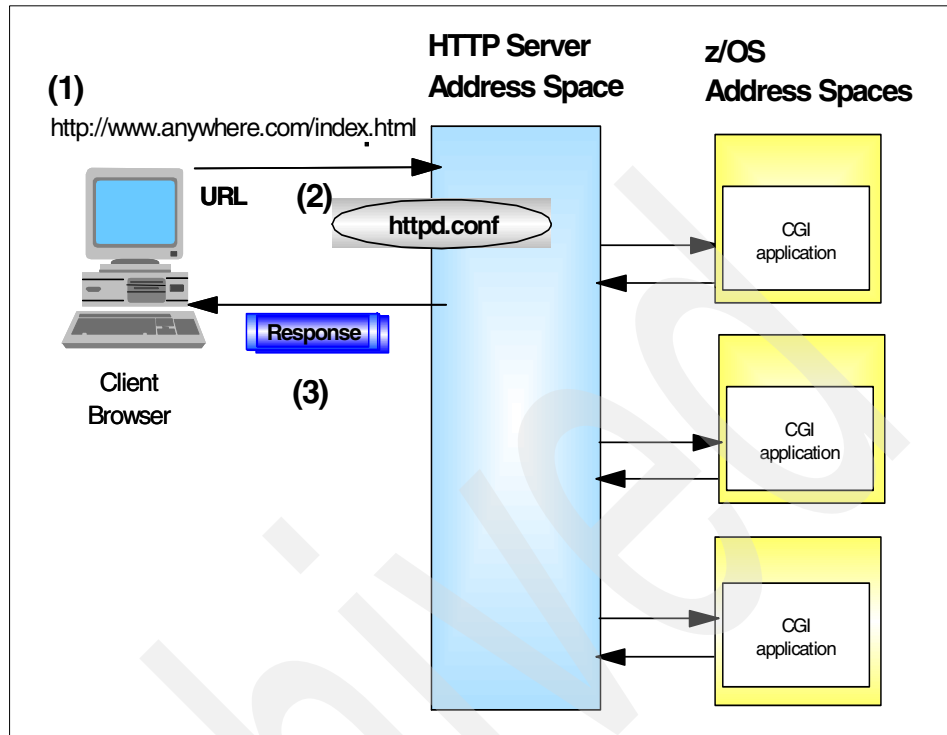


Figure 8-2 How the CGI works

The complexity of CGI executables must be kept to a minimum to maintain server availability. Functions requiring access to large amounts of data or triggering complex back-end processes can tie up the server to the detriment of other users.

If the desired functionality of the browser request is small, CGIs can be very efficient. However, their limited portability and applicability to more complex transactions makes them less desirable than most other alternatives.

FastCGI

FastCGI was created by Open Market Inc. in an intent to overcome CGI performance problems. Basically, FastCGI is a program that manages multiple CGI requests within a single process (address space on z/OS), saving many program instructions for each request. With CGI, each instance of a user requesting a service causes the HTTP server to open a new process that takes control, performs the service, and then is closed. With FastCGI, the overhead for one process is shared among all currently processing requests.

In the z/OS environment, CGIs cause an address space to be created and destroyed for every request. With FastCGI, once an address space is created, it remains there to process other requests.

It is more efficient to run applications with FastCGI than with CGI. And both CGI and FastCGI programs are easy to build and implement.

8.7 e-business-related functionalities

Although most of the HTTP servers share the same capabilities, the following is a discussion of the main functionalities of the IBM HTTP Server for z/OS Version 5.3, since some of its features are z/OS-specific.

8.7.1 Basic functions

The basic functions include:

- ▶ EBCDIC/ASCII file access

z/OS systems store files in EBCDIC encoding, but the standard encoding that goes through the Internet is ASCII. When using the IBM HTTP Server for z/OS, you don't have to worry about this because the HTTP server takes care of conversions that have to be made when sending files to the Web.

- ▶ SMF facilities

As part of the z/OS features, the IBM HTTP Server for z/OS can produce SMF records that can be retrieved later to do performance and usage analysis.

- ▶ Tracing and logging

The IBM HTTP Server for z/OS comes with a complete set of logging, tracing, and reporting capabilities that allow you to keep track of every request that comes to your server.

- ▶ Server Side Includes (SSI)

Server-Side Includes allow you to insert information into CGI programs and HTML documents that the server sends to the client. A Server-Side Include is no more than a variable value (for example, a file “Last modified” date) that a server can include in an HTML file before it sends it to the requestor. It is a nice feature, *but be careful because it can have a serious performance impact*, if turned on in the httpd.conf file while not actually being used.

- ▶ Simple Network Management Protocol (SNMP) MIB

A network management system is an application that runs continuously and is used to monitor, reflect status of, and control a network. Simple Network Management Protocol (SNMP) is the network management standard. It communicates management information with devices in a network.

The IBM HTTP Server for z/OS provides an SNMP Management Information Base (MIB) and SNMP subagent so you can use any SNMP-capable network management system, to monitor your server's health, throughput, and activity. It can then notify you if specified threshold values are exceeded. You can now proactively tune or fix server problems before they become server outages.

- ▶ Cookies support

HTTP is a stateless protocol, that is, a Web server cannot remember anything from the previous connection of a certain client. A *cookie* is information that a Web site puts on the hard disk of the client computer so that it can remember something about that workstation at a later time.

Cookies are commonly used to rotate the banner ads that a site sends so that it doesn't keep sending the same ad as it sends a succession of requested pages. They can also be used to customize pages for a user based on the browser type or other information the user may have provided the Web server. Web users must agree to let cookies be saved for them, but, in general, it helps Web sites to serve users better.

- ▶ Multi Format Processing

Multi Format Processing is an HTML file selection mechanism used for personalization of Web pages. For example, if a company wants to serve its home page with each country's language on one site, it can do that using this mechanism. When a request is sent from a browser to a Web server, the browser sends header information along with the request that informs the Web server of the characteristics of this particular browser. The response received from a browser is called the *accept header*. The Web server can make use of the contents of the accept header to select the appropriate document for the specific client (browser).

- ▶ HTTP-related features

- Persistent connections

As stated before, HTTP had the problem of establishing a new connection every time a new request was made from a client. Persistent connections try to get around this overhead by staying “alive” during a certain amount of time.

- Virtual hosts

Virtual hosts allow you to run one Web server while making it appear to the clients as if you are running several. You can run one instance of the server and assign each customer to a different host. In the Domain Name Server, you define your hosts and associate them with the IP address of the server. You can then configure the server to serve a different set of information, depending on the host for which the request is made. Besides this, you have other options, such as having multiple servers sharing the same IP address, or a single host using multiple IP addresses or multiple hosts using multiple IP addresses. Any combination is possible.

8.7.2 Security functions

Security on z/OS is discussed in more detail in 6.1, “Overview of J2EE security” on page 76. Here we mention those features that are directly related to the IBM HTTP Server for z/OS.

- ▶ Thread level security

An independent security environment can be set for each thread running in the HTTP server, which basically means that every client connecting to the server will have its own security environment.

- ▶ HTTPS/SSL support

The HTTP server has full support for the SSL protocol. Watch out for possible performance implications when this is set to “on.”

- ▶ LDAP support

The Lightweight Directory Access Protocol (LDAP) specifies a simplified way to retrieve information from an X.500-compliant directory in an asynchronous, client/server type of protocol. LDAP support allows the IBM HTTP Server for z/OS to authenticate a user who is accessing a protected URL

- ▶ .Certificate authentication

As part of the SSL support, the HTTP server can use certificate authentication and act as a certificate authority.

- ▶ Proxy support

The HTTP server can act as a proxy server. You cannot, however, use the Fast Response Cache Accelerator (FRCA).

8.7.3 File caching

Performance can be significantly increased using any of the following file caching possibilities:

- ▶ HTTP server caching HFS files

At HTTP server startup time, you can preload the frequently referenced HTML documents, GIF files, and other Web files by using the `CacheLocalFile` directive.

- ▶ HTTP server caching MVS data sets

Traditional MVS data sets can also be cached through an option of the plug-in that allows the HTTP server to access them.

- ▶ UNIX System Services caching HFS files

z/OS Version 2 Release 4 introduced a new file cache command, which is used to cache selected read-only HFS files into the kernel address space.

Note: If you modify cached data, that data is deleted from the cache and all further data access is from disk.

Although the `filecache` command is a function provided by UNIX System Services, we can use this function to complement the IBM HTTP Server for z/OS cache built with the `CacheLocalFile` directive. The `filecache` command can be used to cache DLL and CGI program executables.

- ▶ Fast Response Cache Accelerator (FRCA)

This is a cache mechanism automatically loaded during server operation. You are not required to list the files to be cached in your server configuration file. Dynamic content and protected pages are not cached.

It is very fast because it uses the TCP/IP stack to cache the pages so that requests are handled without traversing the entire kernel or entering the user space.



WebSphere Plugin environment

In this section, we discuss WebSphere Application Server Plugin versions 3.5 and 4.0.1 and how they can be used to serve JSPs and servlets.

9.1 Introduction and role in e-business

The IBM z/OS WebSphere Application Server Plugin is a Java application server. Its job is to receive requests from clients through the HTTP server and then send back a response produced by a Java program (a JSP or servlet). Essentially it “plugs into” the HTTP server and runs in the same address space.

While the HTTP server is generally used to serve static HTML pages or connect to a CGI application, the WebSphere Plugins (3.5 and 4.0.1) are used to serve servlets and JSPs.

See 2.6.4, “Servlets” on page 29 and 2.6.5, “JavaServer Pages (JSPs)” on page 30 for more information about these technologies.

Servlets, running in the WebSphere Plugin, and essentially providing the formatting services, can also talk to EJBs running in an J2EE Application Server container in a separate z/OS address space. The EJBs will provide the actual business logic and connection to the backend systems.

See 9.3.2, “Using servlets in the Plugin to talk to EJBs” on page 125 for more information concerning the local type of configuration

9.2 Where to run your servlets

Starting with WebSphere 4.0.1 you now have a choice where you want to run your servlets. Aside from running the servlet in the WebSphere Plugin (locally), you can also choose to run your servlets remotely in a Web Container residing in the same address space as your EJBs.

See 10.3.1, “Using the WebSphere Plugin” on page 133 for more information concerning the remote type of configuration.

Local vs. Remote

The ability to run servlets locally provides you with a means to migrate using your existing environment with minimum impact. You can continue business as usual and move into the world of EJBs.

The ability to run servlets remotely in a Web container allows you to minimize the overhead when talking to EJBs. You can keep all the pieces to your application within a single z/OS address space.

WebSphere 4.0.x is recommended as a plugin when processing requests for servlets and JSPs running remotely in a Web Container in a J2EE Application Server in a separate z/OS address space. The WebSphere Plugin 3.5 cannot be used in this configuration.

9.3 How the WebSphere Plugin works

On the z/OS platform, both WebSphere 3.5 and 4.0.1 are available as a plugin for the IBM HTTP Server.

Figure 9-1 on page 123 shows a simplified version of how a request for a local servlet or JSP is directed to the WebSphere Plugin and a response is returned to the client.

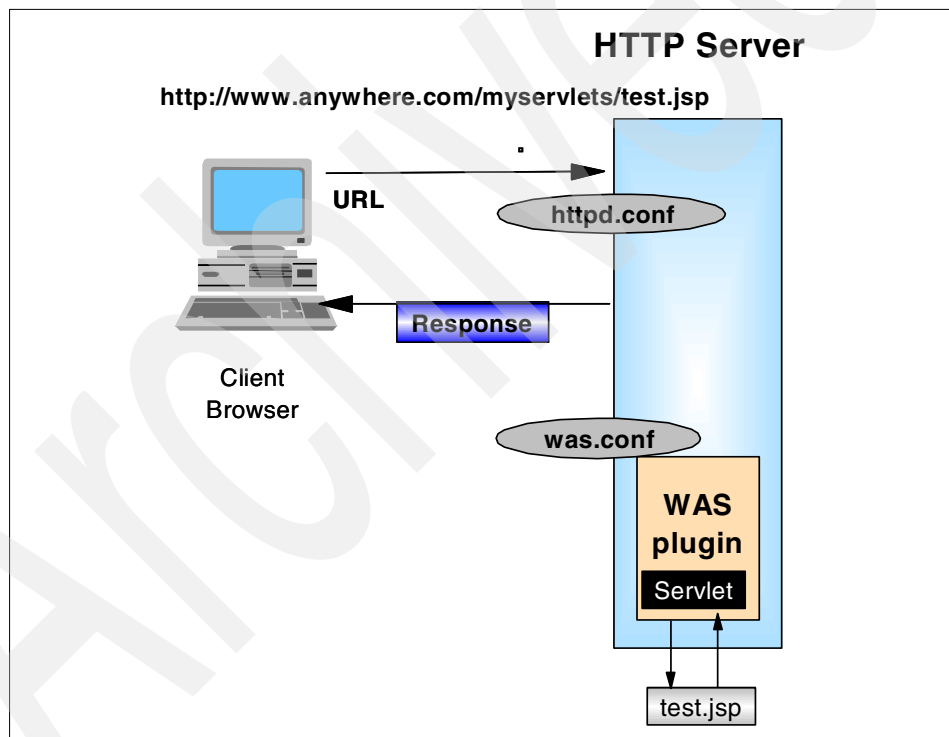


Figure 9-1 Accessing servlets using the WebSphere Plugin

1. The client makes a request (using the HTTP protocol), for example:

`http://www.anywhere.com/myservlets/test.jsp`

2. The HTTP server compares the user specified value `myservlets` in the URL with a matching `Service` directive in the `httpd.conf` file, recognizes that this is a servlet or JSP request, and passes it forward to the WebSphere Application Server Plugin.

```
Service /myservlets/*  
/usr/lpp/WebSphere/AppServer/bin/was401.so:service_exit
```

3. If the JSP (in the example `test.jsp`) is not already loaded, WebSphere loads it from an HFS file and compiles it into a servlet.
4. The servlet is then executed by the JVM that WebSphere loads at startup.
5. The response created by the servlet is then passed back to the HTTP Server.
6. The HTTP Server passes back the response produced by the servlet to the client. If the client is a browser, the response will contain HTML formatted data.

9.3.1 Using servlets in the Plugin to talk to backend systems

While the current recommendation is to encapsulate your business logic within EJBs, you can also use servlets for some of your business logic and that logic may also have to communicate with your backend systems.

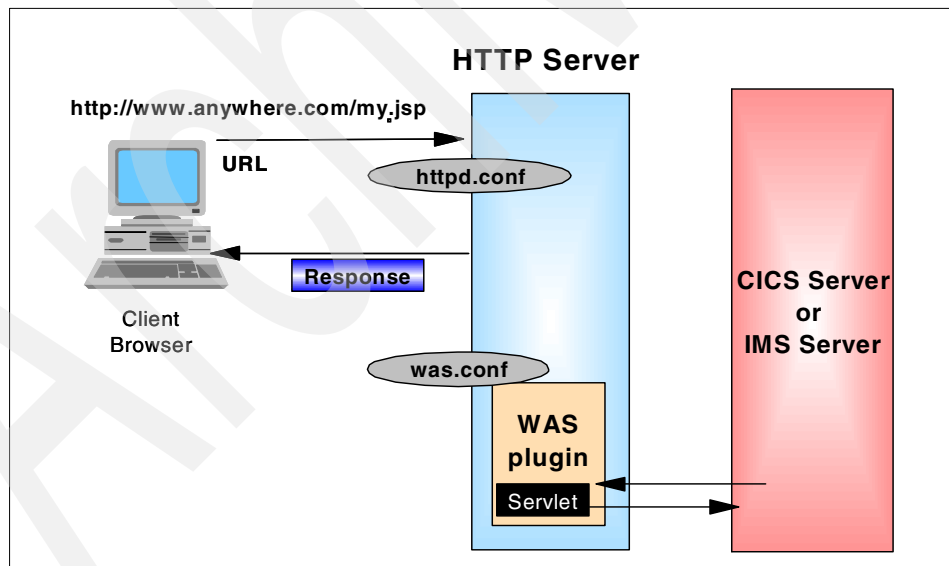


Figure 9-2 Accessing EJBs from servlets

Depending on the backend system involved, there are several means of communication such as the CICS Transaction Gateway (CTG) or the IMS Connect. In addition, JDBC drivers allow for connection to DB2 databases.

While putting business logic into servlets is not the configuration of the future, it exists today as an outdated, interim solution that was needed prior to existence of EJBs.

9.3.2 Using servlets in the Plugin to talk to EJBs

As noted previously, the servlets that run in the WebSphere Plugin can talk to EJBs running in an EJB container in a J2EE Application Server in a separate z/OS address space.

For more information about J2EE Application Server, see Chapter 10, “J2EE Application Server” on page 127.

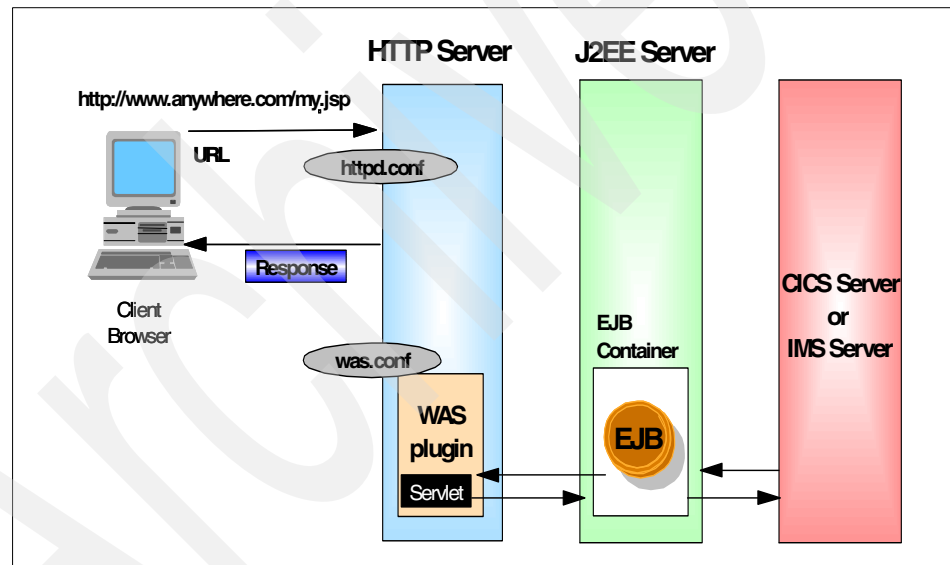


Figure 9-3 Accessing EJBs from a WebSphere Plugin

Aside from just formatting the output, servlets might need to talk to EJBs to get data from databases or invoke transactions. Servlets running locally in a WebSphere Plugin can communicate with EJBs running remotely in a J2EE Application Server.

This is done by using the Java Naming and Directory Interface (JNDI) to locate the EJBs via the LDAP directory stored on DB2. This is where the base environment of the application server (Naming Server) comes into play. For more information concerning the base environment of the WebSphere Application Server environment see “Base Environment” on page 129.

If the servlet calls a number of different EJBs, there can be some minor overhead because of the inter-address space communication. However, unlike CGI applications, there is no major overhead in building a separate new address space, since the J2EE server will already be up and running.

9.3.3 Accessing servlets in a Web container via the Plugin

In addition to running your servlets locally within the WebSphere Plugin, you can also use the WebSphere 4.0.x Plugin to run servlets remotely in a Web container. This allows you to localize your servlets and EJBs to the same z/OS address space.

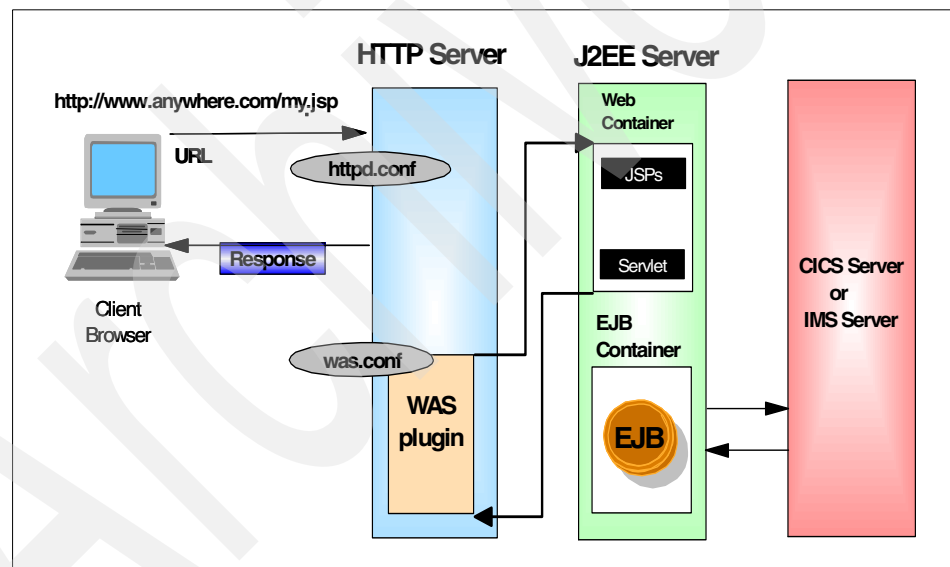


Figure 9-4 Accessing servlets in a Web Container via the WebSphere Plugin

For more information concerning this type of configuration, see 10.3.1, “Using the WebSphere Plugin” on page 133.

J2EE Application Server

In this section, we discuss the WebSphere Application Server for z/OS version 4.0.1 J2EE Application Server component and how it can be used to serve servlets, JSPs and EJBs.

10.1 Introduction and role in e-business

The WebSphere Application Server environment for both the J2EE Application Servers and the CORBA (MOFW) Application Servers is the same. There is a base environment that consists of a number of z/OS address spaces and provides the needed support.

In addition to the base environment, you can have any number of different Application Servers.

This is the environment where you can develop and deploy new e-business applications; it currently supports two types of application servers:

- ▶ J2EE

This server is used to serve EJBs to both servlets and also to CORBA business objects running on a CORBA (MOFW) Application Server.

- ▶ CORBA (MOFW)

This server is used to serve CORBA business objects to either other CORBA business objects or to EJBs.

For more information about the differences between J2EE and CORBA (MOFW) servers, see 3.3, “WebSphere Application Server environment” on page 50.

Serving EJBs

One of the main functions of a J2EE Application Server is to serve EJBs.

EJBs provide you with the ability to write your business objects on any Java-enabled platform and then run them unchanged on any other Java-enabled platform.

You can develop and even test your EJBs on Intel and Windows platforms and then run them, in production, on zSeries and z/OS platforms.

In addition, EJBs enable you to share and extend your business objects. The use of visual tools and the fact that EJBs can dynamically disclose their capabilities allows you to easily extend their capabilities and tie EJBs together. This allows you to leverage your IT investment for the maximum amount of return.

Serving servlets, JSPs, and HTML

Aside from serving EJBs, the J2EE Application Server also provides a Web container for servlets, JSPs, and HTML files. This Web container exists in the same address space as the EJB container and allows for efficient communication between the two.

This also gives you the ability to consolidate the various pieces of your e-business and allows you to manage them more efficiently.

10.2 How it works

Unlike the HTTP server with a WebSphere Plugin, the WebSphere 4.0.1 J2EE Application Server environment consists of a number of z/OS address spaces. There is a base environment which provides the services needed to support the J2EE application server, and then there is the J2EE Application Server itself.

You may choose to run with multiple copies of the same J2EE application server and allow the z/OS Workload Manager (WLM) to start up new instances of that application server as necessary or you can run with just a single instance of a given application server.

In addition, you can also run multiple different J2EE application servers in the same WebSphere Application Server environment.

Base Environment

The Base Environment consists of several address spaces and, once started, is managed by the z/OS Workload Manager (WLM).

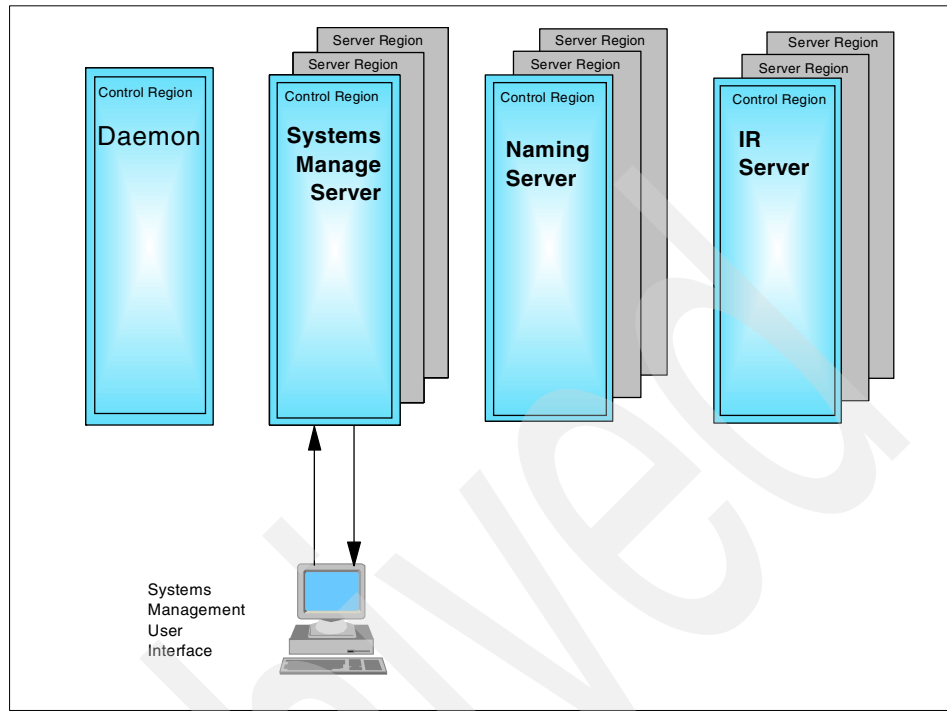


Figure 10-1 WebSphere base environment

Daemon Server

The Daemon Server is started via a procedure (usually located in SYSx.PROCLIB); it brings up the WebSphere Application Server environment. Once the Daemon is started, it will automatically start up the Systems Management, Naming and Interface Repository Control Region servers.

The Daemon Server consists of a single instance (Control Region) within a single z/OS address space.

Systems Management Server

The purpose of the Systems Management Server (SMS) is to allow users to add new applications and control the configuration of the application server environment. This is accomplished by connecting a Systems Management User Interface (SMUI) console to the SMS server via a TCP/IP connection.

The SMS plays an important role during the initial deployment of an application. Once the application is deployed, that role is diminished.

The SMS consists of a Control Region and multiple Server Regions, each in a separate address space, which can be started by WLM if needed.

Naming Server

The Naming Server (NAMING) is used to locate Java objects, such as EJBs, and works in conjunction with the Interface Repository Server. This is the heart of the operational support for both the J2EE and CORBA (MOFW) servers. It allows applications, EJBs, and CORBA Business Objects to locate and communicate with each other.

Like the SMS, the Naming Server consists of a control region and multiple server regions, which can be started by WLM if needed.

Interface Repository Server

The Interface Repository server registers and dynamically discovers business object interfaces. It manages the inventory of CORBA business object interfaces for predicate evaluation queries. The Interface Repository Server also consists of a control region and multiple server regions.

Systems Management User Interface (SMUI)

During the installation of the WebSphere Application Server Environment, one of the tasks is to install the SMUI application on a PC.

This graphical user interface allows you to view the current configuration of your application servers.

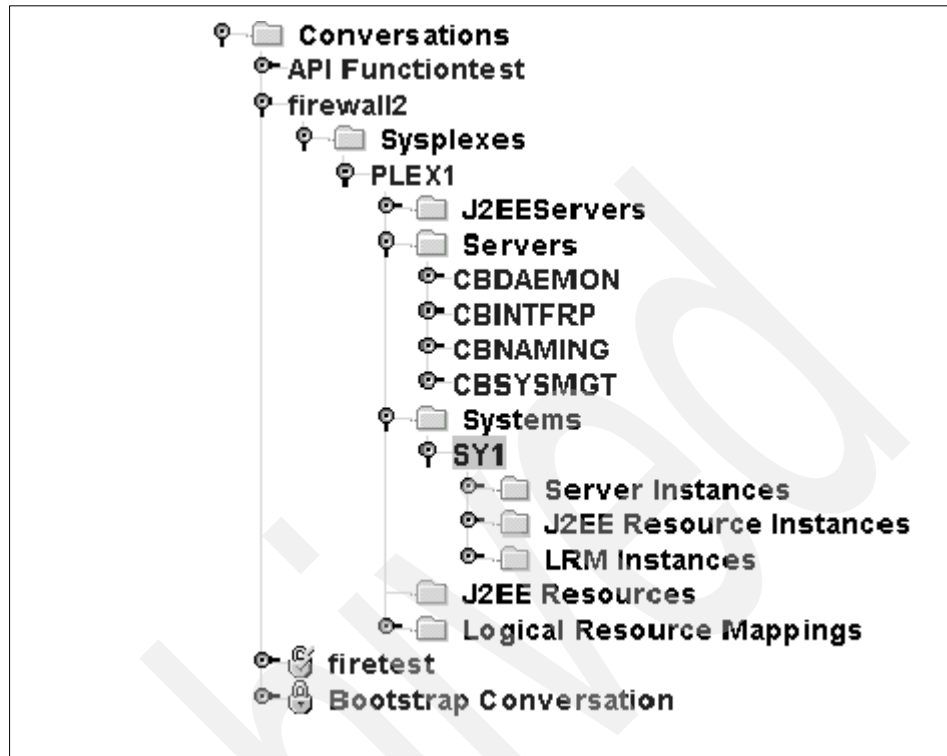


Figure 10-2 Systems Management User Interface screen

The SMUI interface allows you to see, at a glance, the status of different configurations, also called *conversations*. This is where you define and deploy new J2EE applications and is your window into the WebSphere Application Server environment.

10.3 Passing Control to a J2EE Application Server

There are a number of ways to pass control to a J2EE Application Server. One is directly through RMI over IIOP. This is usually accomplished by what is known as a Java Application Client. This redbook does not discuss this scenario.

Instead we discuss using the HTTP server in conjunction with the WebSphere 4.0.1 Plugin (see Figure 10-3 on page 133) or using the new HTTP Transport Handler (see Figure 10-4 on page 134) built into the J2EE Application Server.

10.3.1 Using the WebSphere Plugin

Using the WebSphere 4.0.1 Plugin, you can access servlets running in a Web container on the J2EE Application Server, which can then access EJBs in an EJB container running in the same J2EE Application Server.

Support for the WebSphere Plugin environment allows you to migrate from a WebSphere 3.5 Plugin configuration to WebSphere 4.0.1 Plugin configuration using the Web container in the J2EE Application Server.

The ability to run both servlets and EJBs within the same address space allows you to maintain the different parts of your e-business in a central location.

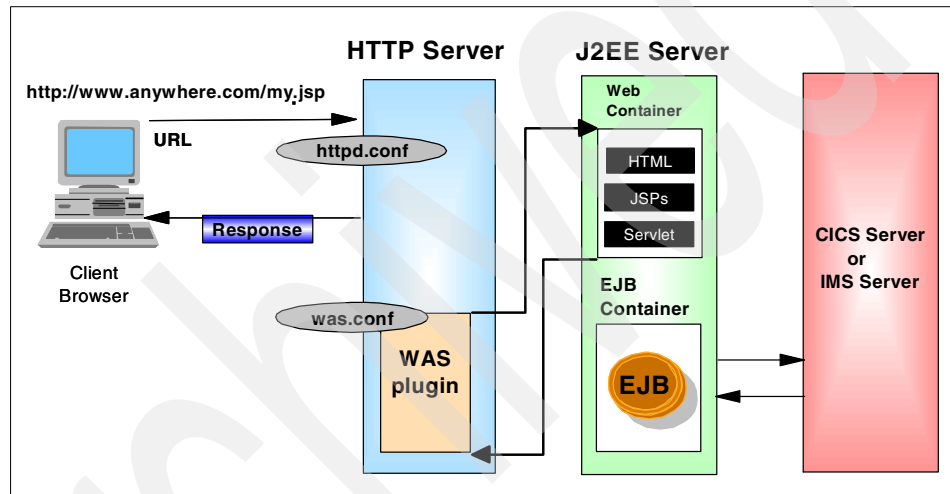


Figure 10-3 Accessing the J2EE Server via the WebSphere Plugin

1. The HTTP Server receives the request and passes it to the WebSphere 4.0.1 Plugin.
2. The WebSphere Plugin recognizes that it doesn't have a definition statement (deployed Web application) for the JSP.
3. The request is then passed to the appropriate Web container.
4. When the actual servlet runs, it can use the Naming Server to locate the appropriate EJB.
5. The EJB can use the Java 2 Connector Architecture to connect to the backend system.

10.3.2 Using the HTTP Transport Handler

The HTTP Transport Handler provides a performance enhancement if you run your Web application in the Web container. It manages incoming HTTP requests and passes control requests for servlets and JSPs running in the Web container faster than a separate HTTP server on z/OS. The Transport Handler assumes that some client browser interface exists in front of the Transport Handler to handle the nuances between different browsers on the client side.

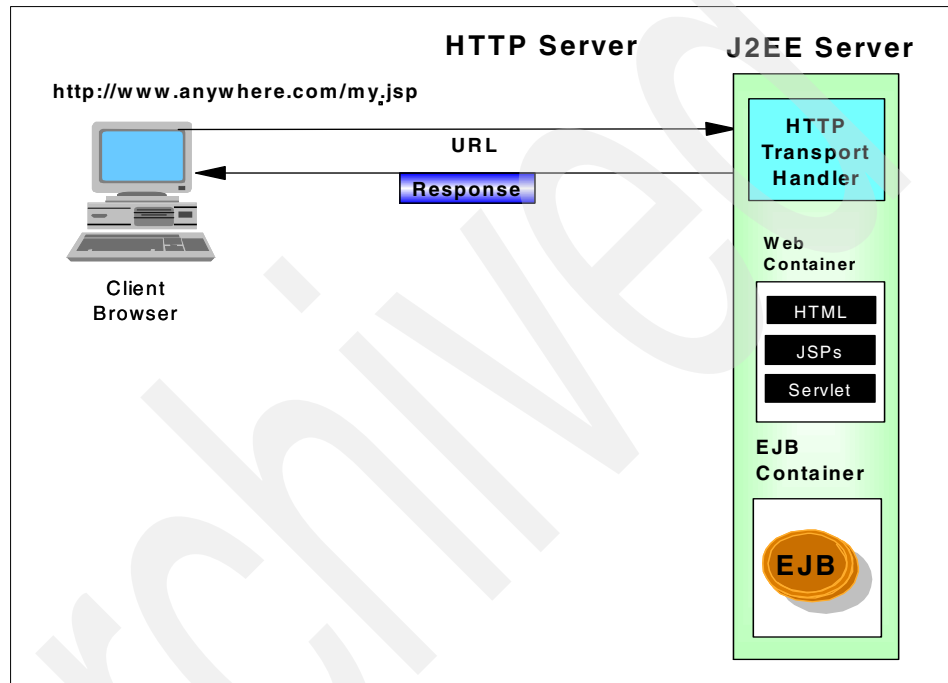


Figure 10-4 Accessing the J2EE Server via the HTTP Transport Handler

10.4 e-business related functionalities

The Java2 specification, Enterprise Edition (J2EE), is a specification developed by several companies (Sun, IBM, Inprise among others). The Java 2 Platform, Enterprise Edition (J2EE) defines the standard for developing multitier enterprise applications. J2EE simplifies enterprise applications by basing them on standardized, modular components, by providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming.

The Java 2 Platform, Enterprise Edition, takes advantage of many features of the Java 2 Platform, Standard Edition, such as “Write Once, Run Anywhere” portability, JDBC API for database access, CORBA technology for interaction with existing enterprise resources, and a security model that protects data even in Internet applications. Building on this base, Java 2 Enterprise Edition adds full support for Enterprise JavaBeans components, Java servlets, JavaServer pages, and XML technology.

Visit the J2EE technology Web site at:

<http://java.sun.com/j2ee/>

WebSphere MQ

As part of your e-business requirements, you may have a need for the asynchronous communication between application components. This type of connection is established by WebSphere MQ. This software was lately renamed from MQSeries to WebSphere MQ. We shall use the new name.

11.1 Introduction and role in e-business

According to normal component-to-component communication, calls happen synchronously. This means that a component calls another component using the RMI-IIOP procedure, and during the call the client or caller waits till the server or the called party finishes. This may be good in some situations. On the other hand, it may cause performance and reliability problems. The server must run in order to satisfy the client request.

The messaging communication is an asynchronous way components call each other. The two parties need not be up and running during the communication, hence it is called asynchronous. Over the past several years many software companies developed message-oriented middleware (MOM); IBM's WebSphere MQ is one of them. MQ sits between two, or many, parties communicating with each other and receives and delivers the messages by handling message queues.

Applications that do not run at the same time can work independently of each other. WebSphere MQ is best suited for applications that do not require immediate responses, but do need guaranteed delivery of messages.

Java supports the Java Messaging Service (JMS). The idea behind JMS is to hide the different APIs used by different MOMs. Programmers can use JMS in their Java code; the MOM actually performing the messaging is a layer below, which can be hidden.

The EJB 2.0 specification incorporates Message Driven Beans, but WebSphere for z/OS V4.0.1 does not support that specification.

11.2 How it works

WebSphere MQ connects applications across different platforms through APIs supported by a high-level programming interface. By leaving the complexity of different operating systems and network interfaces up to WebSphere MQ, application development time can be focused on business logic.

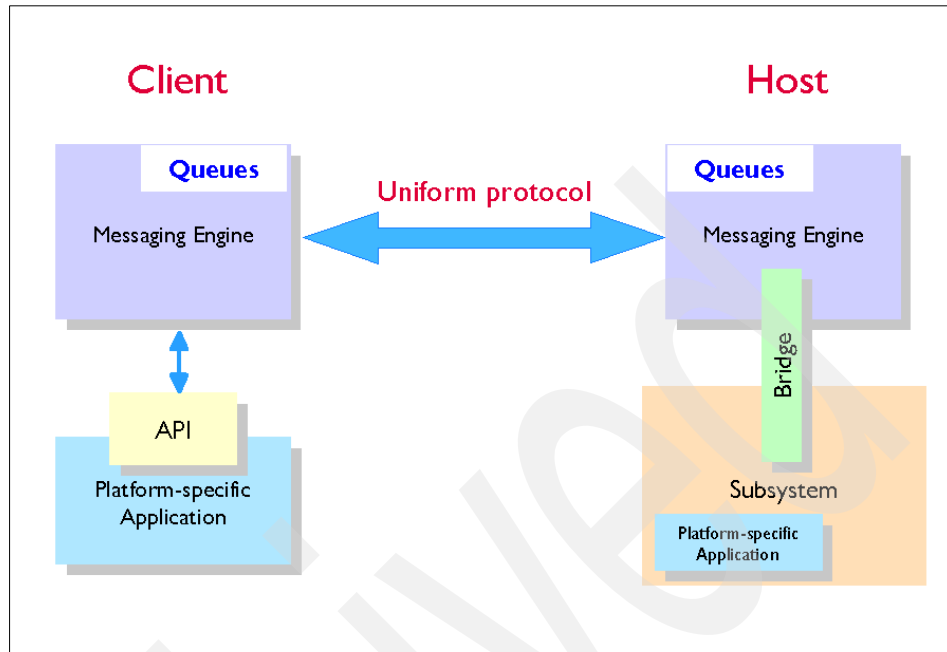


Figure 11-1 WebSphere MQ

As shown in Figure 11-1, application programs communicate with each other using messages and queues. A message-sending program can continue with other tasks or requests without the need to wait for a confirmation or response after a message is sent. The message-receiving program can spawn activities based on a message or send messages to other programs through the same queuing mechanism.

The MQSeries Queue Manager (MQM) provides messaging services for an application, ensures that messages are put in the correct queue, routes messages to other queue managers and processes messages through a common programming interface called the Message Queue Interface (MQI). The Queue Manager can retain messages for future processing in the event of application or system outages. Messages are retained in the queue until a successful completion response is received through the MQI.



DB2

In this section, we discuss the role of DB2 on OS/390 in an e-business environment. DB2 is involved in most applications on OS/390 in various ways and will also be affected in most e-business applications.

12.1 Introduction and role in e-business

The installation of DB2 is mandatory for the WebSphere Application Server V4. DB2 is used by the WebSphere Application Server to provide persistent storage for the LDAP tables used to maintain information about the various objects in both EJB and CORBA containers.

The role of DB2 is mainly that of a database manager, but the possibility of using stored procedures make DB2 an application server as well, although its transactional capabilities are kept within the scope of DB2.

DB2 on OS/390 can also play the following roles in e-business applications:

- ▶ As a data store for existing traditional applications, which in turn can be invoked from new e-business components
- ▶ As a true application server to run stored procedures, again invoked from new e-business components
- ▶ As a data warehouse, accessible from the Web
- ▶ As a database to store Web content accessible from the HTTP server, especially multimedia content
- ▶ As a database for application server, such as WebSphere Commerce Suite, ERP or CRM
- ▶ As a system database, for things like Lightweight Directory Access Protocol (LDAP) for OS/390

12.2 Accessing DB2

There are various ways in which you can access a DB2 server on Z/OS. We mention the most important ones:

- ▶ Java Database Connectivity (JDBC)

JDBC is a Java API to access relational databases in a uniform and platform-transparent way. In some cases, the JDBC API works together with a native driver provided by the database vendor. There are different types of JDBC drivers on the market. The one that is currently provided by IBM for accessing DB2 on OS/390 is a so-called Type-2 driver, which is a combination of some Java classes and some native code to access DB2 locally from any Java program. JDBC is intended to be used for dynamic SQL queries.

- ▶ SQLJ

If you intend to use static SQL queries, you must use SQLJ instead of JDBC. In most cases you will experience better performance, because the access path to DB2 is already predefined.

SQLJ requires more preparation than JDBC and also requires a slightly different way of coding, but the performance benefits make this worthwhile.

12.3 e-business-related functionalities

The most direct functions related to e-business in DB2 are:

- ▶ The availability of Java Database Connectivity (JDBC) drivers to access DB2 tables or stored procedures from Java programs

The default JDBC drivers provided by DB2 for OS/390 only support access from local Java programs (running on the same system). Remote access from Java programs is possible using vendor products or DB2 Connect.

- ▶ The possibility of writing stored procedures in Java, thus taking advantage of all the power of the Java language

This means, for example, that you could very easily move your SQL queries from a Java program to a stored procedure or vice versa, depending on what is most convenient. Stored procedures written in the Java language are available as of DB2 Version 6.1.

Java Database Connectivity (JDBC)

JDBC support for DB2 provides access to relational data from Java applications using a standard API. The JDBC API makes it possible to:

- ▶ Establish and close a connection with a database
- ▶ Send SQL statements
- ▶ Process the results of the SQL statements

There are four types of drivers defined for accessing databases through JDBC. These can be divided into two categories:

- ▶ Drivers connecting a local Java client application to a database server
- ▶ Drivers connecting a remote Java client to a database server across the network

As shown in Figure 12-1, drivers that fall into the first category are:

- **Type 1:** A JDBC-ODBC bridge plus ODBC driver. This combination provides JDBC access using existing ODBC drivers. On OS/390, this type of database access is provided by the Call Level Interface (CLI).
- **Type 2:** A native JDBC driver that converts JDBC requests from the client application into native calls on the database system.

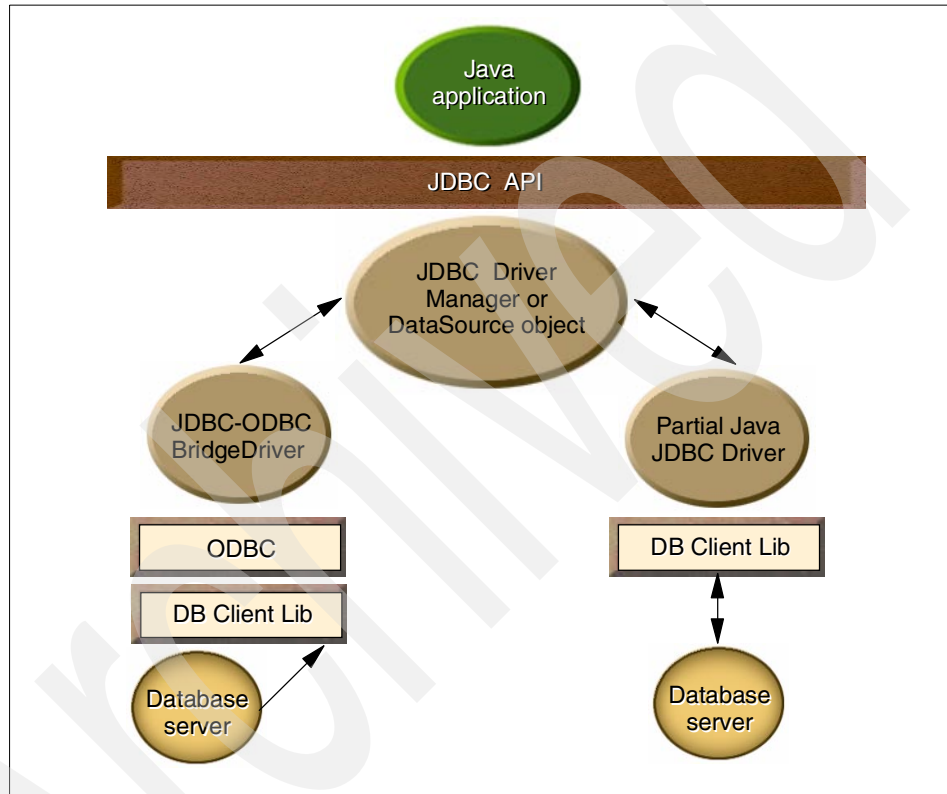


Figure 12-1 JDBC Type 1 and Type 2 drivers

As shown in Figure 12-2, drivers that fall into the second category are:

- **Type 3:** A pure Java driver for database middleware. This style of driver translates JDBC calls into the middleware vendor's protocol, which then translates to a DBMS protocol by way of a middleware server. The middleware provides access to many different databases.
- **Type 4:** A pure Java driver that connects directly to the database server using native TCP/IP or, in the case of DB2, DRDA. This style of driver converts JDBC calls into the network protocol used directly by DBMSs, allowing a direct call from the client machine to the DBMS server and providing a practical solution for Web access.

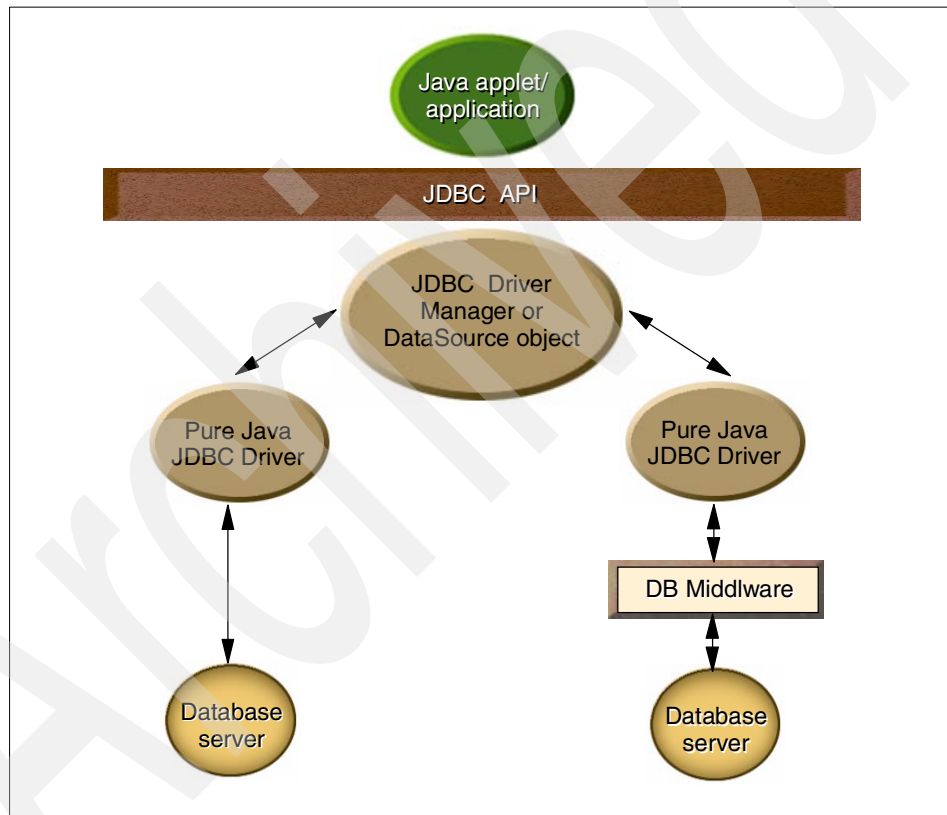


Figure 12-2 Type 3 and Type 4 JDBC drivers

Various types of JDBC drivers are usually provided by the database vendor, but there also other companies that provide JDBC connectivity solutions. DB2 on OS/390 currently supports the Type 2 driver, and there are no IBM JDBC Type 3 or Type 4 drivers available for DB2 on OS/390. However, there are some non-IBM products available.

On OS/390, IBM has replaced the Type 1 driver with a Type 2 driver in order to deliver the performance typically required of production applications and transactions. Database calls served using JDBC are not as efficient as static calls supported by SQLJ.

SQLJ

SQLJ was developed to complement the dynamic JDBC SQL model with a static model. The static model provides strong type checking at application translation time, better manageability of data access through separation of package owner from package runner, and because all SQL is compiled, better performance.

The SQLJ specification consists of three parts:

- ▶ **Part 0:** Specifies the SQLJ language syntax and semantics for embedded SQL statements (called SQLJ clauses) within a Java application.
- ▶ **Part 1:** Provides extensions defining installation of Java classes in an SQL database and also invoking static methods as stored procedures and user-defined functions.
- ▶ **Part 2:** Provides extensions to accessing Java classes as SQL data types.

With SQLJ, Java applications containing SQLJ clauses are translated by a Java translator to produce modified Java code and a platform independent description of the SQLJ clauses called a serialized profile. On the runtime platform, the profile is then customized to produce a platform-dependent Database Request Module (DBRM), which is then bound into a package or plan. The Java program is executed through the JVM on OS/390.

SQLJ has all the benefits of its JDBC counterpart and, in addition, enjoys the performance advantages of the static model. However, SQLJ requires more preparation than JDBC.

CICS

In this chapter we discuss the CICS-specific connectors that are available.

We discuss three major Web-enablement technologies:

- ▶ CICS Web support
- ▶ CICS Transaction Gateway
- ▶ CICS EJB support

13.1 Introduction and role in e-business

CICS transactions play a significant role in the e-business world. Millions of business transactions are executed each day that have been written in the past 30 years, and there is no reason to rewrite these transactions. On the other hand, there is a need to enhance the transactions by providing:

- ▶ A Web-based GUI interface
- ▶ Additional business logic
- ▶ Service to new client types, not just the 3270
- ▶ Separation of the presentation logic and the business logic for easy maintenance

In the past couple of years CICS provided a number of ways to satisfy the above requirements.

13.2 CICS Web enablement

This section introduces the principal CICS Web-enabling technologies, together with references for further information.

The four strategic CICS Web-enabling technologies are:

- ▶ CICS Web Support (CWS)
- ▶ CICS Transaction Gateway (CTG)
- ▶ CICS Enterprise JavaBeans (EJB) support
- ▶ WebSphere Host Integration (incorporating WebSphere Host On-Demand and Host Publisher; we do not deal with this solution)

In addition to the strategic solutions, there are a number of other options which may be appropriate for some business scenarios. We do not deal with these solutions in this book.

- ▶ MQ Series
- ▶ CICS CORBA support
- ▶ CTG terminal servlet
- ▶ CICS to TCP/IP sockets interface
- ▶ Web templates for OS/390
- ▶ CGI and GWAPI

This section starts with a brief overview of the principles of CICS modular application design as it relates to CICS Web-enabling. We then present an introduction to some of the CICS Web-enabling technologies.

13.2.1 The separation of presentation and business logic

A sound principle of CICS application design is to separate the presentation logic from the business logic. Communication between the programs is by using the EXEC CICS LINK command, and data is passed between such programs in a buffer known as a communication area (COMMAREA). The structure of this data in the COMMAREA is also part of the application design. This is illustrated in Figure 13-1.

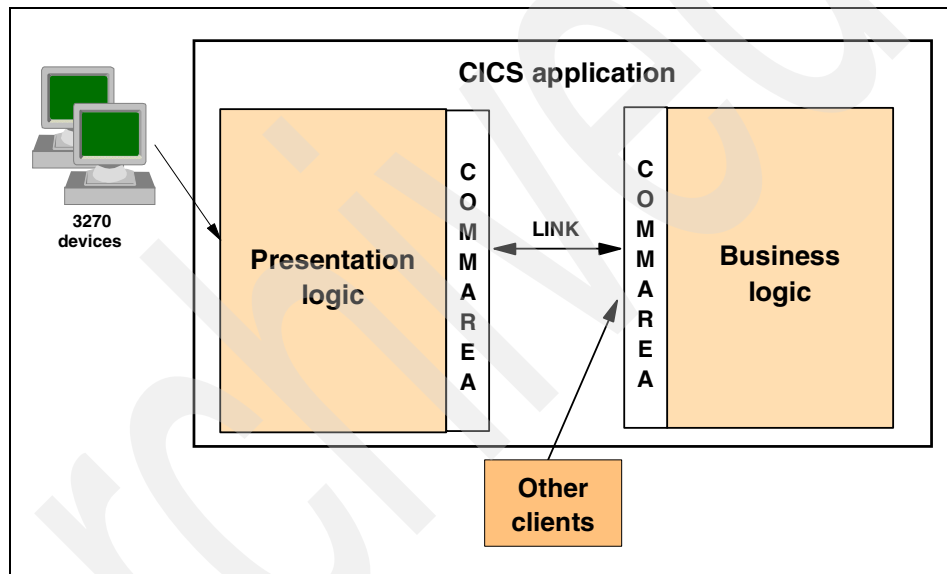


Figure 13-1 Separation of CICS business and presentation logic

Such a modular design provides not only a separation of functions, but is also the key for re-use of existing applications using different presentation methods (such as the Web access methods that are the subject of this book). These presentation and business logic modules are still executed together as a single CICS task, but if designed in this form, they can also readily exploit the distributed program link (DPL) and workload management functions provided by CICS to spread work within a sysplex or between different CICS systems distributed across a network. For further details on a modern CICS modular application design, refer to the O'Reilly publication, *Designing and Programming CICS Applications*, ISBN 1-56592-676-5.

The various methods by which the business logic in a CICS application can be invoked using a COMMAREA interface are as follows:

- ▶ From a CICS application, where the presentation logic is HTTP-based (Web-aware)
- ▶ From a workstation using the CICS Universal Client and External Call Interface (ECI)
- ▶ From a Java applet, servlet or application, using the facilities of the CICS Transaction Gateway and the Java version of the ECI
- ▶ From a Session Bean in CICS TS, using the JCICS classes or the CICS Connector for CICS TS V2.1
- ▶ From another program running in the z/OS sysplex (such as a Web server GWAPI program), using the EXCI (External CICS Interface)
- ▶ From any program which uses an EXEC CICS LINK and a COMMAREA structure to pass data

Many legacy applications were developed without separation of presentation and business logic. Most application programmers did not plan for the Year 2000 data change, nevermind Web-enabling their applications. As a result, these applications are often deemed too difficult to re-engineer. For that reason, IBM has developed Web-enabling technologies that allow re-use of the 3270 interface, as well as technologies that require a callable COMMAREA interface.

13.2.2 CICS Web support

CICS Web support (CWS) is a set of resources supplied with CICS TS for z/OS that provide CICS with some functionality similar to a real Web server. A summary of how a CICS application can be Web-enabled using the CWS is illustrated in Figure 13-2 on page 151.

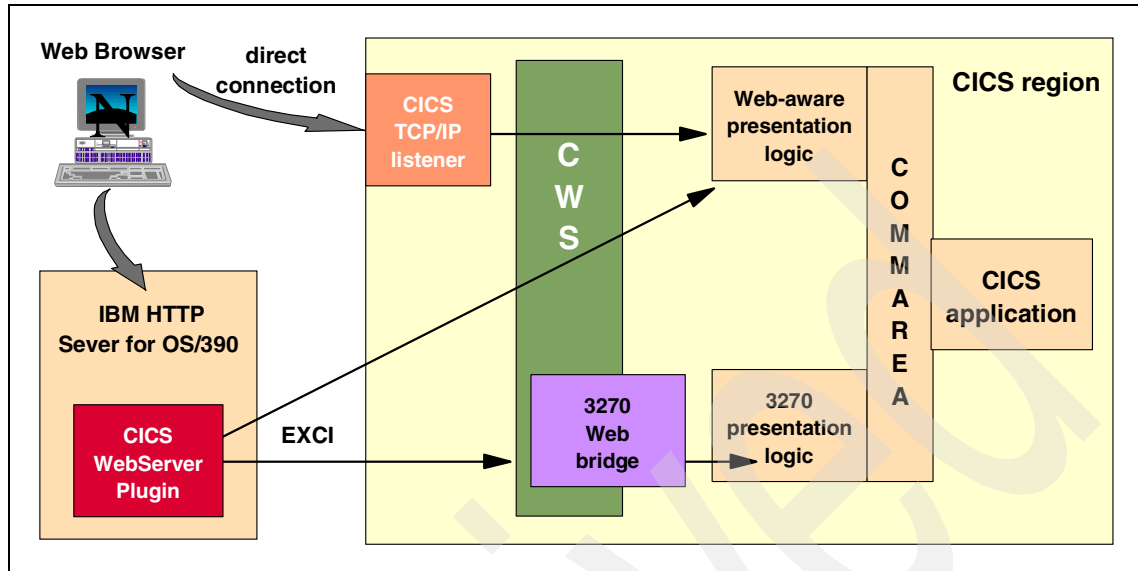


Figure 13-2 CICS Web support overview

CWS provides a native HTTP interface to CICS; this interface can be used by both 3270 based transactions and applications that provide a callable COMMAREA interface. Two different configurations can be used to route the HTTP requests into the CICS region. Both configurations allow the use of the same facilities in CICS, although the configuration of the two options is significantly different. These configurations are as follows:

- ▶ A *direct connection* from a Web browser to CICS. This uses the facilities of the CICS TCP/IP listener to pass the request directly into CICS Web support.
- ▶ Through the HTTP Server for z/OS using the facilities of the CICS WebServer Plugin. This is a CICS-supplied GWAPI extension to the HTTP Server for z/OS. It routes requests from the HTTP Server into the CICS region using the EXCI communication mechanism. The supplied GWAPI module is called DFHWBAPI, and was previously termed the “ICAPI DLL”.

With both, the direct connection and the CICS WebServer Plugin, CWS can be used to invoke two types of CICS applications:

- ▶ To invoke a 3270 transaction, the facilities of the 3270 bridge are used. The 3270 transaction remains unchanged, and the 3270 output is converted to HTML. We will refer to this function as the *3270 Web bridge*. This function is only available when using CICS Transaction Server V1.2 or above.
- ▶ To invoke an existing application that provides a callable COMMAREA interface, some new CICS presentation logic must be written. This logic uses

CICS facilities to interpret, act upon, and then build and return the HTTP data stream. We will refer to a CICS application containing such logic as “Web-aware”. This Web-aware logic can be contained either within the CWS converter Encode and Decode routines, within the original program, or in a separate Web presentation module that links to the original program. To create this Web-aware presentation logic, CWS provides two methods:

- WEB API
- COMMAREA manipulation

The WEB API, together with the DOCUMENT API and TCPIP API, provide a rich set of functions to interpret, manipulate, and build the HTTP data streams within a CICS application. They are part of the CWS function first delivered in CICS TS V1.3, and are described in more detail in Chapter 5 of the *CICS Internet Guide*, SC34-5713, and Chapter 3 of *CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge*, SG24-5480.

CICS Web support, direct connection

Figure 13-3 illustrates the major components of CICS Web support when using Web-aware presentation logic via a direct connection to CICS. The light shading is for CWS components that run under the Web attach transaction; the darker shading is for CWS components that run under the alias transaction.

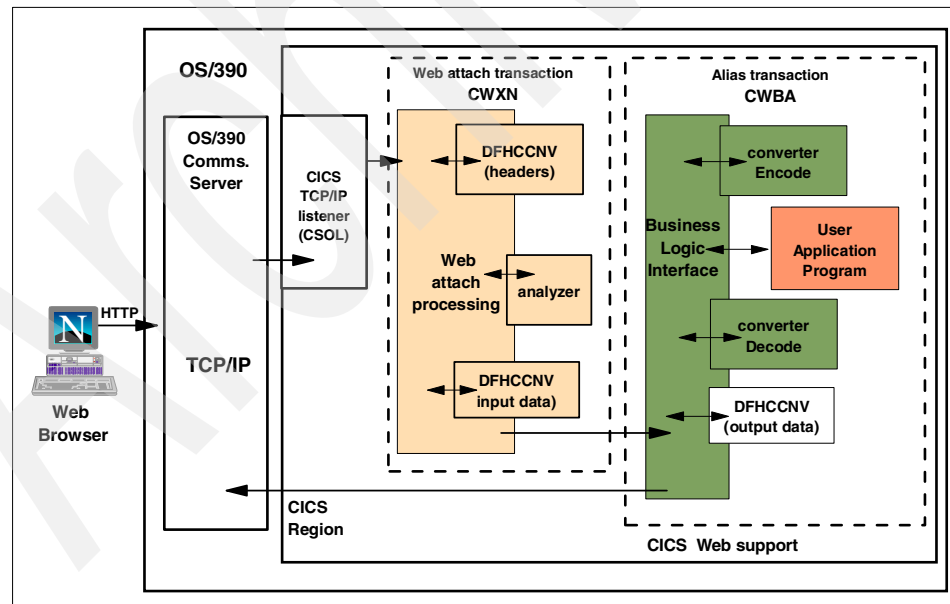


Figure 13-3 CICS Web support—direct connection

Using the CWS direct connection allows greater than 32 KB of data to be returned from the application to the Web browser, although only 32 KB of data can be received by the application.

Archived

13.3 CICS WebServer Plugin

An alternative approach to accessing CICS Web support is through the services of the IBM HTTP Server for z/OS, using the CICS WebServer Plugin, (DFHWBAPI). In this implementation, some of the function previously handled through the CICS-supplied programs for CICS Web support is now replaced by function within the Web server and its plugin.

The CICS WebServer Plugin replaces the functionality of the CWS Web attach transaction, described previously. The IBM HTTP Server for z/OS has to be configured with a service directive in order to function with the CICS WebServer Plugin. This configuration is described in the *CICS Internet Guide*, SC34-5713. Using this service directive, the HTTP Server receives the HTTP request, builds an EXCI request, and invokes the BLI using the CSMI mirror transaction in the target CICS region. The HTTP data stream is passed to the BLI in an EXCI COMMAREA.

Figure 13-4 illustrates the major components of CICS Web support when using Web-aware presentation logic via the CICS WebServer Plugin.

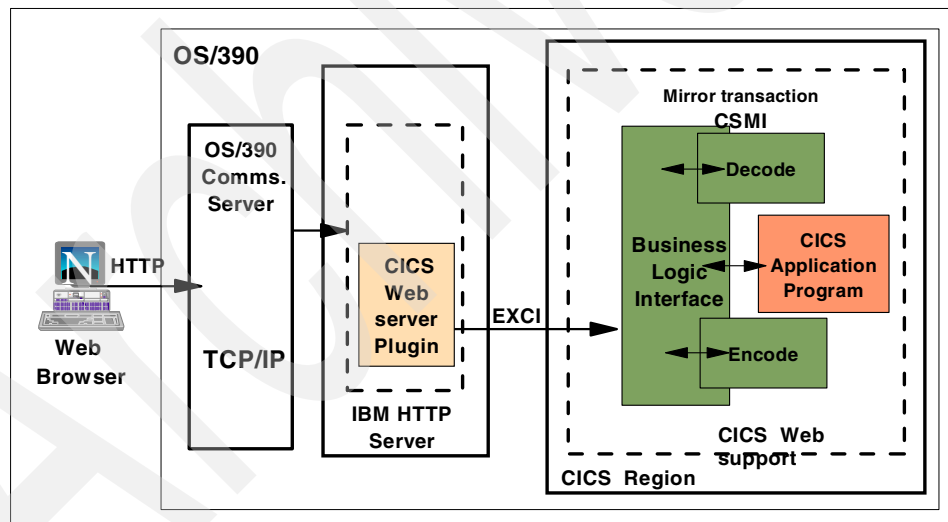


Figure 13-4 CICS Web support — CICS WebServer Plugin

The same facilities within CICS are available using the CICS WebServer Plugin as when using a direct connection, but there are a few important differences, which are summarized here:

- ▶ The IBM HTTP Server for z/OS and the CICS region must be running within the same z/OS sysplex, since the CICS WebServer Plugin uses the EXCI communication mechanism.
- ▶ Only 32 KB of data in the HTTP data stream can be passed to or from the CICS program when using the CICS WebServer Plugin. This is because the EXCI uses a standard CICS COMMAREA on which the restriction of 32 KB applies.
- ▶ Security processing is performed in the IBM HTTP Server for z/OS if using the CICS WebServer Plugin.
- ▶ Code page conversion is performed in the HTTP Server, not in CICS, when using the IBM HTTP Server for z/OS.
- ▶ Some of the WEB API commands may give a slightly different response when using the Web server Plugin.

For further information on using and configuring CWS with the CICS WebServer Plugin, refer to the following manuals:

CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge, SG24-5480

CICS Internet Guide, SC34-5713

For information on configuring the IBM HTTP Server for z/OS, refer to:

HTTP Server Planning, Installing, and Using, SC34-4826

3270 Web bridge

The 3270 bridge feature of CICS, when used in conjunction with CWS, can provide access to 3270 transactions from the Web. We refer to this function as the 3270 Web bridge. To implement this solution, you need only to configure CWS, add CICS PROGRAM and TRANSACTION definitions, and reassemble your BMS mapsets. Most 3270 transactions will then run unchanged, though some applications may require modification. The ease of implementation makes the 3270 Web bridge the preferred solution whenever Web access is required quickly, programming resources are limited, or the application has limited use or life expectancy.

The 3270 Web bridge can be used with either the direct connection to CICS or with the CICS WebServer Plugin. Figure 13-5 illustrates the data flow for a Web browser request using the facilities of the 3270 Web bridge and a CWS direct connection to access a CICS 3270 transaction. The dark shading indicates the components of the 3270 Web bridge. Note that the 3270 bridge feature is only available when using CICSTS for z/OS V1.2 or above.

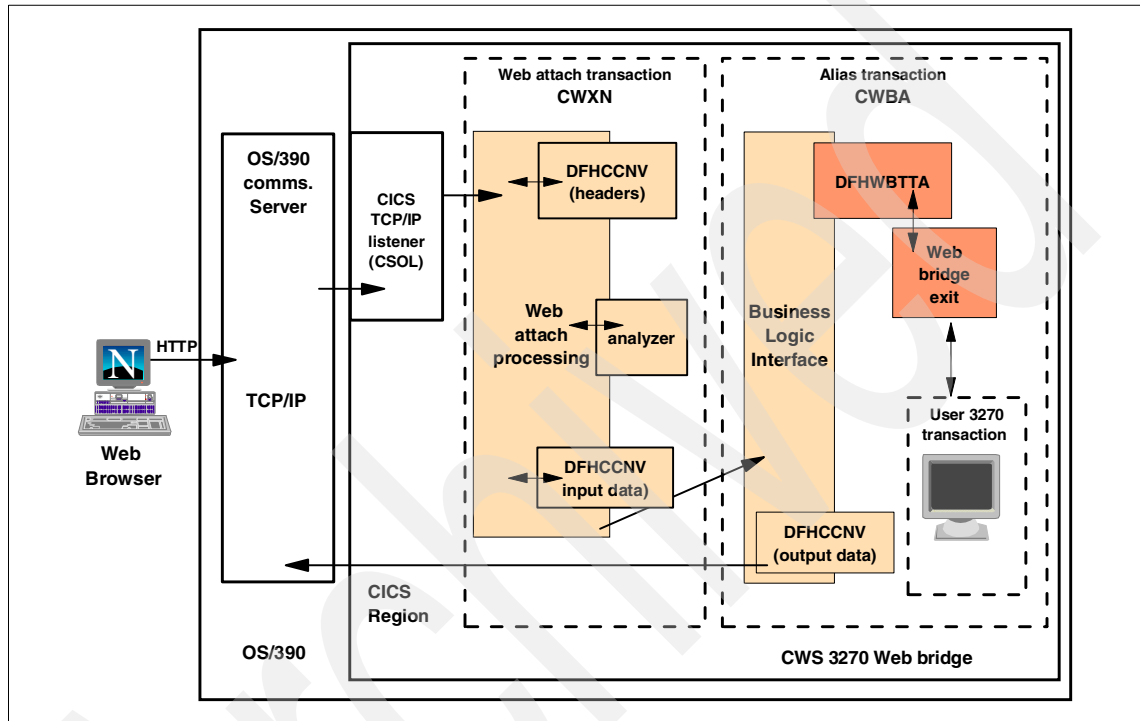


Figure 13-5 CICS Web support — 3270 Web bridge

The initial data flow is the same as that described in Figure 13-3 on page 152 for the description of CICS Web support and the BLI. However, instead of invoking the user program, the Web terminal translation program, DFHWBTTA, is invoked by the BLI. DFHWBTTA starts the transaction to be run in the 3270 bridge environment, where it runs in conjunction with the CICS-provided Web bridge exit DFHWBLT.

13.3.1 CICS Transaction Gateway

The CICS Transaction Gateway (CTG) is a set of client and server software components incorporating the facilities of the CICS Universal Client that allow a Java application to invoke services in a CICS region. The Java application can be an applet, a servlet, an EJB, or any other Java application.

CTG V4.0 or later is required for WebSphere Application Server for z/OS V4.0 or later.

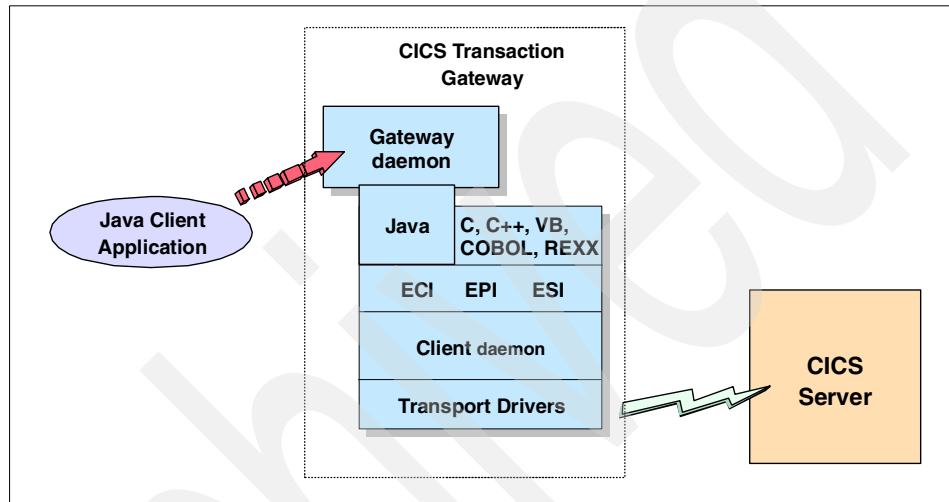


Figure 13-6 CICS Transaction Gateway

The CICS Transaction Gateway consists of the following components:

- ▶ Gateway daemon
- ▶ Client daemon (only on distributed platforms)
- ▶ Java class library
- ▶ Configuration tool (GUI configuration tool)
- ▶ Terminal Servlet (3270 emulator)

We discuss only the components that relate to our topic.

Gateway daemon

This long-running process functions as a server to network-attached Java client applications (such as applets or remote applications) by listening on a specified TCP/IP port. The CTG supports four different CTG *network protocols* (TCP, SSL, HTTP, or HTTPS), each of which requires a different CTG *protocol handler* to be configured to listen for requests.

On z/OS the External CICS Interface (EXCI) is used to provide ECI access to COMMAREA-based CICS programs. Subsequently, the EPI and ESI interfaces are not available with the z/OS CTG, and there are a few differences between the ECI support provided on distributed platforms.

The CICS Transaction Gateway daemon is not usually required when running servlets within the WebSphere Application Server on z/OS, since the CTG *local* protocol can be used to directly invoke the native functions in the underlying EXCI. The EXCI passes the request onto the attached CICS region, which is not aware that the request comes from a Java application. WebSphere for z/OS and the CICS should be on the same member of the Parallel Sysplex (because of the local protocol used), although CICS can route work to other members of the CICSplex.

Java class library

The CTG Java class library provides the following components:

- ▶ Base classes

These classes are used to instantiate a `JavaGateway` object, and then methods from the ECI, EPI, or External Security Interface (ESI) classes are used to pass requests to the connected CICS region.

- ▶ CICS CCF connector classes

The IBM Common Connector Framework (CCF) is an architecture that provides Java developers with a standardized set of interfaces to access Enterprise Information Systems (EIS). The CCF connector classes implement the CCF interfaces and programming model. The CCF is comprised of three core components:

- CCF classes

The principal components are the `CICSConnectionSpec`, the `ECIInteractionSpec`, and the `EPIInteractionSpec`. These are used to control the interaction with the CICS EIS.

- Java Record Framework

The Java Record Framework is used to build `Record` objects to wrap data structures such as a COBOL COMMAREA or a BMS map. It also provides a powerful set of marshalling options for the encoding of data, and for retrieving fields from the COMMAREA or BMS data streams.

- Enterprise Access Builder (EAB)

The EAB is a function of VisualAge for Java. Through a set of SmartGuides, the EAB allows Command beans to be developed that encapsulate the CCF classes, and the Java Record Framework Records.

J2EE Connector Architecture

The J2EE Connector Architecture has now replaced the CCF as the strategic way of connecting to EIS. If you are developing new applications that connect to EIS, we strongly recommend that you use the J2EE Connector Architecture instead of the CCF.

CTG V4.0.2 provides J2EE Connection Architecture support and WebSphere for z/OS with PTF UQ99329 provides the J2EE Connector support. These connectors, which are also known as resource adaptors, are also designed to be RRS-compliant; in other words, they are designed specifically to work with the resource recovery services (RRS) component of z/OS or OS/390. Resource recovery consists of the protocols and program interfaces that allow WebSphere for z/OS, the RRS component of z/OS or OS/390, and CICS to work together to make consistent changes to multiple protected resources. Protected resources are considered critical to a company's work, and the integrity of these resources must be protected.

WebSphere for z/OS, the RRS component of z/OS or OS/390, CICS subsystems, and these RRS-compliant connectors are all designed to participate in two-phase commit processing, which enables z/OS or OS/390 to restore critical resources to their original state if they become corrupted because of a hardware or software failure, human error, or a catastrophe.

Version 1.0 of the J2EE Connector Architecture defines a number of components that make up this architecture (see Figure 13-7 on page 160):

- ▶ Common Client Interface (CCI)

The CCI defines a common API for interacting with resource adaptors. It is independent of a specific EIS. A Java developer communicates to the resource adapter using this API.

- ▶ System contracts

These are a set of system-level contracts between an application server and EIS. They extend the application server to provide:

- Connection management
- Transaction management
- Security management

These system contracts are transparent to the application developer, that is, they do not implement these services themselves.

► Resource adapter deployment and packaging

A resource adapter provider develops a set of Java interfaces/classes as part of its implementation of a resource adapter. The Java interfaces/classes are packaged together with a deployment descriptor to create a *Resource Adapter Archive* (represented by a file with an extension of rar). This Resource Adapter Module is used to deploy the resource adapter into the application server.

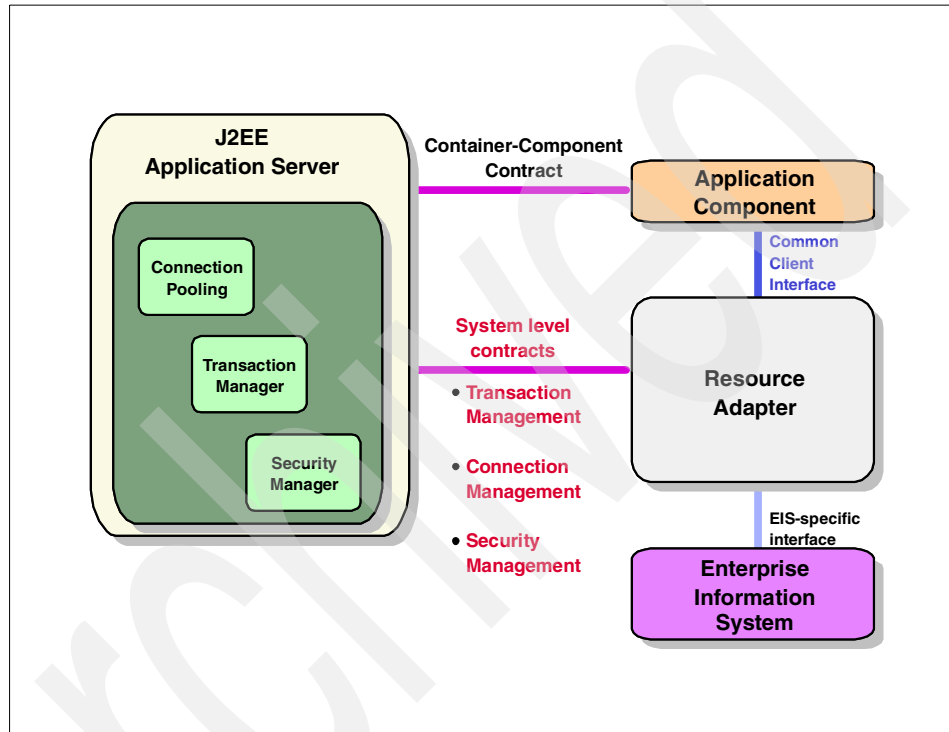


Figure 13-7 J2EE Connector Architecture components

CTG Version 4.0.2 provides support for accessing CICS from WebSphere Application Server V4 for z/OS. See *Java Connectors for CICS*, SG24-6401 for details.

13.3.2 CICS EJB support

Enterprise beans are reusable Java server components written to Sun Microsystem's Enterprise JavaBeans (EJB) specification. They can be used in an application server called an Enterprise Java Server (EJS). The EJS provides interfaces and services defined by the EJB specification.

Enterprise beans execute within a container provided by the EJS. They are located by looking up their names in a name server using the Java Naming and Directory Interface (JNDI). The EJB Container provides services such as transaction support, persistence, security, and concurrency.

CICS TS V2.1 provides partial support for Version 1.1 of the EJB specification. The EJB Container within CICS provides the services required by enterprise beans running within the CICS EJB server.

Restriction: CICS TS V2.1 does not support entity beans. Session beans are a natural extension to the existing transactional capabilities of CICS; however, entity beans have no obvious mapping to existing CICS functionality.

How CICS provides EJB support

This section describes how CICS provides the capability to run enterprise beans in the CICS environment. The components that make up the CICS EJB Server environment in CICS TS 2.1 are:

- ▶ TCP/IP listener
- ▶ Request receiver
- ▶ Request models
- ▶ Request stream
- ▶ Request processor
- ▶ EJB container
- ▶ Object store
- ▶ Java Virtual Machine
- ▶ CORBA Server
- ▶ Deployed JAR files

These components are illustrated in Figure 13-8 on page 162. If they are split between a listener region and an AOR in a logical CICS EJB Server, then the components on the left must run in the listener region and the components on the right in the AOR.

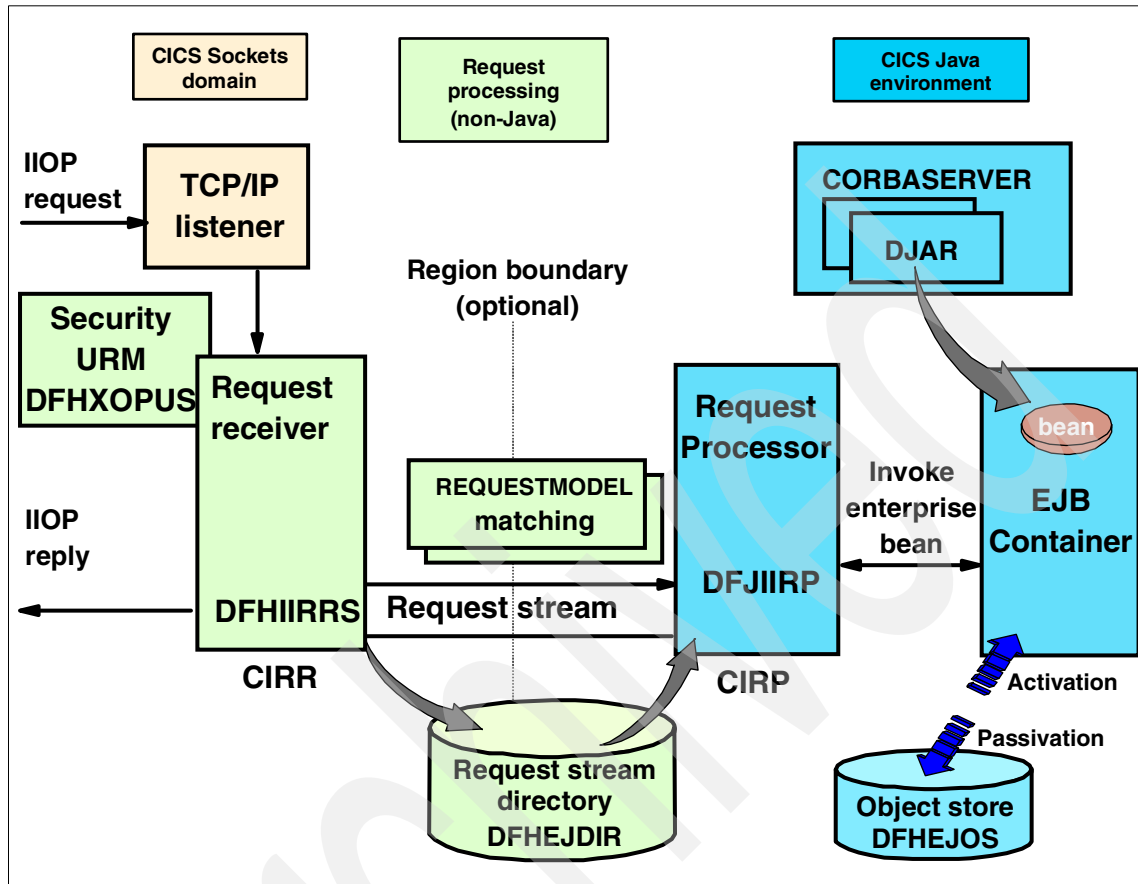


Figure 13-8 Components of the CICS EJB Server

For details about each component, see *Architecting Web Access to CICS*, SG24-5466.

Web access using CICS EJB support

In this section we discuss the different ways that the EJB support in CICS can be used to enable Web access to CICS applications. We also discuss how these applications can be enhanced by adding business components which can invoke services within WebSphere or enable database access. For details on application development using enterprise beans, refer to Chapter 16, "Application development" on page 183.

Session beans in CICS

Figure 13-8 on page 162 shows how you can first provide Web access to existing CICS applications by developing a session bean that uses the CICS Connector for CICS TS or the JCICS classes to invoke the existing application. In this scenario, the Web presentation logic for the application can be provided by developing servlets and JSPs which can be deployed within WebSphere on z/OS or on a distributed platform. For further details on this, refer to Chapter 3 in *Enterprise JavaBeans for z/OS and OS/390, CICS Transaction Server V2.1*, SG24-6284.

Database access

Having provided Web access to the application, new functionality can be added, or the existing application can be rewritten to use session beans and JDBC, SQLJ, or Data Access Beans to provide database access. For details on developing a session bean using these methods, refer to Chapter 10 in *Enterprise JavaBeans for z/OS and OS/390, CICS Transaction Server V2.1*, SG24-6284.

Entity beans

As discussed in 13.3.2, “CICS EJB support” on page 160, CICS TS 2.1 does not provide support for entity beans. However, from CICS, you can access entity beans on any other EJB server that does support them. WebSphere Application Server V4.0 does support entity beans, and therefore new components can be added to your session beans to invoke entity beans in WebSphere Application Server. One reason why you would run your session beans in CICS and entity beans in WebSphere Application Server is that CICS is optimized to run pseudo-conversational transactions, which are similar to session beans in the EJB model. WebSphere Application Server, however, is optimized for long-lived data objects, which are similar to entity beans in the EJB model.

Session beans in WebSphere Application Server

It may be appropriate for some business objects to be deployed as session beans in WebSphere Application Server. This may be due to specific resource or platform requirements. In this case, these session beans can be part of an application which encapsulates business logic across session beans running in CICS and in WebSphere Application Server. The WebSphere Application Server, in this scenario, may be running on a distributed platform which has the specific resources required by the CICS application.

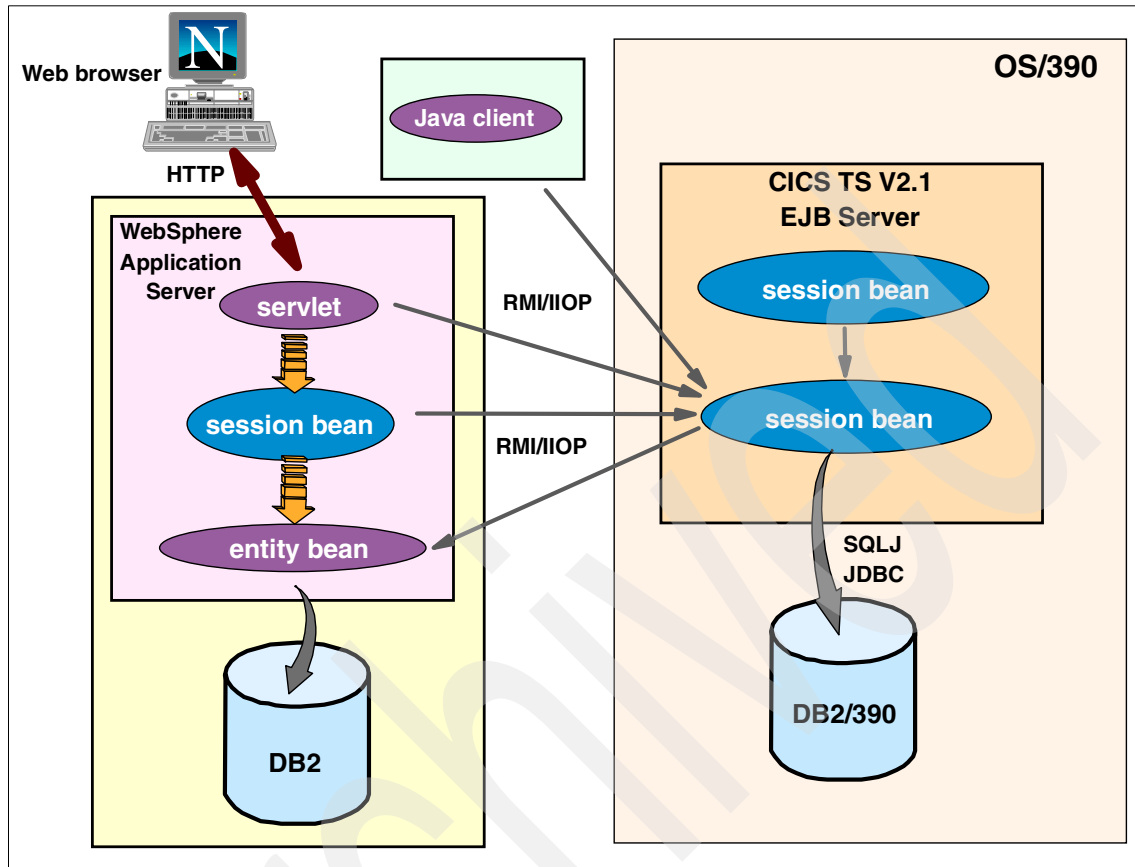


Figure 13-9 Web access using CICS EJB support

This flexibility of design enables business components to be deployed where the platform and the products provide the most efficient environment for the business logic embodied within the application. For details on developing enterprise beans for WebSphere Application Server, refer to *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144.

IMS

In this chapter we discuss the IMS-specific connectors that are available.

First, we discuss the Open Transaction Manager Access. Then we deal with IMS Connect, which implements OTMA, and finally we detail the IMS Connector for Java product.

14.1 Introduction and role in e-business

A huge portion of the world's business data resides in IMS databases. It is an integral part of our everyday life. From ATM transactions to airline reservations, IMS data drives businesses all around the world.

The following is a discussion of some of the ways you can connect to your IMS data.

14.2 Open Transaction Manager Access

Open Transaction Manager Access (OTMA) is a function of IMS that was introduced with IMS Version 5. OTMA is a transaction-based connectionless client-server protocol that provides an access path and an interface specification for sending and receiving transactions and data from IMS.

OTMA is specifically implemented for IMS in a z/OS sysplex-capable environment. Therefore, the domain of OTMA is restricted to the domain of MVS Cross System Coupling Facility (XCF). XCF is a component of z/OS that provides functions to support cooperation between authorized programs running within a sysplex. OTMA and MVS applications compose an XCF group, where OTMA and the applications are group members. In a parallel environment, different members can be on different z/OS images.

OTMA is designed to be a high-performance protocol that allows z/OS programs to access IMS applications. The support of large networks is handled through the use of OTMA clients. Due to the fact that the OTMA connections utilizes internal cross system communication that is almost comparable to main memory processing, OTMA performance is good.

OTMA provides the facility for IMS to communicate very efficiently with z/OS applications other than VTAM. These MVS applications are called OTMA clients. These OTMA clients can be IBM written application, independent software vendor's application, or user-written code.

IMS Connect is a commonly used OTMA client. The IBM-supplied IMS e-business Connectors, such as IMS Connector for Java, are based on the use of IMS Connect. These e-business Connectors are discussed in more detail in *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980 and *e-business Cookbook for z/OS, Volume II: Infrastructure*, SG24-5981. The MQSeries product also has the OTMA interface implemented, and another example of an OTMA client can be DCE/RPC.

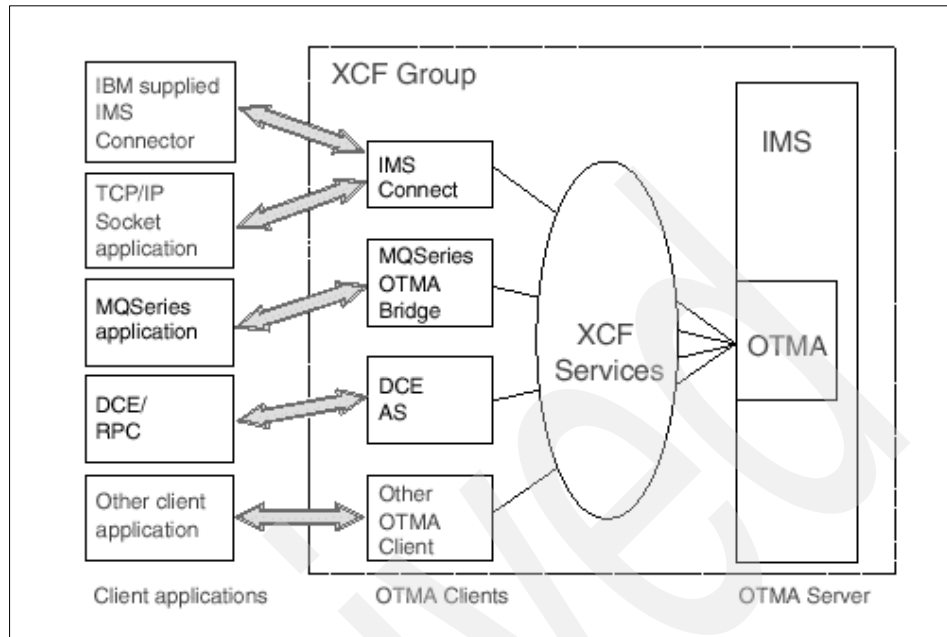


Figure 14-1 OTMA components

14.2.1 IMS Connect

IMS Connect functions as a server that provides connectivity to IMS Transaction Manager from any TCP/IP client on a TCP/IP network. It provides communication linkages between remote workstations and IMS as well as multiple TCP/IP clients accessing multiple data store resources. IMS Connect was formerly called IMS TCP/IP OTMA Connection (ITOC).

Three core components make up the IMS Connect functionality:

- ▶ The Workstation Communication Component (WCC) processes communications between IMS Web and IMS Connect.
- ▶ The Datastore Communication Component (DCC) processes communications between IMS Connect and IMS.
- ▶ The Command Component (CMD) processes commands that are received from the MVS console operator.

Figure 14-2 on page 168 shows a general view of the IMS Connect environment. The IMS Connect architecture is designed to support any TCP/IP clients communicating with socket calls. IMS Connect also supports the TCP/IP clients using the IMS Connector for Java, which is a collection of Java beans, that

enable a Java application to communicate data requests, via TCP/IP and the IMS Connect, to IMS. IMS Connect supports the Local Option connection, which is a non-TCP/IP connection between IMS Connect and IMS Connector for Java with WebSphere Application Server.

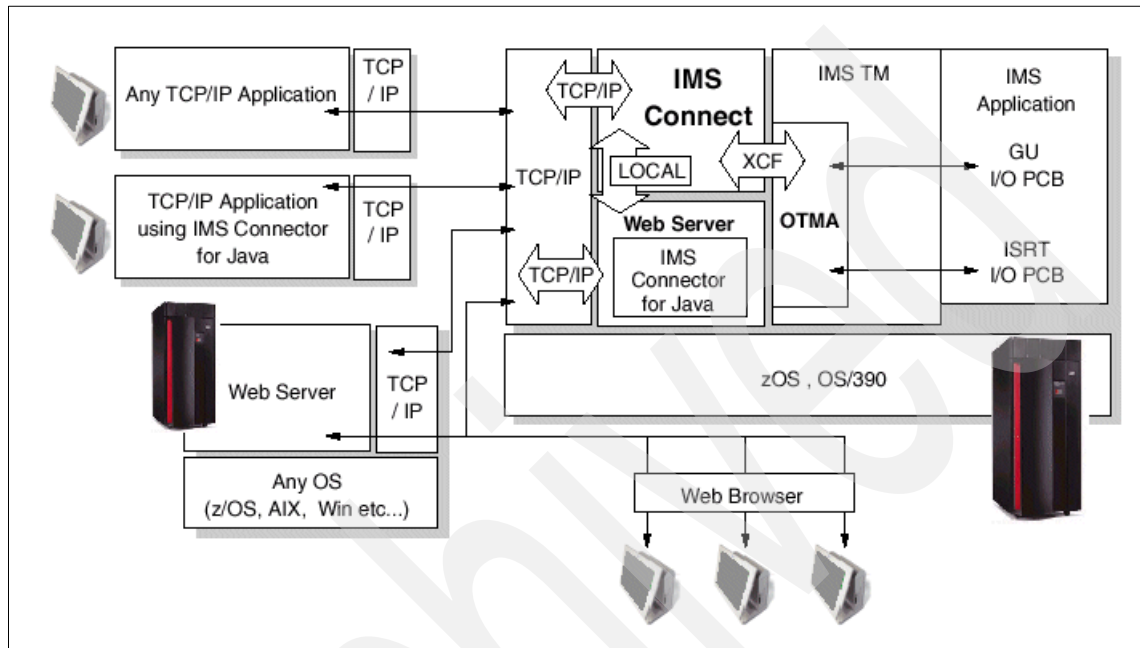


Figure 14-2 IMS Connect components

14.2.2 IMS Connector for Java

IMS Connector for Java is a collection of Java objects that can enable a Java application to access IMS transaction data. IMS Connector for Java was formerly known as IMS TCP/IP OTMA Connection for Java (JITOC).

IMS Connector for Java should not be confused with IMS Connect. The differences are:

- **IMS Connector for Java**

A collection of Java beans, or objects, that enable a Java application to communicate data requests through TCP/IP and the IMS Connect to IMS. The IMS Connector for Java is the Java communications side of the interface.

► **IMS Connect**

A z/OS address space that provides a connection between TCP/IP and an IMS subsystem. The IMS Connector for Java, using its predefined connection beans, communicates with IMS Connect through TCP/IP.

IMS Connect has been enhanced to support the Local Option, which allows non-TCP/IP communications (program calls) between IMS Connector for Java with WebSphere Application Server and IMS Connect 1.1 when IMS Connect 1.1 and WebSphere reside in the same z/OS image.

Restriction: Local Option is supported only with IMS Version 7.1. APAR PQ45057 must be applied to IMS Connect in order to enable the Local Option support that allows IMS Connector for Java to use Local Option.

In multi-platform scenarios, communications between IMS Connector for Java and IMS Connect would normally be handled over TCP/IP. In z/OS, communications between these two components are handled by a program call, eliminating the need for TCP/IP along with the associated overhead (Local Option).

IMS Connector for Java can be used to develop Java applications that communicate with IMS using the following three environments:

► **IBM VisualAge for Java**

With the IBM VisualAge for Java development environment you can develop Enterprise Access Builder (EAB) commands (Java applications) that use the IMS Connect to access IMS transactions. Once developed, a Java application can be run in a Java Virtual Machine (JVM).

► **WebSphere Studio**

With the WebSphere Studio environment you can develop Java servlets from the EAB commands that you develop in VisualAge for Java. When converted to a servlet, the Java application can then run in the Web server using a Web browser.

► **WebSphere Application Server**

The WebSphere Application Server and the IBM HTTP Server enable you to run Java servlets, developed using WebSphere Studio, as true server applications from a Web browser.

IMS Connect is a prerequisite to using IMS Connector for Java. It is a program product that has to be installed on the host IMS machine, through which a Java application or servlet accesses IMS OTMA. A Java application or servlet acts as a TCP/IP client to IMS Connect. IMS Connect accepts messages from its TCP/IP clients and routes them to IMS OTMA using Cross-System Coupling Facility (XCF).

The IMS Connector for Java provides a Common Connector Framework (CCF) compliant Java interface to the IMS Connect.

With the availability of PTF UQ99329, WAS V4.0.1 for z/OS delivers an IBM implementation within its runtime referred to as “Connection Management” that supports access to IMS connectors optimized for the zSeries platform. WebSphere now supports IMS Connector for Java 1.2.2. These connectors, which are also known as resource adaptors, implement the J2EE Connector Architecture and are also designed to be RRS-compliant; in other words, they are designed specifically to work with the resource recovery services (RRS) component of z/OS or OS/390. Resource recovery consists of the protocols and program interfaces that allow WebSphere for z/OS, the RRS component of z/OS or OS/390, and IMS to work together to make consistent changes to multiple protected resources. Protected resources are critical to a company's work; their integrity must be protected.



Part 3

Application development

Faint background watermark text: ARCHITECTURE



Application design

Much of the rest of this redbook is concerned with putting an infrastructure in place to deliver e-business. However, e-business is actually delivered by applications. In this chapter, we discuss application design considerations, how to perform application development, and the tools that you can use.

15.1 Application design considerations

There are many different options for building an e-business solution, and this is true on z/OS, where there are typically many different ways of doing things. Consequently, choosing which option to use can be confusing, especially since these different ways have different implementations that make them better choices in some cases, and not so good in others.

The important thing is to find the best solution for your particular environment. In order to do that, you need to start with what you are trying to achieve. This section aims to help you think through the key considerations.

15.2 Your objective

First you need to decide what your objective is. Possible answers may be:

- ▶ Make information that is currently available in enterprise systems such as CICS, IMS, DB2 or an ERP system, available on the Web.
- ▶ Build a Web site, with static or dynamic content.
- ▶ Implement new applications on the Web, perhaps with user personalization and data storage.
- ▶ Build an e-commerce Web site for online shopping.
- ▶ Develop a completely new way of doing business with customers and other businesses, using Web interfaces.

The focus of each of these requirements is very different. In some cases, they begin with existing systems and extend them to the Web. In other cases, they imply a brand-new start.

15.3 Application functionality

The next point to understand is the application functionality that you require, and how you will provide that functionality. We can think of four possible situations:

- ▶ If you are simply extending an existing application to be accessible on the Web, then you already have all of the functionality that you need. In this case, you are simply adding a different presentation layer to support the HTTP protocol.
- ▶ You may be using an existing application as the basis, but adding new functions. In this case, you need to decide how and where to add those functions.

- ▶ You may be building a brand-new application from scratch, but you may recognize that the application will require access to existing application logic and data. Again, you need to decide how best to build the new application so that it both meets your new requirements and also connects well to the existing applications.
- ▶ You may be building a completely new application with no need to link to existing applications. Indeed, if you are an Internet start-up company, you may not even have any existing applications. In this case, you obviously have a clean sheet to start with.
- ▶ However, a successful e-business requires many different application types, and it is likely that you will need to implement an ERP or CRM application at some time in the future. In designing your new applications, you should plan how you will do that when the time comes.

15.4 Existing systems

In all of the preceding cases, we have shown that the link to existing (or new) enterprise systems is a key part of the solution. If you only need to connect to one particular system (now and in the future), you can choose a specific connector for that system. If you will need to connect to many systems, you may want to build a generic connection infrastructure. For example, if you base all your connections on MQSeries, you get access to many systems on many platforms, and you can do all of the routing and data conversion in one place in the messaging infrastructure. (This is one of the aims of CORBA also, but it assumes that all of the systems are CORBA-enabled, which is not true today.)

15.5 Software levels

While thinking about existing systems, now is also a good time to consider and document the existing software levels of your subsystems and also of your z/OS operating system. Consider, when you next plan to upgrade these systems, what release levels you will go to, and how you could introduce an additional logical partition (LPAR) at a higher software level for these new applications.

This is important because the new technologies of the Web are moving forward at a very rapid pace. To get these new technologies to work, and also to get good performance and availability, you will want to be at a recent level of the operating environment and subsystem software. This contrasts with most users' production

environments, where they prefer to upgrade infrequently (perhaps every one to two years) to provide a more stable service by minimizing the amount of change activity. Since many of the connectors require recent software levels, it is important that you understand what you have now.

15.6 Application layers

Consider how you want to build your application in terms of layers. People talk about a 2-tier or 3-tier model, but in practice you could build your application architecture out of many more layers than that. Figure 15-1 on page 177 shows an example where there are five layers:

- ▶ Presentation logic in a client device, for example in a PC
- ▶ Presentation logic in a server, for a Java servlet or JavaServer Page running in a Web server
- ▶ Business logic in new technologies such as Java and EJBs
- ▶ Business logic in existing systems such as CICS, IMS and SAP R3
- ▶ Data in existing databases such as DB2

This split of functionality provides ease of development and maintenance. Different parts can be modified by different groups of people without affecting the rest of the system. You can have presentation experts building the Web pages, data administrators managing the data, and business logic specialists writing the application code in the middle.

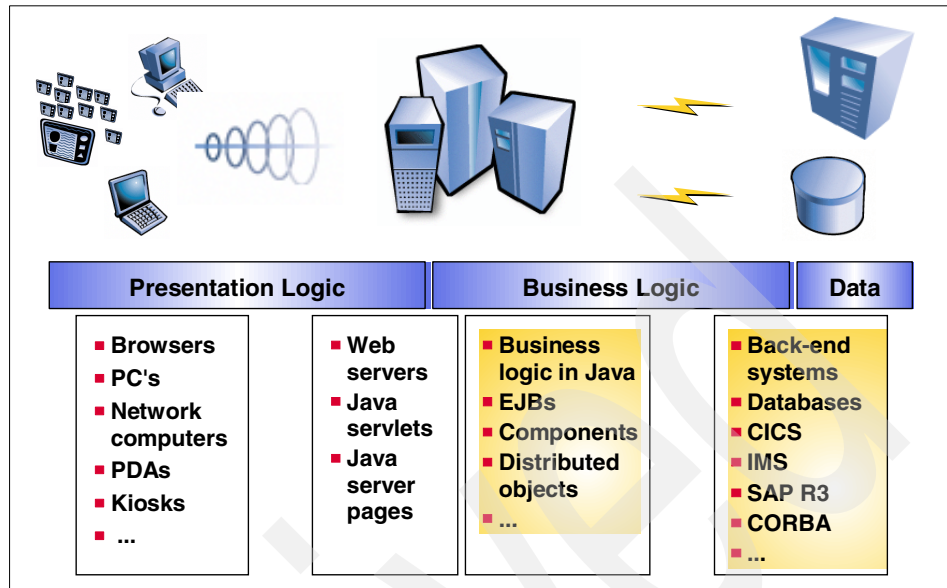


Figure 15-1 Application layers

The picture could be much more complex, especially in the business logic layer where you could spread the application code across many different layers. For example, you could have a message queuing layer, or an interface layer written in C, for communication between the two parts of the business logic, and you could build application logic in there as well.

This split of business logic can provide increased flexibility and scalability, by allowing you to split the application across more servers and add more servers at any point as required.

Note: On zSeries, you do not need to do this because you can run very large workloads on a single machine, and even larger ones in a Parallel Sysplex.

However, it also adds complexity, introduces more connections and opens up the potential for decreased performance and availability by having more components.

We suggest that you think very carefully about how you plan to split your application logic, and build an architected way of doing it so that each type of logic sits in a defined layer and interfaces with other layers in a structured manner.

This is especially important when you are linking together new Web application logic with existing applications; consider the following:

- ▶ Should you build all your new logic in the new WebSphere environment?
- ▶ Should you rewrite all of your existing logic in the new WebSphere environment?
- ▶ Should you put all the logic into your existing subsystem (for example, by adding new CICS transactions) and put the minimum possible logic in WebSphere?
- ▶ Should you build the application so that some logic is in WebSphere and some is in the existing subsystem, and, if so, which logic should reside where?

These are not simple questions. The best solution will depend on your particular circumstances and on where you plan to put your future development.

15.7 Object-oriented or traditional language

As part of your consideration of application layers, you need to consider whether your development effort is going to be focused on traditional procedural languages such as COBOL, PL/1, FORTRAN and C, or on object-oriented languages such as C++ and Java.

The advantage of coding in an object-oriented language is that the application logic is an abstract mapping of your business logic. For example, if your business deals with customers, your application will also have objects called *customers*. This close mapping makes it easier to develop applications and to modify them in the future. A customer is unlikely to change, although the way in which you deal with that customer (the interactions between objects) may change. This is easier to change in an object-oriented language than in a procedural language.

The main object-oriented languages in use today are C++ and Java. Of these, Java has the most interest because of some special features:

- ▶ It eliminates some capabilities in C and C++ that cause many of the programming errors. Thus it is faster for developing applications.
- ▶ It generates code that can be run on any platform (assuming that platform has a Java Virtual Machine). Therefore, it allows objects to be developed once, and they can then be deployed on any platform. This gives great benefits in terms of application reuse, for both independent software developers and in-house developers.
- ▶ With Enterprise JavaBeans, standard facilities are provided for producing advanced applications with transactional capability, security and persistent

data. These facilities are provided by the system and do not need to be coded explicitly by the application developer.

Table 15-1 shows a list of Java program types possible in an e-business application running on a z/OS server.

Table 15-1 Java components on z/OS

Java program type	When to use
Java applet	Displaying Graphical User Interface
HTML	Displaying text-based user interface in browser
Java servlet	Connection logic or non-transactional business logic
JavaServer Page	Building dynamic HTML-based user interface
Java (Access) Bean	Connection logic or non-transactional business logic
Enterprise JavaBean	Transactional business logic
DB2 Stored Procedure	Data access logic
CICS or IMS transaction	Transactional business logic and data access logic

There are benefits in using the new object-oriented technologies. However, there are also some reasons why you may want to use them in a limited way, at least for now:

- ▶ The technologies are new, and are therefore not proven for large scale deployment. While they hold the promise of many things, it is likely that the initial implementations will not be as complete, robust or highly performing as needed by some applications.
- ▶ Also because they are new, few people have experience in using them, and especially in using them well.

Newer technologies tend to perform less well than existing technologies for a couple of reasons:

- ▶ As we abstract more away from the hardware, we introduce additional layers in the code, which requires more processing power. We have seen this in moving from assembler to third-generation languages, such as COBOL, to fourth-generation languages. We also saw it in the move from flat files to VSAM to DB2 for data access. Now no one would suggest that we should go back to writing everything in assembler. As processing power gets cheaper, it

makes sense to simplify the application developers' job, and thus increase their productivity, even if it uses more processing power.

- Technologies improve in performance over time. Initially the focus is on getting the functions working with reasonable performance. Once the solution is available, work can be done to analyze which parts in the whole solution require performance improvements, and those improvements are then made. We have seen this happen with many products on many platforms over many years. Note that response time of new technologies is often good. From an end-user viewpoint, performance may be acceptable. However, the use of resources by the components may be higher than you want it to be.

15.8 Skills

Another factor to consider in your application design is the skills you currently have and the skills you want to develop. Today's marketplace allows you to buy any skills that you do not have. However, you need to consider that as part of your long-term direction. You will be more productive and deliver more robust applications if you focus on a few skill areas and build real experience in both the development and delivery parts of your organization.

15.9 Performance

We discussed performance earlier in terms of new technologies. The key point is to understand what the response time requirement of the application is, and then make sure that your design is able to deliver that response time.

Keep in mind that the response time probably includes not only the response time of a single component, but the aggregate of several components plus the time to communicate between them. Some of the connectors provide a more efficient connection than others, so this may be a consideration.

15.10 Scalability

Another consideration for performance is scalability. It is fine to have a solution that performs very well at low load, but you need to make sure that the solution will still perform well at the expected peak load (plus a safety margin). You need to consider what will happen if the load is significantly higher than you expect (which can happen on the Web) and what you would do to the solution to enable it to handle that.

15.11 Security

Security is also a consideration. We discuss this in detail in Chapter 6, “Security management” on page 75.

Think about the following:

- ▶ Client-to-server security

If this is across the Internet, you may want those messages encrypted. If you use an HTTP protocol between the Web browser and Web server, the encryption is easily provided using SSL. If not, you may need to specifically code that into the application.

- ▶ Security between servers

If you use SSL, the encryption ends at the Web server. Any communication between application layers behind that is not encrypted unless you code it in the application. That may or may not be an issue, depending on the application. It is probably more of an issue if you are sending messages between different physical servers, and less of an issue if you can stay in the same physical server as on S/390.

- ▶ Where and how you will do user authentication

A Web server can provide authentication for you, as can other application servers. Will you need to map user IDs as you move between application layers, or can you use the same ID for all layers?

- ▶ How you will do access control

This will vary depending on the application server.



Application development

Archived

16.1 Application development

We expect companies to move towards component-based development. We also expect there will be an evolution in how e-business applications are developed:

- ▶ Web enablement based on *first generation* solutions is typically very quick to implement with existing programming skills. Each solution is different. There are less standard tools, less portability and typically better performance than server-side Java connectors. There is no new business logic and little attempt to upgrade technology.
- ▶ *Second generation* solutions use the power of Java to provide new business logic to enhance the existing logic. Java connectors are used to connect to existing applications. Tools are available. These solutions are more focused on standards, tools, portability and beginning the process of application re-engineering.
- ▶ *Third Generation* solutions transform business processes through the use of component technology and Enterprise JavaBeans. These solutions are an evolution of the second generation with enhanced transactional and security context at the thread level, and data persistence built into the application programming model.

We discuss how you develop applications for each of these generations, and considerations for migrating from one generation to the next.

16.2 First Generation - non-Java

These solutions generally require little or no application development. The objective is to connect to your existing application logic and data in the easiest way possible. Most of the solutions are unique, based on the system you are connecting to.

Two of the solutions are worth specific mention here: MQSeries and Domino.

MQSeries

MQSeries provides a messaging service that allows you to connect to many different applications. The interface that you use is putting messages on a queue and taking messages off a queue.

In the First Generation scenario, you would access the queues using the standard MQSeries API. You code a C program that runs as a CGI program within the HTTP server, and that CGI program makes the calls to the MQSeries API.

MQSeries is also a valid Second Generation and Third Generation solution, but the way you call the MQSeries APIs is different.

Lotus Domino

Lotus Domino (the server for Lotus Notes clients) can be a First Generation and Second Generation solution. Domino provides you with the ability to write code in LotusScript or Java, and that code can access other systems through connectors. Thus, Domino allows you to enhance your current applications by making use of Notes facilities.

16.3 Migration from First Generation to Second and Third Generation

As you build your First Generation solution, you should also consider how you can migrate forward to Second and Third Generation should you wish to in the future.

Many of the First Generation solutions are very specific to a particular requirement and are not built to standards. Therefore, there is no forward migration path. To move to a Java-based solution you would throw away what you have and start again. However, since the First Generation solutions typically involve little or no application coding, you have very little to rework anyway. Also, the First Generation solutions can continue to coexist with later solutions for as long as you require them.

If you want to code application logic in your First Generation solution, we recommend that you consider writing as much as possible of that logic so that you can carry it forward, as follows:

- ▶ For DB2, stored procedures can contain application logic that is called by all types of solutions. A stored procedure is a group of SQL statements that form a logical unit and perform a particular task. Stored procedures are used to encapsulate a set of operations or queries to execute on a database server. One of the big advantages in using stored procedures is that the execution of one call statement on the application client can trigger any amount of work on the application server. We always recommend stored procedures for high volume access to DB2, whatever the application.
- ▶ For CICS, consider writing the logic in CICS transactions.
- ▶ If Domino is your application server, any LotusScript agents (code) will continue to work in the future, and Java code that you develop can coexist with it in the future.

- ▶ If you use MQSeries as a connector and build routing logic into it, all generations will be able to make use of that, although the specific calls to get MQSeries services change.
- ▶ For IMS, the first and second generations rely on IMS Connect.

16.4 Second Generation - Java

Second Generation solutions are built on Java. We believe that people will move towards Java for its platform independence and its support of object-oriented development (for more information, see 2.1, “Object-oriented technology” on page 12. In Second Generation solutions, we assume no use of Enterprise JavaBeans, since that technology is just now being delivered and therefore may not yet be available in the environment that you need or with the robustness that your application requires.

This section is intended to offer you some of the basic considerations in implementing Java applications on zSeries.

16.4.1 Coding in Java

The first requirement is to develop your application in Java. You may use a visual tool such as VisualAge for Java to do this. We discuss tools later. However, for a simple piece of code you could use an editor on z/OS to create the program in a UNIX file (the default file extension for source code is `.java`). You then compile it with the `javac` command into the executable file (executable by the JVM) with a default file extension of `.class` (in case of a single class file) or `.jar` (in case of an archive with multiple Java classes).

16.4.2 Object-oriented or not

Java does not require the use of object-oriented development methods. In fact, you have three choices:

- ▶ You can develop a procedural program in Java, using it as you would use any other language. In this case you would start using Java just as a new language with a new syntax.
- ▶ You can develop a program according to object oriented principles, and package it as a JavaBean. A sound transformation of an object oriented design into an implementation is key and the Java language would be used in the most object-oriented way possible.
- ▶ You can develop a program according to procedural principles, but wrap coding around it to make it into a JavaBean.

We recommend that you move towards the second option, true object-oriented development, but you don't have to. The third option provides a useful bridge between the procedural and object-oriented worlds.

16.4.3 Transactional or not

Consider whether your application requires a transactional capability. Does it update databases or initiate other transactions in such a way that you need updates to happen all together or not at all? For example, if you are allowing a customer to transfer money from one account to another, you need to make sure that if a failure occurs either the updates to both accounts happen, or neither happens. It is not acceptable to update one account and then, because of a failure, not to update the other one.

If you require transactional capability, you should think about how you can provide that. The Java 1 environment does not provide any services to help you. They come with EJBs, which are part of the Java 2 framework. The options you have are:

- ▶ Rely on another system, such as CICS or IMS, to provide the transactional capabilities. This is possible provided the transaction is contained within one of these environments, or you go to other environments using RRS.
- ▶ Write the code to manage transactions in your Java servlet or application. You need to code the calls to start, commit, and rollback transactions. Note that this work will be unnecessary in the Java 2 framework, as the EJB container will provide these services automatically.

16.4.4 Deployment options

Having decided to code your application in Java, you have a number of deployment options. You need to think about this before you begin to develop the application:

- ▶ From a Web and Java viewpoint, the purest deployment method is running under the control of a servlet engine such as WebSphere Application Server. This provides industry standard interfaces and services. This means that you can write code that can be deployed on any platform (provided that it supports WebSphere). This is a great advantage over CGI or GWAPI programs that must be recompiled when moving from one platform to another. WebSphere also provides a number of services that avoids the need for you to write those in your Java code. You can write both servlets and JavaServer Pages (JSP) and deploy them on S/390. See *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980 for more information on this.
- ▶ You could deploy your Java code as a standalone application. It could be used as a server program for Web browsers. It could be called from a CGI program.

Or, it could be started as a batch or UNIX application. Since it does not run under the control of a servlet engine, this application must handle more things for itself; for example, encryption between itself and the Web browser, and handling incoming requests. Depending on the function you require, you may need to make calls to platform-specific functions, in which case your code will not be portable.

- ▶ You could deploy your Java code into a Domino server. You would do this if you want to make use of the functionality of Domino.
- ▶ You could deploy your Java code in CICS. CICS provides a Java class library, known as JCICS, that allows you to access CICS resources and integrate your Java programs with programs written in other languages. Most of the functions of the CICS EXEC API are supported. If you build your application using these classes, it will be a CICS-specific program and cannot be deployed *as is* in any of the other deployment environments.
- ▶ Java can be deployed as a DB2-stored procedure. This is recommended if you need to code a complex query on DB2 tables. You can consolidate all DB2 output in a stored procedure and then send it forward to the calling Java servlet or applet. This is a better way of writing applications than including multiple SQL queries in a servlet or applet and processing all the results there.
- ▶ You could deploy your code to IMS using the IMS Connector for java based on the APPC (Advance Program to Program Communication) protocol.

We discuss the use of WebSphere Application Server further. The Domino and CICS implementations are discussed in the documentation for those products. The standalone Java application is probably of less interest to most people, since it does not provide a Web server function and requires you to write more code yourself.

16.4.5 Developing Java applications

You may develop your applications using a standard Java toolkit, such as VisualAge for Java. You may develop to the JavaBeans standards, and may include Beans supplied by other companies. For a Web application, you also need to develop Web pages. For this, you can use a tool such as IBM WebSphere Studio. These tools provide very rapid, visual application development, and they generate code that can be deployed on any platform. For more information on these tools, see *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980. Java developers can use the same tools to develop for S/390 that they use today for other platforms. There are a few areas you need to watch out for.

16.4.6 Standards

The first consideration is the *standard* to which you code. This area of information technology is evolving very quickly. As it does so, standards are produced for Java, JavaBeans, EJBs, and many other components. In some cases, new standards are incompatible with the previous one so that an application written to one level of the standard will not port to the next level (for example, the JNI interfaces have changed). In other cases, standards have not been fully agreed upon when products are shipped, so they may be slightly incompatible with the final published version.

You need to check that your development environment and proposed deployment environment are compatible, at least in the functions that you will use in your code. You should also validate that the techniques that you use do indeed deploy successfully. As each standard matures, this becomes less of an issue for that standard. Of course, new standards are coming along all the time.

16.4.7 Connectors

The second consideration is the use of *connectors*. The second generation connections for Java are based on standards, such as the Common Connector Framework (CCF), and Java Database Connectivity (JDBC) for access to relational databases.

CCF

CCF provides a common way to access multiple application servers, making it much simpler to develop applications. You no longer need to know the characteristics of a particular interface to a particular product.

The preferred way of using CCF connectors is in combination with the VisualAge for Java Enterprise Access Builders (EABs). The EAB provides an abstraction called a *command*. A command is a JavaBean that equates to a single interaction with another application server. A command can be used with any CCF-based connector, so it does not depend on the system you are communicating with.

CCF was succeeded by the J2EE Connector Architecture. See 16.6.3, “Connectors: J2EE Connector Architecture” on page 192.

16.4.8 Java Native Interface (JNI)

There are some situations when an application cannot be written entirely in Java, for example:

- ▶ The standard Java class libraries do not (yet) support some z/OS-specific features needed by the application.

- ▶ You want to implement a small portion of performance-critical code in a lower-level language such as C or assembler.
- ▶ You want to reuse already existing modules and libraries written in other programming languages, such as COBOL.
- ▶ You want to call Java code from other programming languages.

In these cases, you can make use of the Java Native Interface (JNI). The JNI allows you to:

- ▶ Call C/C++ functions from Java
- ▶ Call Java methods from C/C++
- ▶ Create, inspect, and update Java objects (including arrays and strings) and Java fields from C/C++
- ▶ Catch and throw Java exceptions from C/C++

The JNI defines a C/C++ interface to Java. Other programming languages are not supported. Therefore, to integrate a COBOL module in Java, you have to write a piece of C/C++ code, which in turn, performs the link to the COBOL module.

16.4.9 Deploying in WebSphere Application Server on z/OS or OS/390

OS/390 requirements

The minimum you need to run a Java application on OS/390 is:

- ▶ OS/390 Version 1 Release 1 or higher. A minimum of OS/390 Version 2 Release 8 (or higher) is recommended for support of the new powerful functions of JDK 1.3 and for performance enhancements.
- ▶ UNIX System Services enabled
- ▶ Language Environment 1.5 or higher
- ▶ Java for OS/390
- ▶ FTP or Network File System (NFS) enabled, for transferring files between a workstation and z/OS

We highly recommend that you always get the latest version of the JDK, because the product is improving in functionality and performance continuously. The software prerequisites of the various JDK releases can be found at:

<http://www.ibm.com/servers/eserver/zseries/software/java>

Deploying an application on z/OS

Once the code has been developed in VisualAge for Java and WebSphere Studio on the workstation, the main steps to deploy it on z/OS are:

1. Export your class files to WebSphere Studio from VisualAge for Java and create a WAR file. A WAR file is like a JAR file in the sense that it packages your Web application class files, and a deployment descriptor.
2. Use the Application Assembly Tool to create an EAR file, from the WAR file.
3. Use the System Management EUI to import the EAR file, and then export your application to the z/OS.

If you wish to deploy Java batch jobs to the z/OS, you can simply FTP your files to the z/OS.

More information about deployment of these applications can be found in *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980.

16.5 Migrating from Second Generation to Third Generation

As you build your Second Generation solution, you should also consider how you can migrate forward to Third Generation.

All of the Second Generation solutions are built on Java. Technically speaking, the Java code is reusable in a Third Generation environment. However, to ease your migration, we make the following recommendations:

- ▶ Any application logic that you write for things like persistence and transaction support are not required in Third Generation, since the EJB container provides those services. You will be able to run your Second Generation application code. For JavaBeans, you define them with Bean Managed Persistence (BMP). However, you may want to minimize your investment in such code that will not be necessary in the future.
- ▶ The connectors are also enhanced in Third Generation, providing more functions that your applications can use.

Almost all of your application logic that you code in Second Generation can be carried forward to Third Generation, provided that you code in a standard way. If you use items for which the standards change, you may need to do some code changes to meet the new standards. JavaBeans will be easily migrated to EJBs. Write Second Generation servlets in the object model, not the procedural model.

16.6 Third Generation - Enterprise JavaBeans

Third Generation solutions are built on Enterprise JavaBeans. This gives the benefits of platform independence and object-oriented development, but means that the system provides the standard functions needed for an enterprise-style application. For more information, see 2.5, “Java 2 Platform” on page 19.

16.6.1 Coding in Java

As for Second Generation solutions, Third Generation applications are developed in Java. You use the same tools as for the Second Generation. The process for deploying the applications is similar. The main difference is that you must use a development tool that can build EJBs (for example, VisualAge for Java V4.0). The deployment platform must provide support for EJBs (on z/OS, for example, WebSphere Application Server Enterprise Edition Version 4.0.1).

16.6.2 Transactional or not

In EJB technology, the assumption is that all applications (Beans) are transactional. However, they do not have to be. The container, implemented by WebSphere Enterprise Edition, for example, provides transactional capability without the need to code anything in the application. You define the Bean to have Container Managed Persistence (CMP) and the container will manage transaction start and end.

However, you can define a Bean to have Bean Managed Persistence (BMP). In this case, the container will not provide transactional services and the Bean must specifically manage the transactional state, if required.

16.6.3 Connectors: J2EE Connector Architecture

The J2EE Connector Architecture defines a standard architecture for connecting the Java 2 Platform Enterprise Edition (J2EE) platform to heterogeneous Enterprise Information Systems (EIS). Examples of EIS include transaction processing systems, such as CICS Transaction Server, and Enterprise Resource Planning systems such as SAP.

The complete J2EE Connector Architecture specification defined by the Java Community Process can be downloaded at:

<http://java.sun.com/j2ee/download.html#connectorspec>

The Connector Architecture enables an EIS vendor to provide a standard *resource adapter* for its EIS. A resource adapter is a middle tier between a Java application and an EIS, and permits the Java application to connect to the EIS. The types of Java applications include applets, servlets, and enterprise beans. A resource adapter plugs in to any application server supporting the J2EE Connector Architecture.

An application server vendor only needs to extend its system once to support the J2EE Connector Architecture, and is then assured of connectivity to multiple EISs. Likewise, an EIS vendor provides a standard resource adapter, and it has the capability to plug in to any application server that supports the J2EE Connector Architecture. This is shown in Figure 16-1.

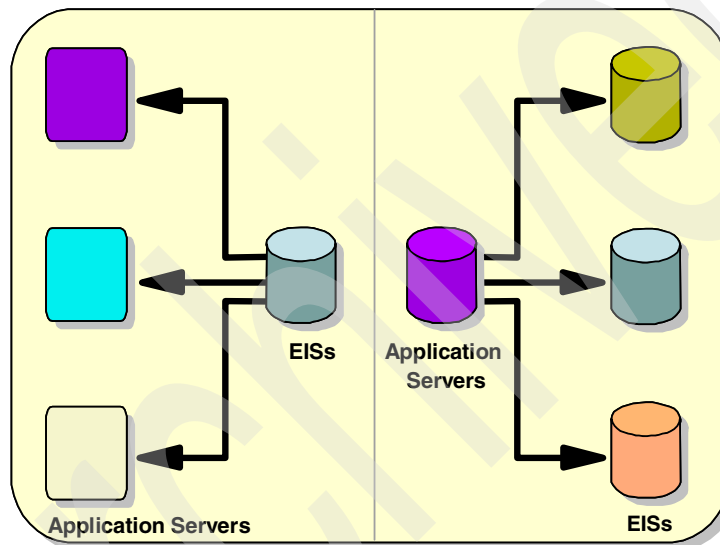


Figure 16-1 The role of resource adapters

16.6.4 Deployment options

To deploy a Third Generation solution, you must have an application server that provides an EJB container. WebSphere Application Server for z/OS 4.0.1 offers such a container. Deployment is similar to that of Second Generation deployment, in that you use the Application Assembly Tool and System Management End-User Interface (deployment tools). In *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980 we discuss this in more detail.

WebSphere or CICS

How do you choose between these options for running EJBs? Again you need to be clear on what you want to achieve and where you are starting from. These technologies are starting from different points—WebSphere from the object-oriented model and Java, needing to build transactional capabilities; and CICS from the transactional model, needing to build new object-oriented capabilities. They will both take some time to mature. Their backgrounds will mean that for the present they will be better suited to different requirements.

Development tools

Development tools, which we discuss in this chapter, are one of the key parts of the Application Framework for e-business. They provide the most effective way for customers and independent software vendors to build e-business applications. The recommended tools are fully integrated with IBM application servers. They allow you to make use of existing applications and data, and provide for easy development through the use of visual programming and code generation.

The tools that can be used for application development:

- ▶ WebSphere Studio
- ▶ VisualAge for Java
- ▶ WebSphere Studio Application Developer
- ▶ Lotus Domino Designer

In addition to these tools, one uses the Application Assembly Tool to assemble enterprise applications according to J2EE standards and the System Management EUI to convert the application to a format that the z/OS will be able to use.

17.1 WebSphere Application Development Solution

Before looking at the individual products that you would use for application development, we first describe a packaged offering called the WebSphere Application Development Solution V2 (product number 5655-ADS). IBM has packaged together its proven, enterprise application development software pre-loaded on S/390 hardware and integrated it with its leading workstation development tools. You can buy the component products separately, but this solution gives your development team a ready-to-run build and test environment for e-business on S/390. The host software is preloaded and preconfigured, so you can bypass installation and immediately use the new processor.

The WebSphere Application Development Solution consists of:

- ▶ An S/390 Multiprise 3000 processor, which provides a standalone application development and test system. This removes any dependencies on your current S/390 servers and helps you make a quick start.
- ▶ The S/390 server comes preloaded with z/OS and the application development tools that you need:
 - *Enterprise subsystems*: CICS Transaction Server, IMS/ESA, DB2, QMF, WebSphere and MQSeries
 - *Languages*: COBOL, PL/1, REXX, VisualAge Generator Host Services, Java for OS/390
 - *Tools*: Application Testing Collection, WebSphere Host-On-Demand, Tivoli NetView, CICS COBOL Conversion Aid, Fault Analyzer for OS/390, System Automation for OS/390
- ▶ The solution also includes a comprehensive set of development tools that run on a Windows NT workstation:
 - *VisualAge family*: COBOL Enterprise Edition, Java Enterprise Edition, Generator Developer
 - WebSphere Studio
 - Personal Communication

Recent versions of all of these products are provided so that you can use the latest function available.

17.1.1 Quick start scenarios

The S/390 Application Development Solution comes with 14 quick-start usage scenarios for application programmers, each with a sample application and a detailed tutorial, so you can quickly and easily tackle development and maintenance tasks such as:

- ▶ Web-enabling 3270 applications using WebSphere Host On-Demand. Host On-Demand can provide direct access to host terminals from a Java-enabled Web browser.
- ▶ Web-enabled CICS 3270 applications.
- ▶ Extending existing S/390 applications to the Internet and adding new function using VisualAge for Java. VisualAge for Java has the Enterprise Access Builders (EABs) that generate middleware code to connect to such middleware options as MQSeries, CICS, IMS, Host On-Demand, SAP R/3, RMI and IIOP.
- ▶ Testing applications as you migrate to new S/390 middleware and tools by using VisualAge COBOL and Application Testing Collection (ATC), including Automated Regression Testing Tool (ARTT).

The S/390 Application Development Solution helps you deliver e-business applications quickly. To learn more about the IBM S/390 Application Development Solution, see:

<http://www.ibm.com/servers/eserver/zseries/software/ads>

17.2 IBM VisualAge Family

The IBM VisualAge application development tools significantly increase productivity through features such as visual development environments, an ability to generate most infrastructure code, multilanguage support to make use of existing developer skills, remote debugging capability and team-based development.

The VisualAge Family consists of the following tools:

- ▶ VisualAge for Java
- ▶ VisualAge C++
- ▶ VisualAge Smalltalk
- ▶ VisualAge for COBOL, a visual development environment for COBOL
- ▶ VisualAge PL/1, a visual development environment for PL/1
- ▶ VisualAge Generator, which assists with code reuse and extending applications to the Web

Each provides a productive way to develop and maintain applications. More information about these products can be found on the Web at:

<http://www.ibm.com/software/ad>

Since the focus of this book is on e-business, particularly using Java, we discuss VisualAge for Java in more detail.

17.2.1 VisualAge for Java

VisualAge for Java is a visual tool for building applications in Java. As Figure 17-1 shows, it provides a visual interface that allows you to build applications in a modular way. It takes care of the linkage code required between the various components of the application, and allowing you to simply develop the specific business logic that you need.

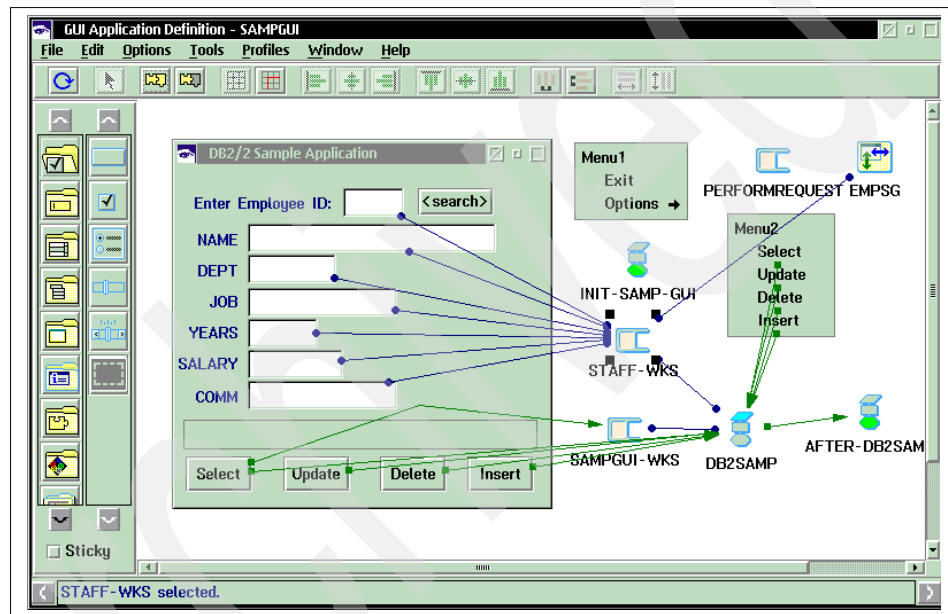


Figure 17-1 VisualAge for Java

Some of the benefits of using VisualAge for Java are:

- ▶ It supports development of business logic in individual components, and provides a framework for putting those components together. Design patterns and sample business content are provided, thus reducing the amount of coding that you need to do and reducing the number of potential errors that you introduce.
- ▶ It is designed for a team programming environment. Different roles and tasks are supported. It integrates closely with WebSphere Studio for Web site development.
- ▶ It supports development of Enterprise JavaBeans, thus eliminating the need to code the logic associated with transactions, persistence and other

enterprise requirements in your code. It supports the EJB specification, including wizards for the creation of session and Entity Beans, persistent mapping tools for CMP entity Beans, automatic creation of a test client, and advanced deployment tools.

- ▶ The persistent framework in VisualAge for Java forms the mapping layer of Entity Beans to existing data schemas. This framework allows Java developers to map EJB fields to existing database columns. Developers can generate EJBs from existing database tables, as well as generate database schemas from existing EJBs.
- ▶ It automates the generation of middleware connectors, using what are known as Enterprise Access Builders (EABs). Using the EAB wizard, you can easily create Beans and EJBs that facilitate access to CICS, IMS, Domino, MQSeries, SAP R/3, RMI, IIOP and other applications. Developers, therefore, do not need to concern themselves with the specific interface to each application type, nor do they need to write data conversion code. They can spend their time developing business logic, rather than learning middleware APIs.
- ▶ It provides Access Beans for easy connection to relational databases such as DB2. It also provides tools to simplify the coding and deployment of DB2 stored procedures.
- ▶ It produces code that runs in WebSphere Application Server, including servlets, JSPs and JavaBeans.
- ▶ It includes powerful debugging facilities.
- ▶ It has an XML Metadata Interchange (XMI) bridge to business models developed in Rational Rose. Once you have developed your business model in Rational Rose, you can use the bridge to generate the corresponding Java code in VisualAge. You can also generate new models in Rational Rose from Java code in VisualAge.

VisualAge for Java Version 4.0 contains an environment that allows developers to build applications that target the Java 2 Platform.

It supports many platforms for development, including Windows NT/2000. It supports many platforms for testing and debugging, including z/OS, OS/400, AIX, Windows, and OS/2.

17.3 IBM WebSphere Studio

WebSphere Studio provides everything you need to start creating Web sites. It simplifies Web site development by providing these features:

- ▶ A Visual page designer that helps you lay out Web pages.

- ▶ Support for dynamic Web pages using JavaServer Pages (JSPs), and a JSP debugging facility.
- ▶ Wizards to help develop and debug Java code and JSPs.

For more information about this product, see:

<http://www.ibm.com/software/ad>

17.4 WebSphere Studio Application Developer

WebSphere Studio Application Developer is the follow-on technology for WebSphere Studio, Professional and Advanced Editions and VisualAge for Java Enterprise Edition. These new tools support the 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
- ▶ Increased speed to market

Application Developer provides integrated development tools for all e-business development roles. These roles include Web developers, Java developers, business analysts, application architects, and enterprise application programmers.

Application Developer brings together the most popular features of WebSphere Studio “Classic” and VisualAge for Java, and combines them with the advantages of the latest technology, providing open standards, tool integration, more flexibility, and the ability to tie in existing applications.

Figure 17-2 on page 201 shows all the tasks you can manage with Application Developer.

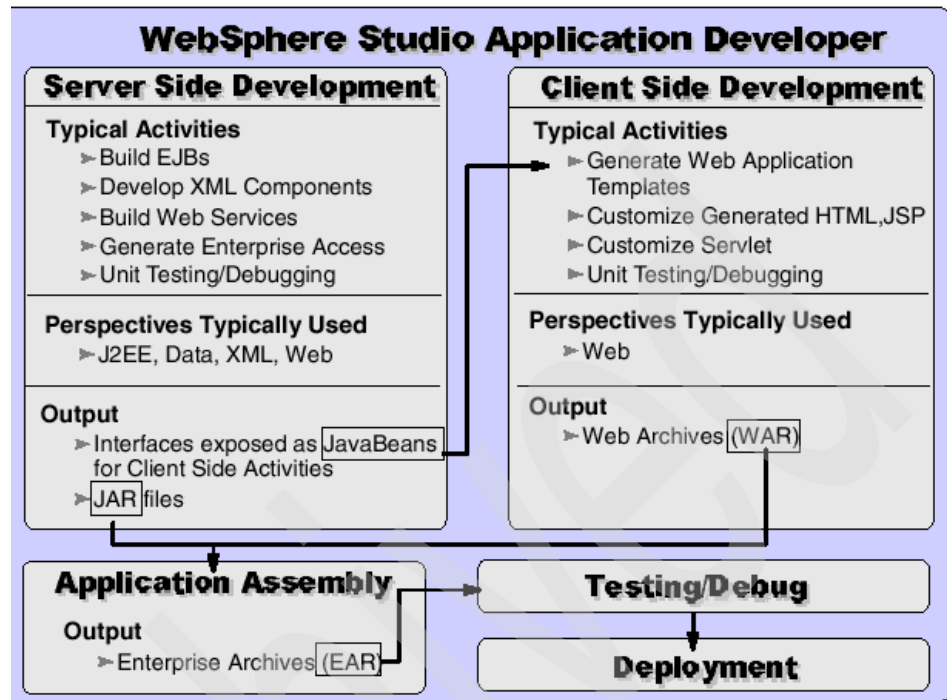


Figure 17-2 Building a J2EE application with WSAD

IBM WebSphere Studio Application Developer Integration Edition for Windows®, V4.1 is a member of the WebSphere Studio product family. Application Developer Integration Edition provides J2EE™ developers and application integration specialists with an easy-to-use, integrated development environment for building, testing, integrating and deploying J2EE applications.

WebSphere Studio Application Developer Integration Edition includes all of the functionality in WebSphere Studio Application Developer, plus:

- ▶ Powerful graphical tools to help you quickly and easily build custom application adapters to integrate your J2EE application with your back-end systems, thus allowing you to save time and money by reusing your existing resources.
- ▶ Visual flow-based tools that increase developer productivity by allowing them to visually define the sequence and flow of information between application artifacts such as adapters, Enterprise JavaBeans™ components, Web services, or other flows.
- ▶ Wizards that help you build and deploy complex Web services out of adapters, EJB components, flows, and other Web services.

- ▶ Support for the full set of Enterprise services provided by WebSphere Application Server Enterprise Edition such as Business Rule Beans, internationalization, and work areas that deliver additional integration capabilities, developer productivity, and business agility.

17.5 Lotus Domino Designer

Lotus Domino Designer, like WebSphere Studio, allows you to create content and manage powerful e-business sites. It integrates with VisualAge to provide a complete team programming environment. For more information about this product, see:

<http://www.lotus.com/home.nsf/welcome/dominodesigner>

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 205.

- ▶ *e-business Cookbook for z/OS, Volume II: Infrastructure*, SG24-5981
- ▶ *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980
- ▶ *Enterprise JavaBeans for z/OS and OS/390 WebSphere Application Server V4.0*, SG24-6283
- ▶ *IBM SecureWay Host On-Demand 4.0: Enterprise Communications in the Era of Network Computing*, SG24-2149
- ▶ *Stay Cool on OS/390: Installing Firewall Technology*, SG24-2046
- ▶ *Ready for e-business: OS/390 Security Server Enhancements*, SG24-5158
- ▶ *Java 2 Network Security*, SG24-2109
- ▶ *OS/390 Security Server 1999 Updates Technical Presentation Guide*, SG24-5627
- ▶ *IBM Component Broker on System/390*, SG24-5127
- ▶ *CICS Transaction Server for OS/390 Version 1 Release 3: Web Support and 3270 Bridge*, SG24-5480
- ▶ *Enterprise JavaBeans for z/OS and OS/390, CICS Transaction Server V2.1*, SG24-6284
- ▶ *Enterprise JavaBeans for z/OS and OS/390 WebSphere Application Server V4.0*, SG24-6283
- ▶ *EJB Development with VisualAge for Java for WebSphere Application Server*, SG24-6144

Other resources

These publications are also relevant as further information sources:

- ▶ *CICS Internet Guide*, SC34-5713
- ▶ *HTTP Server Planning, Installing, and Using*, SC34-4826
- ▶ *OS/390 UNIX System Services Planning Guide*, SC28-1890
- ▶ *Firewall Technologies Guide and Reference*, SC24-5835
- ▶ *z/OS V1R2.0 ICSF Overview*, SA22-7519
- ▶ *SecureWay Security Server LDAP Server Administration and Use*, SC24-5923

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ Enterprise JavaBeans
<http://java.sun.com/products/ejb/index.html>
- ▶ XML project
<http://xml.apache.org/index.html>
- ▶ z/OS XML site
<http://www.ibm.com/servers/eservers/zseries/software/xml>
- ▶ z/OS Security site
<http://www-1.ibm.com/servers/eserver/zseries/zos/security/>
- ▶ “Security on z/OS: Comprehensive, current, and flexible,” published in the IBM Systems Journal, Volume 40, Number 3, 2001
<http://www.research.ibm.com/journal/sj/403/guski.html>
- ▶ J2EE technology site
<http://java.sun.com/j2ee/>
- ▶ J2EE Connector Architecture download
<http://java.sun.com/j2ee/download.html#connectorspec>
- ▶ To get the latest JDK for z/OS
<http://www.ibm.com/servers/eserver/zseries/software/java>
- ▶ WebSphere Application Development Solution for OS/390
<http://www.ibm.com/servers/eserver/zseries/software/ads>

- ▶ IBM e-business application development tools
<http://www.ibm.com/software/ad>
- ▶ Information about Lotus Domino development tools
<http://www.lotus.com/home.nsf/welcome/dominodesigner>

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the CD-ROMs button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

A

Access Control Environment Element (ACEE) 87
access control list (ACL) 78
Apache XML project 39
application client 26, 34–35
application client container 26
application development
 objectives 174
application server 193
Application Testing Collection (ATC) 197
auditing 76
Authentication 76, 80
Authorization 76, 80
Automated Regression Testing Tool (ARTT) 197

B

basic mapping support (BMS) 155
Bean Managed Persistence (BMP) 192
business logic interface (BLI) 154, 156
Business Logic Layer 29
business rule 29
bytecode 17

C

CICS Universal Client 150, 157
CICS Web Support (CWS) 149
CICS Web support (CWS) 150
 See also 3270 Web bridge
 See also business logic interface
 See also CICS WebServer Plugin
 COMMAREA manipulation 152
CICS WebServer Plugin 154
class 13
Client certificates 81
COMMAREA 149, 154
Common Client Interface (CCI) 159
Common Connector Framework (CCF) 189
Common Gateway Interface (CGI) 114
Common Object Request Broker Architecture 25
confidentiality 76
connectors
 CICS 147, 165

CICS Web Support (CWS) 149
first generation connectors, overview 55
IMS TCP/IP OTMA Connector (IMS TOC) 167
introduction 55
second generation (Java) connectors, overview 55
third generation (Enterprise Java) connectors, overview 55

Container Managed Persistence (CMP) 192
containers 80
cookie 118
CORBA
 CICS support of 148
CORBA. *See* Common Object Request Broker Architecture
Credential 80
CSMI, mirror transaction 154
CTG
 Java class library 158

D

data integrity 76
Data Stores 29
DB2
 Java Database Connectivity 143
 stored procedures in Java 143
declarative security 76
DeMilitarized Zone 94
Deployment 33
deployment descriptor 33–35, 77
DFHWWAPI, CICS WebServer Plugin 151
directory services 90
distributed program link (DPL) 149
DMZ 94
DOCUMENT API 152
Document Type Definition 35
DTD. *See* Document Type Definition

E

EAR. *See* Enterprise Archive
e-business scenarios 59
 components 60
 new business logic 66

- overview 60
- solution types 60
- Web serving 61
- Web-enabling current systems 62
- EJB container 27
- EJBROLE 89
- eNetwork Communications Server 93
- Enterprise Access Builder (EAB)
 - command 189
- Enterprise Access Builders (EABs) 189, 197, 199
- Enterprise ARchive 35
- Enterprise Information System 28
- Enterprise Information Systems (EIS) 192
- Enterprise Java Server (EJS) 160
- Enterprise JavaBean 85
- Enterprise JavaBeans 21, 24, 27, 35
 - types 32
- Enterprise JavaBeans (EJB) 160
 - support in WebSphere 66
- Enterprise JavaBeans (EJBs) 30
- Entity Bean 32
- EXCI 150
 - use by CICS WebServer Plugin 151
- EXEC CICS DOCUMENT 152
- EXEC CICS WEB 152
- Experimental API's 39
- Extensible Markup Language 35
- Extensible Markup Language (XML)
 - benefits 37
 - components, optional 38
 - Document Type Definition files (DTD) 38
 - Extensible Link Language (XLL) 38
 - Extensible Style Language (XSL) 38

F

- Fast Response Cache Accelerator (FRCA) 119–120
- FastCGI 116
 - advantage 116
- file caching
 - HTTP server caching HFS files 120
 - HTTP server caching MVS data sets 120
 - UNIX System Services caching HFS files 120
- filecache command 120
- Firewall
 - types 92
- firewall 91
 - IP filtering 92

- Network Address Translation 92
 - on OS/390 93
- Virtual Private Network 92
- Form-based authentication 81

G

- getCallerPrincipal 77
- getUserPrincipal 76
- Go Web Server API (GWAPI) 115
- Go Webserver API (GWAPI) 150–151

H

- Host on Demand 197
 - overview 65
- HTML
 - disadvantages 37
- HTTP 23
 - persistent connections 118
 - virtual hosts 119
- HTTP basic authentication 81
- HTTP digest authentication 81
- HTTP header 82
- HTTP server
 - external program 110
 - incoming communication 112
 - outgoing communication 114
 - overview 110
 - protocols supported 112
- HTTP server plug-ins
 - ServerInit 114
 - ServerTerm 114
 - Service 114
 - WebSphere Application Server (WAS) 115
- HTTPS 23
 - port number 112
- HTTPS basic authentication 81
- HyperText Transfer Protocol (HTTP) 112
- Hypertext Transfer Protocol (HTTP) 112

I

- IBM HTTP Server for OS/390
 - CacheLocalFile directive 120
 - certificate authentication 119
 - cookies support 118
 - Fast Response Cache Accelerator (FRCA) 120
 - file caching 119
 - HTTPS/SSL support 119

- LDAP support 119
 - Multi Format Processing support 118
 - proxy support 119
 - security functions 119
 - Server Side Includes (SSI) 117
 - Simple Network Management Protocol (SNMP) 118
 - SMF recording 117
 - thread level security 119
 - tracing and logging 117
 - ICAPI
 - ICAPI DLL, *See DFHWBAPI*
 - See also GWAPI*
 - Identification 76
 - IDL. *See* Interface Definition Language
 - IIOP. *See* Internet Inter-ORB protocol
 - IMS Connect 167
 - Command Component (CMD) 167
 - components 167
 - Datastore Communication Component (DCC) 167
 - Workstation Communication Component (WCC) 167
 - inner firewall 95
 - Interface Definition Language 25
 - Internet Inter-ORB protocol 25
 - Internet Security Association Key Management Protocol (ISAKMP) 93
 - IP filtering 92
 - IP packet filtering 93
 - IP Security 93
 - IP tunnel 92
 - isCallerInRole 77
 - isUserInRole 76
- J**
- J2EE 20–22, 134
 - J2EE application 35
 - J2EE Connector Architecture 192
 - J2EE Connectors 86
 - J2EE container 26
 - J2ME *See* Java 2 Micro Edition
 - J2SE *See* Java 2 Standard Edition
 - J2SE. *See* Java 2 Standard Edition
 - JAF *See* JavaBeans Activation Framework
 - JAR *See* Java Archive
 - Java 15
 - development approaches 186
 - transactional capabilities 187
 - Java 2 Enterprise Edition 19–20, 28–29
 - Java 2 Micro Edition 19
 - Java 2 Standard Edition 19, 27
 - Java Archive 22, 26, 34–35
 - Java Database Connectivity 24
 - Java Database Connectivity (JDBC) 86, 142–143, 189
 - local drivers 143
 - on OS/390 146
 - remote drivers 143
 - Type 1 driver 144
 - Type 2 driver 144
 - Type 3 driver 145
 - Type 4 driver 145
 - Java Messaging Service 24
 - Java Messaging Service (JMS) 86
 - Java Naming and Directory Interface 24
 - Java Naming and Directory Interface (JNDI) 161
 - Java principal 88
 - Java Runtime Environment 19
 - Java Transaction API 25
 - Java Transaction Service 25
 - Java Virtual Machine 19, 25
 - Java Virtual Machine (JVM) 17
 - Java2 134
 - JavaBean 15
 - JavaBeans
 - customization 16
 - events 16
 - introspection 15
 - persistence 16
 - properties 16
 - JavaBeans Activation Framework 25
 - javac command 186
 - JavaMail 25
 - JavaServer Pages 21
 - JRE. *See* Java Runtime Environment
 - JSP (JavaServer Page) 30
 - JTA. *See* Java Transaction API
 - JTS. *See* Java Transaction Service
 - Just-In-Time (JIT) compiler 18
- L**
- Lightweight Directory Access Protocol 75
 - Lightweight Directory Access Protocol (LDAP) 90, 119

M

Message Queue Interface (MQI) 139
Method
 lookup 24
method, calling 14
migration
 from generation 1 to generation 2 185
 from generation 2 to generation 3 191
MQSeries 24, 138
MQSeries Queue Manager (MQM) 139
Multi Format Processing 118

N

Network Address Translation 92
Network Address Translation (NAT) 93
non-repudiation 76

O

object 13
 life cycle 30
 object state 30
object oriented programming 14
Object Request Broker 25
object, methods 13
object, properties 13
outer firewall 95

P

Parallel Sysplex 87
Presentation Layer 28
Principal 80
Principal name 80
programmatic security 76
programming
 business logic 149
 COMMAREA manipulation 152
 presentation logic 149
 See also ECI, EPI, ESI
 Web-aware 152
proxy server 92
Public API's 39

R

RACF 84
Redbooks Web site 205

Remote Method Invocation 25
resource adapter 193
Resource Adapter Archive 160
Resource Management Facility (RMF) 99
Resource Recovery Services (RRS) 71
RMI. *See* Remote Method Invocation
RMI/IIOP 25
role-based model 78

S

S/390 Application Development Solution
 quick start scenarios 196
SDK 19
Secure Socket Layer (SSL) 82
SecureWay Security Server 93
Security domain 80
security exposures 92
security policy 77
Security role 80
security roles 77
server regions 87
Server Side Includes (SSI) 117
servlet 29
Session Bean 32
Simple Network Management Protocol (SNMP)
118
single signon 84
SNMP Management Information Base (MIB) 118
Socks servers 93
socksified 93
Software Development Kit 19
SQLJ 143, 146
 Database Request Module (DBRM) 146
 Part 0 146
 Part 1 146
 Part 3 146
 serialized profile 146
SQLJ specification 146
Subject 80
System Authorization Facility (SAF) 87
System contracts 159
System Management Facilities (SMF) 99

T

transactions
 in e-business 70
two-phase commit protocol 71

U

Unit of Work (UOW) 71

V

Virtual Private Network 92

W

WAR. See Web Archive

WEB API 152

Web application 35

Web Archive 34

Web container 27

Web-aware, CICS application 150, 152

WebSphere Application Server 122

WebSphere Studio

features 199

Workload Manager (WLM) 99

compatibility mode 101

goal mode 102

X

X.500-compliant directory 119

XML Metadata Interchange (XMI) 199

XML4J 39

Z

z/OS SecureWay Security Server (RACF) 88

e-business Cookbook for z/OS Volume I: Technology Introduction

(0.2" spine)
0.17" <-> 0.473"
90 <-> 249 pages



Redbooks

e-business Cookbook for z/OS Volume I: Technology Introduction

**Planning your
software for
e-business
enablement on the
z/OS platform**

**Functions and
features of
e-business software
on z/OS**

**Sample scenarios
and technology
options**

This IBM Redbook is the first volume in the “e-business Cookbook for z/OS” series. It provides an overview of e-business and how it relates to zSeries and its z/OS operating system.

The technology portfolio on z/OS in the e-business enablement arena is growing rapidly. Multiple solutions are possible, each requiring a different set of products. In this book we define several scenarios, each using a different “style” of technology. The scenario you adopt is dependent on your strategy, the skills in your organization, and the functionality you need. Within that scenario, various technology options are available; your selections will depend mostly on the subsystems deployed in your organization.

The other volumes in this series are *e-business Cookbook for z/OS, Volume II: Infrastructure*, SG24-5981, and *e-business Cookbook for z/OS, Volume III: Java Development*, SG24-5980. They address the implementation and configuration of the technology, and describe how to develop solutions.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks