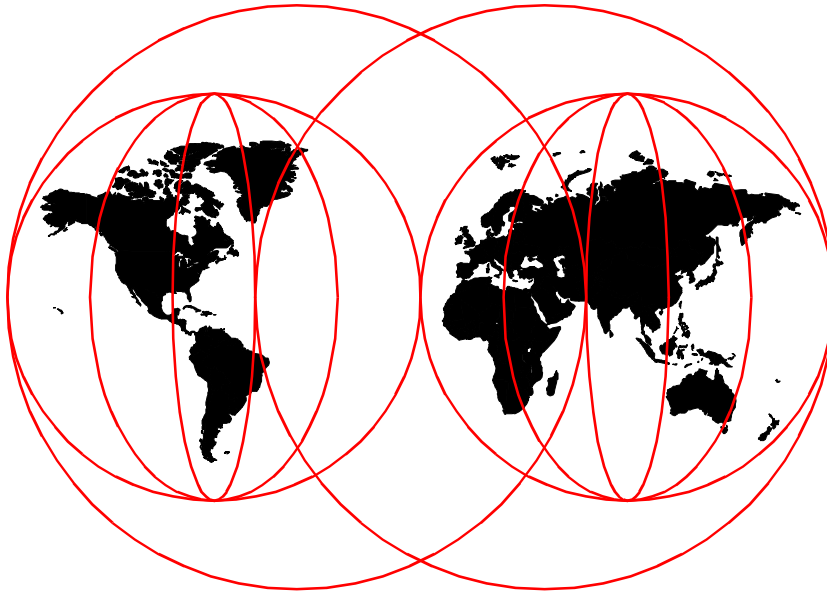


Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition

Carla Sadtler, Florian Hilgenberg, Joseph Kwek, Leo Marland, Witold Szczepońnik, Guru Vasudeva



International Technical Support Organization

www.redbooks.ibm.com

SG24-5864-00



International Technical Support Organization

**Patterns for e-business:
User-to-Business Patterns for Topology 1 and 2
using WebSphere Advanced Edition**

April 2000

Take Note!

Before using this information and the product it supports, be sure to read the general information in Appendix A, "Special notices" on page 337.

First Edition (April 2000)

This edition applies to Version 3.021 of IBM WebSphere Application Server, Advanced Edition, Program Number 41L0696 for use with the Windows NT, AIX, and Solaris operating systems.

Comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 678
P.O. Box 12195
Research Triangle Park, NC 27709-2195

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. All rights reserved.

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

Preface	ix
The team that wrote this redbook	ix
Comments welcome	xii
 Chapter 1. Introduction to patterns	1
1.1 Patterns for e-business	1
1.2 How to use these patterns	3
1.3 Patterns for e-business Web site	4
1.4 The Application Framework for e-business	4
1.5 An integrated view of e-business solutions	6
1.6 Structure of this redbook	7
 Part 1. User-to-Business patterns: topologies 1 and 2	9
 Chapter 2. Choosing the application topology	11
2.1 Application topology 1	11
2.1.1 Application topology 1: business driver	12
2.1.2 Application topology 1: key features	12
2.1.3 Application topology 1: considerations	14
2.2 Application topology 2	14
2.2.1 Application topology 2: business driver	14
2.2.2 Application topology 2: key features	15
2.2.3 Application topology 2: considerations	16
 Chapter 3. Choosing the runtime topology	19
3.1 An introduction to the node types	20
3.1.1 Web application server	20
3.1.2 Public Key Infrastructure (PKI)	21
3.1.3 Domain Name Service (DNS) node	21
3.1.4 User node	21
3.1.5 Directory and security services node	21
3.1.6 Database server node	21
3.1.7 Protocol firewall and domain firewall nodes	22
3.1.8 Load balancer node	22
3.1.9 Shared file system node	22
3.1.10 Web server redirector node	22
3.1.11 Application server node	23
3.1.12 Existing applications and data node	23
3.2 Runtime topology A	23
3.2.1 Proven basic topology	23
3.2.2 Proven variation 1	25

3.2.3	Emerging variation 2	27
3.2.4	Emerging multi-tier variation 3	29
3.3	Runtime topology B	31
3.3.1	Proven basic topology	32
3.3.2	Proven variation 1	33
3.3.3	Emerging variation 2	36
3.3.4	Emerging multi-tier variation 3	38
3.4	Intranet vs. Internet runtime topologies.	40
Chapter 4. Product mapping		43
4.1	Runtime topology options	43
4.1.1	Implementing a redirector	43
4.1.2	Clones running on application servers	45
4.1.3	Session sharing across servers	46
4.1.4	Achieving HTTP session affinity	47
4.2	Product mapping for basic runtime topology A	48
4.3	Product mapping for variation 1 of runtime topology A	49
4.4	Product mapping for variation 2 of runtime topology A	51
<hr/>		
Part 2. User-to-Business patterns: guidelines		53
Chapter 5. Performance guidelines		55
5.1	Web server performance considerations	55
5.2	Integration server performance considerations	59
5.3	Java and Java Virtual Machines	60
5.3.1	Just-In-Time compiler	60
5.3.2	Adaptive compilers	61
5.3.3	Static compiler	61
5.3.4	Selecting JVMs	61
5.4	Where to find more information	62
Chapter 6. Technology options		65
6.1	Web client	66
6.1.1	Web browser	67
6.1.2	HTML	68
6.1.3	Dynamic HTML	68
6.1.4	XML (client-side)	69
6.1.5	JavaScript	70
6.1.6	Java applets	70
6.2	Web application server	72
6.2.1	Java servlets	73
6.2.2	Java Server Pages (JSPs)	74
6.2.3	JavaBeans	74

6.2.4 XML	75
6.2.5 JDBC and SQLJ	75
6.2.6 Enterprise JavaBeans	77
6.2.7 Connectors	78
6.2.8 Additional enterprise Java APIs	79
6.3 Where to find more information	79
Chapter 7. Application design guidelines	81
7.1 Application elements	82
7.2 Understanding supporting technologies	86
7.2.1 Java servlets	86
7.2.2 Java Server Pages (JSPs)	90
7.2.3 JavaBeans and Enterprise JavaBeans	91
7.3 Application structure	92
7.3.1 Model-View-Controller (MVC) design pattern	94
7.3.2 MVC design pattern example	98
7.3.3 Advantages and disadvantages of the MVC design pattern	102
7.4 Application component contracts	104
7.4.1 Result beans and View beans design pattern	105
7.4.2 Result bean and View bean design pattern example	107
7.4.3 Advantages and disadvantages of Result beans and View beans	114
7.5 Application output formatting	115
7.5.1 Formatter beans	116
7.5.2 Formatter bean example	116
7.5.3 Advantages and disadvantages of Formatter beans	117
7.6 Application business logic granularity	117
7.6.1 Command beans	118
7.6.2 Command bean example	119
7.6.3 Advantages and Disadvantages of Command beans	126
7.6.4 An alternative approach	126
7.7 Application session management	128
7.7.1 Session management example	129
7.7.2 Session management design considerations	137
7.8 Application Security	140
7.8.1 Other design considerations	143
7.9 Conclusion	145
7.10 Where to find more information	145
Chapter 8. Application development guidelines	147
8.1 The development process	147
8.2 The scope of this chapter	148
8.3 The application and architecture domains	149

8.4	Solution outline	150
8.5	Macro design	151
8.6	Micro design	153
8.6.1	Use case	154
8.6.2	Class model and class diagram	157
8.6.3	Interaction diagram	163
8.6.4	State diagram	166
8.6.5	Component model	166
8.6.6	Deployment model	169
8.7	Build cycle	172
8.7.1	Develop source code	172
8.7.2	Testing	188
8.8	Deployment	193
8.9	Where to find more information	195
Chapter 9. System management products and guidelines		197
9.1	Managing your WebSphere application	199
9.1.1	WebSphere resource management	200
9.1.2	Using the WebSphere administrative console	202
9.1.3	WebSphere Site Analyzer	205
9.2	User-to-business WebSphere end-to-end security	208
9.2.1	Physical systems security	208
9.2.2	Operating systems security	208
9.2.3	Network security	209
9.2.4	Web application security	211
9.2.5	WebSphere Application Server security model and policy	213
9.2.6	HTTP Single Sign-On (SSO)	219
9.2.7	WebSphere V3 security differences with V2	219
9.3	Backup and recovery of your systems	220
9.3.1	Using Tivoli Storage Manager	221
9.3.2	Application backup and recovery	225
9.3.3	Guidelines for backup and recovery	227
9.4	Where to find more information	228
Part 3. Application topology 1: a working example		229
Chapter 10. The Pattern Development Kit and an example topology		231
10.1	The Pattern Development Kit	232
10.2	PDK section A	233
10.2.1	PDK application interaction	236
Chapter 11. Step 1: Modifying the PDK application		239
11.1	Using the Pattern Development Kit in VisualAge for Java	239

11.1.1	Changing the PDK application in VisualAge for Java	239
11.2	Using the PDK in WebSphere Studio	244
11.2.1	Changing the PDK application with WebSphere Studio.	244
Chapter 12.	Step 2: Expanding the PDK to multiple machines.	249
12.1	Setting up the network environment	249
12.2	Separating the application server from the Web server.	250
12.3	Setting up the application server on Machine B	251
12.3.1	Creating the JDBC driver and DataSource definition.	252
12.3.2	Create an application server.	254
12.3.3	Set up the application file structure.	260
12.3.4	Define the servlet	262
12.4	Separating the database from the Web application server	268
12.5	Testing the application	273
Chapter 13.	Step 3: Securing the PDK application	275
13.1	Enabling application security in WebSphere	275
13.2	Enabling WebSphere global security	276
13.2.1	Protecting the application	280
13.3	Hints and tips	286
Chapter 14.	Step 4: Cloning an application server	287
14.1	Topology and product mapping	287
14.2	Preparing the WebSphere administrative domain	288
14.3	Creating a model	289
14.4	Creating the first clone	294
14.5	Creating the next clone	295
Chapter 15.	Setting up a standalone servlet redirector	297
15.1	Creating a standalone redirector	297
15.2	Preparing to use firewalls	301
15.3	Testing the redirector	304
Chapter 16.	Setting up firewalls	305
16.1	Designating the network interfaces.	306
16.1.1	Domain firewall.	306
16.1.2	Protocol firewall	307
16.2	Setting up the general security policy	307
16.3	Creating the network objects	308
16.3.1	Domain firewall.	308
16.3.2	Protocol firewall	309
16.4	Configuring the domain name service.	310
16.5	Creating the firewall rules and services	310
16.5.1	Domain firewall rules and services	310

16.5.2 Protocol firewall rules and services.	319
16.6 Building connections on the firewall	321
16.6.1 Domain firewall.	322
16.6.2 Protocol firewall	322
16.7 TCP/IP routing	323
Chapter 17. SecureWay Directory Configuration	325
17.0.1 Administering the SecureWay Directory LDAP server.	327
17.0.2 Working with the directory tree	328
17.0.3 Using DMT to add your own directory entries	330
17.0.4 Use DMT to add a new user and assign a password.	334
Appendix A. Special notices	337
Appendix B. Related publications.	341
B.1 IBM Redbooks publications	341
B.2 IBM Redbooks collections.	341
B.3 Other resources	342
17.1 Referenced Web sites	344
How to get IBM Redbooks	345
IBM Redbooks fax order form	346
Index	347
IBM Redbooks review	351

Preface

Patterns for e-business are a group of proven, reusable assets that can help speed the process of developing applications. The pattern discussed in this book, the User-to-Business pattern, is the general case of users interacting with enterprise transactions and data. In particular it is relevant to those enterprises that deal with goods and services which cannot be listed and sold from a catalog.

This redbook discusses two application topologies of the User-to-Business patterns. Application topology 1 describes a situation where you are building an application that has no need to connect to back-end or legacy data. Topology 2 extends topology 1 to describe the situation where you need to access existing data on legacy or third-party systems.

Part 1 of the redbook takes you through the process of choosing an application topology and a runtime topology. It then gives you possible product mappings for implementation of the chosen runtime topology.

Part 2 is a set of guidelines for building your e-business application. It includes information on application design, technology options, application development, performance, and security.

Part 3 takes you through a working example, showing the implementation of an e-business application using application topology 1.

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, Raleigh Center.

The overall manager for Patterns:

Jonathan Adams is an IT consultant with IBM's Software Group and the leader of the Patterns for e-business initiative. He works closely with all areas of IBM and industry consultants. His commitment to the idea of a systematic approach to end-to-end e-business architecture is based on his many years of work in the field with major IBM customers in the United Kingdom.

The ITSO advisor for this project:

Carla Sadtler is a Senior Software Engineer at the International Technical Support Organization, Raleigh Center. She writes extensively in many areas including WebSphere, SecureWay Communications Servers, network

integration, and Web-to-host integration products. Before joining the ITSO 14 years ago, Carla worked in the Raleigh branch office as a Program Support Representative. She holds a degree in mathematics from the University of North Carolina at Greensboro.

Lead technical advisor and author:

Guru Vasudeva is an IT Architect with the e-business National Practice, IBM Global Services, United States. He has had more than eight years of experience in the software engineering arena, specializing in Internet architectures, client/server solutions, component-based development, and object-oriented technologies with a particular focus on the insurance and telecommunications industries. Mr. Vasudeva has successfully led several complex software design and development efforts during his career. He holds a computer science engineering degree from Mysore University, India.

Authors:

Florian Hilgenberg is an IT specialist in IBM Global Services Germany. He has more than four years of experience in application development, working as an application developer and consultant on object-oriented software for customer projects in various industries. He is an IBM Certified Developer Associate - IBM VisualAge for Smalltalk. He holds the equivalent of a masters degree in computer science from the Berufsakademie of Stuttgart, Germany. His areas of expertise include object-oriented analysis, design, and implementation using Smalltalk and Java.

Joseph Kwek is an IT Specialist with IBM Global Services, Singapore. In his three years with IBM, he has worked on the RS6000/AIX team implementing and supporting cross-platform systems solutions. He is currently focused on WebSphere services in ASEAN/SA. He holds a computer science degree from the National University of Singapore. In addition, he is a Microsoft Certified System Engineer, an IBM Certified Advanced Technical Expert for RS6000/AIX, and a Tivoli Certified Consultant for Tivoli Storage Manager.

Leo Marland is a Senior Consulting IT Architect in IBM Canada, providing architecture consulting and delivery services for e-business solutions to customers in the finance, insurance, government and healthcare industries. He has 19 years of experience with IBM including application development, software product development, and e-business services and consulting. He holds a doctorate in Theoretical Physics from Oxford University. His areas of expertise include e-business architecture and object technology including Java.

Witold Szczepoń is an IT Architect with IBM Global Services, Germany. He joined IBM in 1995 and worked as a developer, application designer and architect in the area of Telecommunications Management Network (TMN) and Web Application Architectures. His areas of expertise include object-oriented analysis, design, and implementation. He holds degrees from the University of Oklahoma, USA (Master of Science) and from the Technical University Braunschweig, Germany (Diplom Informatiker).

A special thanks to **Anthony Griffin** for his invaluable contribution in building the Pattern Development Kit.

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

Margaret Ticknor
International Technical Support Organization, Raleigh Center

George Galambos, Distinguished Engineer, e-business Architecture and Design Consulting
IBM Canada

Geoffrey Hambrick,
IBM Austin

Mike Bonett
IBM Gaithersburg

Michael Conner
IBM Austin

Michael Fraenkel
IBM Raleigh

Nataraj Nagarathnam
IBM Raleigh

Derek Ho
IBM Austin

Carmine Greco
IBM Raleigh

Srinivas Koushik, IBM Distinguished Engineer, Chief Architect US e-business Services
IBM USA

Comments welcome**Your comments are important to us!**

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

- Fax the evaluation form found in “IBM Redbooks review” on page 351 to the fax number shown on the form.
- Use the online evaluation form found at <http://www.redbooks.ibm.com/>
- Send your comments in an Internet note to redbook@us.ibm.com

Chapter 1. Introduction to patterns

We are all familiar with the pace of development of the computer industry during its relatively brief history. The rapid advances in computer hardware have been driven in no small part by the use of standards and well specified components for assembly. The desire to apply these same approaches to software construction gave rise to object-oriented software, design patterns and component-based development.

The idea of design patterns (see *Design Patterns - Elements of Reusable Object-Oriented Software*, by E. Gamma, R. Helm, R. Johnson, J. Vlissides) in software design and construction was inspired by the idea of using patterns in the design of buildings (see *A Pattern Language*, by C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel). The idea of design patterns has gained acceptance by software designers and developers because it enables an efficiency in both the communication and implementation of software design, based upon a common vocabulary and reference.

Buschmann, et al, authors of *Pattern-Oriented Software Architecture - A System of Patterns* identified patterns for system architecture at a higher level than the original design patterns. Their patterns are related to the macro-design of system components such as operating systems or network stacks.

Information technology architects, encouraged by the success of design patterns, and facing challenges in systematic and repeatable description of systems, have also explored the idea of architectural patterns.

The Enterprise Solution Structure (ESS) work (see "Enterprise Solutions Structure" in *IBM Systems Journal, Volume 38, No. 1, 1999* at <http://www.research.ibm.com/journal/sj38-1.html>) looked at patterns for complete end-to-end system architectures. ESS is now part of the IBM Global Services methodology.

1.1 Patterns for e-business

The Patterns for e-business aim to communicate in a highly accessible fashion the business pattern, systems architecture (application and runtime topologies), product mappings, and guidelines required for different classes of applications. For the User-to-Business patterns there is also an associated Pattern Development Kit, which provides sample application code to illustrate effective use of those patterns.

The patterns are cataloged according to the following business context scheme:

- User-to-business
- User-to-online buying
- Business-to-business
- User-to-data
- User-to-user
- Application integration

This redbook focuses on two of the application topologies for the User-to-Business pattern. For maximum benefit we recommend you use this redbook in conjunction with the IBM Pattern Development Kit and selected references, such as *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00, for more extended coverage of some topics.

The Pattern Development Kit was used in our lab during the development of this book as the running application code to validate the runtime topologies and product mappings.

In this book, we hope to give you an idea of all the important aspects of the Patterns for e-business, as applied to the User-to-Business patterns. These aspects are shown in Figure 1.

Patterns for e-business

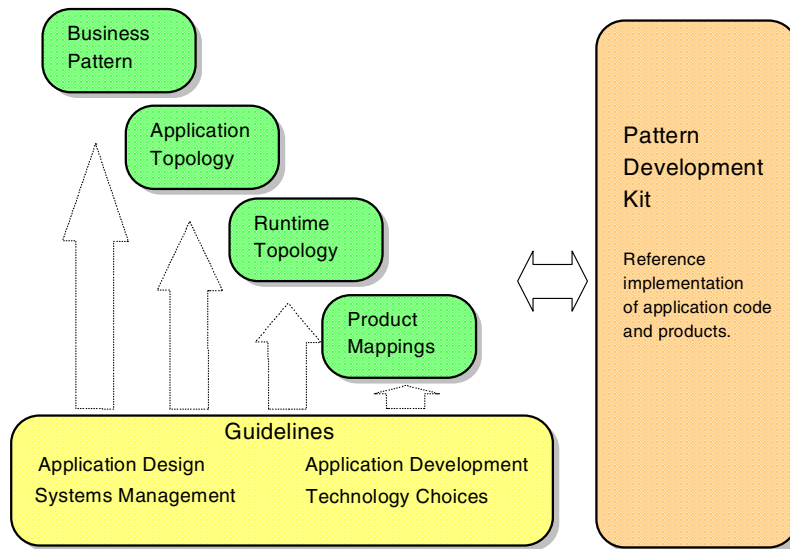


Figure 1. Patterns for e-business

1.2 How to use these patterns

The Patterns for e-business are particularly focused upon addressing common business application problems and providing answers to frequent architecture, design, and implementation questions.

We recommend that you can use the Patterns for e-business in a number of ways according to your needs:

- As a starting point for an end-to-end system architecture.
- As a detailed example and prescriptive approach, following the product mappings and guidance provided.
- As a way to design more complex, multi-channel systems, when several patterns are used together.

As with the design patterns and ESS work, we anticipate that architects and designers will want to combine these patterns to compose solutions to more complex system architectures. As the other Patterns for e-business are published we will identify the appropriate integration points for such composition.

We recommend that you use the Patterns for e-business together with an appropriate development methodology that considers the full set of requirements that are to be understood and implemented, whether these requirements concern the function of the solution or its operational characteristics such as availability, scalability, or performance.

1.3 Patterns for e-business Web site

The Patterns for e-business are published on IBM developerWorks, a portal for developers, and can be located at:

<http://www.ibm.com/software/developer/web/patterns>

This interactive patterns site acts as a guide to aid you in the selection of the pattern and topologies most relevant to your needs. While you can navigate via shortcuts to the information you most need, the site is structured to enable you to “drill down” into the material as you:

1. Select a business pattern.
2. Select an application topology.
3. Review runtime topologies.
4. Review product mappings.
5. Review guidelines.

At the time of writing, the Web site has material for the user-to-business and user-to-online buying patterns, with material for the other business patterns in the process of development.

You can also register at this site for pattern-related updates, which will include the Pattern Development Kit for user-to-business when it is available.

1.4 The Application Framework for e-business

The advent of e-business, with the requirement for interoperability that it brings, has been a major catalyst for the more rapid adoption of standards by the industry.

IBM's Application Framework for e-business establishes:

- A recommended approach for building systems, embodied in the Patterns for e-business.
- Innovative technology delivered in a rich product portfolio.
- Cross-platform standards, including Java and XML.

The Framework, with the standards it proscribes for e-business systems and their components, can be applied to:

- Custom application code
- Application packages
- Software products

The Patterns for e-business are an integral part of the IBM Application Framework for e-business. The Patterns make it easy to apply the technologies, standards, and products of the Application Framework to provide an e-business solution.

Figure 2 shows a pictorial summary of the major technology standards included in the Application Framework, with an indication of the areas where they are important.

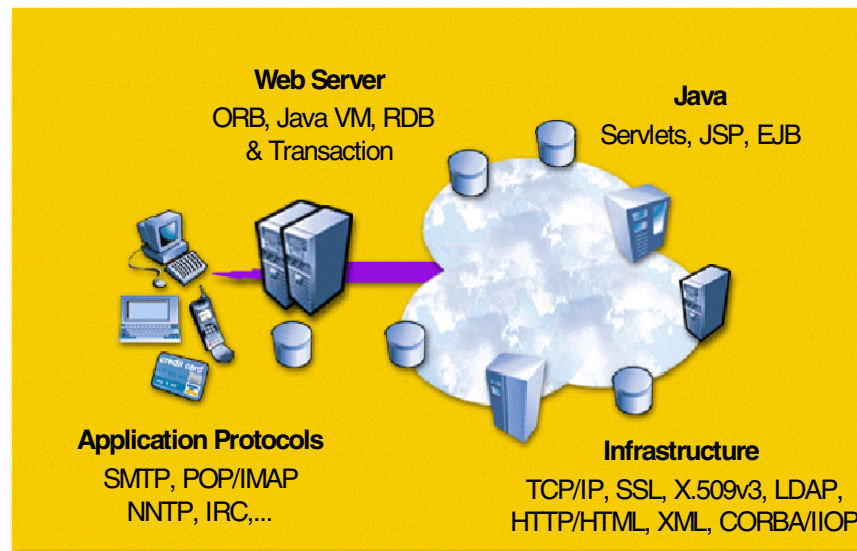


Figure 2. Technology standards and the Application Framework for e-business

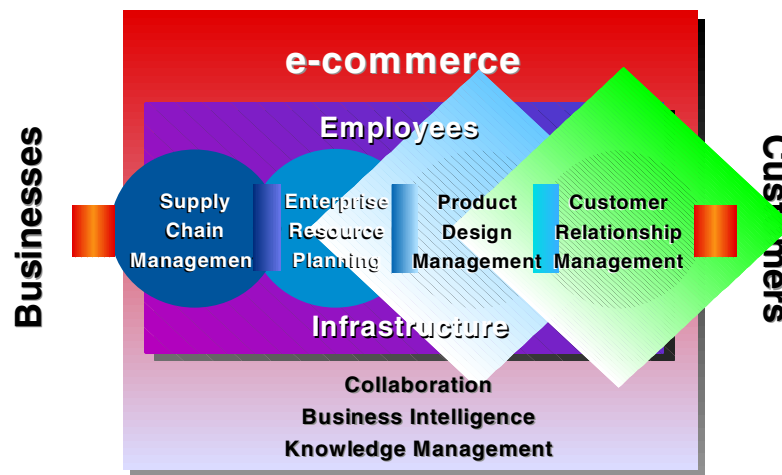
Framework white papers are an important source of information for the guidance material included in the Patterns for e-business. The Application Framework site at <http://www.ibm.com/software/ebusiness> includes a library section with this series of white papers.

1.5 An integrated view of e-business solutions

We mentioned earlier that a potential usage of the Patterns for e-business is to compose several patterns together for more complex system architectures.

As the following diagram shows, IBM views e-business as an integration of many application domains into systems that connect a business with its customers, partners, and suppliers.

e-business Requires Integration



These systems are not confined to Web interfaces, although increasingly many of the user interfaces to the combined system will use Web technology.

The common set of node descriptions in the Patterns for e-business enable communication between architects and designers from very different application domains and will suggest areas for shared nodes and infrastructure.

This is similar to the process of using design patterns to solve a programming design problem, where classes in the composed pattern play multiple roles, derived from the source patterns (see *Pattern Hatching - Design Patterns Applied* by J. Vlissides). It is different, however, in that design pattern composition is based on class diagrams and white box by nature, whereas composing architectural patterns is more component-based.

The Patterns for e-business may be applied to e-business solution areas. Here is a guide to where you may find them most applicable:

Table 1. Patterns for e-business and e-business solutions

e-business solution area	Business pattern
Customer relationship management	User-to-Business pattern
e-commerce	User-to-Online Buying pattern
Supply chain management	Business-to-Business pattern
Collaboration	User-to-User pattern
Business intelligence; knowledge management	User-to-Data pattern
Business application integration	Application Integration pattern

But for now, let us introduce you to the first set of patterns, those for user-to-business.

1.6 Structure of this redbook

Chapter 2, “Choosing the application topology” on page 11 introduces application topologies 1 and 2 for the User-to-Business pattern. With very accessible notation, these application topologies capture the essential “shape” of the application solution.

Chapter 3, “Choosing the runtime topology” on page 19 discusses the runtime topologies for these application topologies. It includes discussion of variations of these topologies that are appropriate for scalability and availability.

Chapter 4, “Product mapping” on page 43 provides sample product mappings to populate the logical runtime topologies. As with Chapter 3, the focus remains on the operational aspects of the solution.

In the chapters that follow, to make the material as valuable and relevant as possible, specific product mappings are referenced and the material is based on actual testing and analysis of the Pattern Development Kit application code running on the various runtime topologies.

Chapter 5, “Performance guidelines” on page 55 introduces performance guidelines by considering the components of a user-to-business solution that are particularly relevant to performance.

Chapter 6, “Technology options” on page 65 discusses the technology options available to implement a Web application and advises on appropriate usage.

Chapter 7, “Application design guidelines” on page 81 introduces consideration of the functional components of the application within the context of the runtime topologies.

Chapter 8, “Application development guidelines” on page 147 provides guidelines for application development, considering the roles, processes, and tools that are required.

Chapter 9, “System management products and guidelines” on page 197 looks at the asset management, security, and availability aspects of an e-business application.

Part 1. User-to-Business patterns: topologies 1 and 2

Chapter 2. Choosing the application topology

Once the business pattern is chosen, it is time to choose an application topology. The application topologies use logical nodes to illustrate various ways to configure the interaction between users, applications, and data. The chosen application topology is later mapped to a runtime topology by mapping the logical nodes to runtime nodes.

An application topology shows the principal layout of the application, focusing on the shape of the application, the application logic, and the associated data. It does not show middleware, or the files or databases where Web pages may be stored. The application design is also not described in the application topology. For information about application design refer to Chapter 7, “Application design guidelines” on page 81.

There are two different types of application topologies:

- Web-up, where the premise is to build a Web application from scratch.
- Enterprise-out, where the idea is to Web-enable an enterprise application, that is, builds an additional Web channel to access an enterprise application.

This book will concentrate on two Web-up specific application topologies for the User-to-Business pattern:

- Topology 1, for use when no legacy or third-party applications are required
- Topology 2, for use when the access to legacy and/or third-party applications is required

By using the information for each application topology:

- Business drivers
- Key features
- Considerations

you should be able to choose the application topology that best fits your requirements with regard to users, applications, and data.

2.1 Application topology 1

User-to-business application topology 1 is applicable in a situation where you are building a new application and there is no need to interface with legacy or third-party applications or data. All the data required is handled by the new

e-business application. It does not need to share or exchange data with other applications, nor does it need to access data from another application.

The application implements all required functions; it does not rely on functionality provided by other existing applications.

2.1.1 Application topology 1: business driver

This topology describes the situation where the customer is planning a new application or is extending a current Web publishing capability, with an e-commerce capability and no back-end integration (a classic Web-up strategy). For example, it allows businesses to provide customers with read-only access to marketing and sales literature via the Web. Furthermore, customers can make business transactions that update a database within the new application.

This topology is sufficient for companies that do not have any legacy or third-party systems to integrate with, or at least do not have that need at this point in time. This topology can be easily extended to topology 2 in the future to include access to existing applications such as inventory management, or third-party applications such as credit checks. See 2.2, “Application topology 2” on page 14 for more information.

2.1.2 Application topology 1: key features

Application topology 1 represents the target topology for most application server vendors. It allows developers to replace the monolithic fat client design with a layered approach. This architecture uses a thin client with application business logic on the second tier. The second tier can access a local database maintaining the application data. It aims to address the scalability problems of client/server and at the same time provide reuse of the business logic and data by all styles of Web browsers.

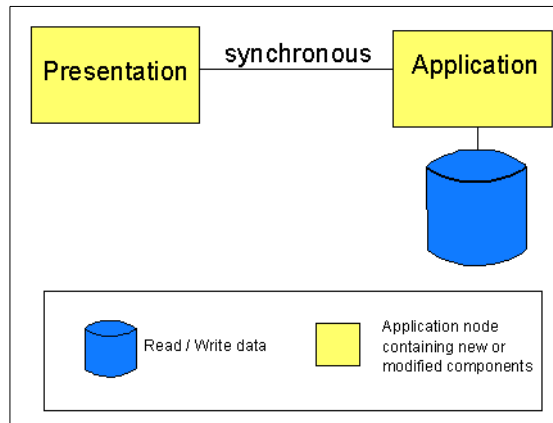


Figure 3. Application topology 1

Application topology 1 has a simple application and data layout. The application is divided into two different logical application nodes to accomplish a separation of the presentation logic from the business logic, making it a logical 2-layer architecture.

The presentation node is responsible for all presentation logic of the application. The application node is responsible for all business logic and data access of the application.

The communication between the presentation logic and business logic layer is synchronous, meaning that any request coming from the user interface directly invokes business logic on the application node. After the execution of the business logic, control is passed back to the presentation node that uses the results to update the user interface.

In this topology, all data is held in a read/write storage directly accessible by the application node, usually a local database.

There is only a small amount of presentation logic (for example, JavaScript) running on the client, which is typically a thin browser client. Most of the presentation logic and the entire business logic runs on a server. To scale the application, the power of the server machine can be increased or the application can be spread across many servers. These options provide the ability to increase the application's overall scalability, especially when the number of users is growing, without having to change the logical application topology.

Since the presentation logic and business logic are separated, it is easy to adapt the presentation node to new kinds of clients. The easiest approach is to use thin browser-based clients. But it is also possible to extend the presentation logic to new client platforms, for example, client Java applications or Web appliances like Web-enabled cellular phones or Personal Digital Assistants (PDAs), without the need to change the business logic in the application node.

2.1.3 Application topology 1: considerations

A Web application server may implement both tiers of the layered design, but developers should exercise caution. Many vendors promote ease of development by mixing scripting and components, paying little attention to engineering the application with separate presentation and business logic layers. This combined approach should be avoided as separation promotes the effective use of discrete skill sets and code reuse. Failing to separate the layers can lead to code that is hard to maintain and extend. Also, be aware that you may incur significant departmental system management costs when the business logic and data are held outside the IT organization.

2.2 Application topology 2

Application topology 2 for the User-to-Business pattern applies to scenarios where there is the need for integration with legacy or third-party applications. The new application is not built as a stand alone solution, but instead has to integrate with existing applications. The integration can be achieved on a functional basis or a data basis, that is, using a transactional interface or a data interface.

When trying to integrate with an existing application, it is important to find out whether this application has to be changed, and if such a modification is allowed. If the new application is going to use existing data you must determine if the data can be used “as is” or if the structure needs to be changed, and if so, is this modification possible. Notice that in most cases the objective is to leave the existing applications and data unchanged.

2.2.1 Application topology 2: business driver

Application topology 2 allows for one or more point-to-point connections to back-end legacy applications or databases. This is a very common requirement for businesses delivering goods and services over the Web. For example, an e-commerce application can be integrated with existing back-end applications such as inventory management.

Often this topology is used to extend existing application topology 1 solutions to integrate with legacy or third-party systems, for example, inventory management or credit card checking.

2.2.2 Application topology 2: key features

Application topology 2 for user-to-business is an extension of application topology 1. It allows the second tier's new application business logic to access existing applications or data, such as inventory management, or third-party applications, such as credit checks. These applications reside on a third tier elsewhere in the network.

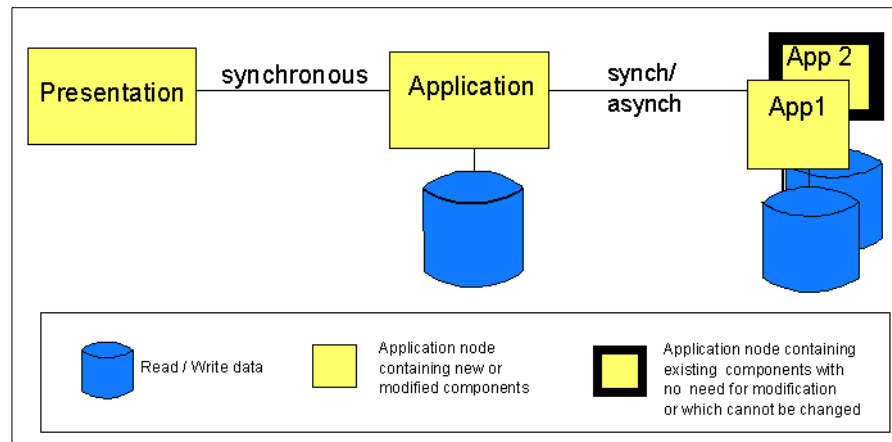


Figure 4. Application topology 2

Application topology 2 has the same logical application nodes known from application topology 1 and at least one additional node, making it a logical three-tier architecture. Depending on the requirements, this additional logical layer contains new, modified, or unmodified components and resides in the third tier.

As in topology 1, the presentation node is responsible for all presentation logic of the application. The application node resides in the second tier, and is responsible for the new business logic and integration with the existing back-end applications. The third tier node might also provide business logic, or may only be used to access legacy or third-party data.

The communication between the first and second tier is synchronous, as described in 2.1.2, "Application topology 1: key features" on page 12.

The connection between the new application node and existing or modified nodes can be asynchronous or synchronous. The communication type depends on the communication characteristics and capabilities of the back-end system. When there is an existing batch system to integrate, the communication has to be asynchronous, whereas an existing database system will be called synchronously.

The data can reside in the second and/or the third logical tier. Usually new data, such as user data or profiling information, will be put in the second tier and is directly accessible by the new application node. The data of existing systems is kept in the third tier, and is most likely only accessible through the existing back-end systems. It may be necessary to build new, or to modify existing, components on the back-end to give the new application proper access to existing data. In some cases it might even be possible to directly access legacy data from the second tier, for example, accessing an existing database directly from the second tier application.

2.2.3 Application topology 2: considerations

The same considerations mentioned in 2.1.3, “Application topology 1: considerations” on page 14 apply to application topology 2. Additionally the following concerns have to be taken into account.

If the developer needs to provide access to many applications on the third tier, an advanced application topology may be more appropriate. You should consider an application topology that exploits a hub-and-spoke architecture between the second and third tiers.

You should consider how this topology will be deployed to avoid systems management complexity. Complexity arises when updated corporate data resides on more than one tier with the second and third tiers both within the same organization but physically distributed. For example, synchronization of backups may be cumbersome.

If there are different IT organizations developing the new application and maintaining or changing the existing systems, the development might be difficult to coordinate, especially if the interfaces between the new and the existing systems are not properly defined and documented.

As stated before, the new application might require changes to existing production systems. This is always a critical task, especially when the back-end systems or third-party applications are mission critical. Building a test system for the existing applications that need to be changed is a good way to avoid production system failure, while developing and testing the new

e-business application. Another problem might be in having skilled resources that can change the third tier application. Often these are quite old applications and finding someone who understands them well enough to change them may be difficult.

Chapter 3. Choosing the runtime topology

Application topologies 1 and 2 represent a starting point for delivering e-business applications. In the simplest case, application topology 1, there is no interaction with legacy back-end systems. In the more complicated case, application topology 2, there are connections to legacy back-end systems.

Once the application topology has been chosen, it is time to choose the runtime topology. A runtime topology uses nodes to group functional and operational components. The nodes are interconnected to solve a business problem. An application topology leads to an underlying runtime topology.

This chapter will discuss two runtime topologies corresponding to the application topologies developed in Chapter 1, “Introduction to patterns” on page 1, and variations of each. These runtime topologies are based on the Enterprise Solution Structure (ESS) Thin Client Transactional pattern and are a representative solution for the User-to-Business pattern. For more information about the Enterprise Solution Structure (ESS), see “Enterprise Solutions Structure” in *IBM Systems Journal, Volume 38, No. 1, 1999* at <http://www.research.ibm.com/journal/sj38-1.html>.

Each topology has four variations:

- Proven basic topology
- Proven variation 1
- Emerging variation 2
- Emerging multi-tier variation 3

A variation that is labeled “proven” means that it is based on technology that has been around a while and has been the chosen method in many production systems.

An “emerging” variation is one that is based on newer technology that may or may not have been proven in production environments, but has significant benefits and is worth considering.

You may find that, depending on the customer requirements, you will need to extend the variations or combine them to get the desired results.

Most references to vertical scalability in this chapter assume upgrading the power, number or memory capacity of the processors for a given platform. Please bear in mind that one of the benefits of the Application Framework for e-business is that vertical scalability to a big iron solution is also possible by

migrating the application to another Application Framework for e-business supported platform, such as OS/390, OS/400, or RS/6000 SP.

3.1 An introduction to the node types

The runtime topologies will be shown in graphical form in the following sections. Each topology will consist of several nodes, describing the function represented on that node. Most topologies will consist of a core set of common nodes, with the addition of one or more nodes unique to that topology. To understand the runtime topologies, you will need to review the following node definitions.

3.1.1 Web application server

A Web application server node is an application server that includes an HTTP server (also known as a Web server) and is typically designed for access by HTTP clients and to host both presentation and business logic.

The Web application server node is a functional extension of the informational (publishing-based) Web server. It provides the technology platform and contains the components to support access to both public and user-specific information by users employing Web browser technology. For the latter, the node provides robust services to allow users to communicate with shared applications and databases. In this way it acts as an interface to business functions, such as banking, lending, and HR systems.

This node would be provided by the company, on company premises, or hosted inside the enterprise network and inside a Demilitarized Zone (DMZ) for security reasons. In most cases, access to this server would be in secure mode, using services such as SSL or IPSEC.

In the simplest design, this node can provide the management of hypermedia documents and diverse application functions. For more complex applications or those demanding stronger security, it is recommended that the application be deployed on a separate Web application server node inside the internal network.

Data that may be contained on the node includes:

- HTML text pages, images, multimedia content to be downloaded to the client browser
- Java Server Pages
- Application program libraries, for example, Java applets for dynamic downloading to client workstations

3.1.2 Public Key Infrastructure (PKI)

PKI is a collection of standards-based technologies and commercial services to support the secure interaction of two unrelated entities (for example, a public user and a corporation) over the Internet. In the context of the topologies defined in this redbook, PKI supports the authentication of the server to the browser client, using the SSL protocol.

3.1.3 Domain Name Service (DNS) node

The DNS node assists in determining the physical network address associated with the symbolic address (URL) of the requested information. The DNS is that of the Internet service provider, although DNS is implemented on the accessed site, too.

3.1.4 User node

This node is most frequently a personal computing device (PC, etc.) supporting a commercial browser, for example, Netscape Navigator or Internet Explorer. The level of the browser is expected to support SSL and some level of DHTML. Increasingly, designers should also consider that this node may be a pervasive computing device, such as a Personal Digital Assistant (PDA).

3.1.5 Directory and security services node

This node supplies information on the location, capabilities and various attributes (including user ID/password pairs and certificates) of resources and users, known to this Web application system. The node may supply information for various security services (authentication and authorization) and may also perform the actual security processing, for example, to verify certificates. The authentication in most current designs validates the access to the Web application server part of the Web server, but it can also authenticate for access to the database server.

3.1.6 Database server node

This node's function is to provide a persistent data storage and retrieval service in support of the user-to-business transactional interaction. The data stored is relevant to the specific business interaction, for example, bank balance, insurance information, current purchase by the user, etc.

It is important to note that the mode of database access is perhaps the most important factor determining the performance of this Web application, in all but the simplest cases. The recommended approach is to collapse the

database accesses into a single call or very few calls. This can be achieved via coding and invoking stored procedure calls on the database.

3.1.7 Protocol firewall and domain firewall nodes

Firewalls provide services that can be used to control access from a less trusted network to a more trusted network. Traditional implementations of firewall services include:

- Screening routers (the protocol firewall in this design)
- Application gateways (the domain firewall)

The two firewall nodes provide increasing levels of protection at the expense of increasing computing resource requirements. The protocol firewall is typically implemented as an IP router, while the domain firewall is a dedicated server node.

3.1.8 Load balancer node

The load balancer provides horizontal scalability by dispatching HTTP requests among several, identically configured Web servers.

3.1.9 Shared file system node

The timely synchronization of several Web servers is achieved by using a shared file system as the content storage and capitalizing on the replication capability of this technology.

3.1.10 Web server redirector node

In order to separate the Web server from the application server, a so-called *Web server redirector node* (or just redirector for short) is introduced. The Web server redirector is used in conjunction with a Web server. The Web server serves HTTP pages and the redirector forwards servlet and JSP requests to the application servers. The advantage of using a redirector is that you can move the application server behind the domain firewall into the secure network, where it is more protected than within the DMZ. Static pages can be served from the DMZ by this node.

The redirector can be implemented, for example, by either a reverse proxy server or by a Web server plug-in such as the servlet redirector function of IBM WebSphere Application Server Advanced Edition. More information on implementing a redirector can be found in 4.1.1, “Implementing a redirector” on page 43.

3.1.11 Application server node

This node provides the infrastructure for application logic and may be part of a Web application server. It is capable of running both presentation and business logic but generally does not serve HTTP requests. When used with a Web server redirector the application server node will run both presentation and business logic. In other situations, it may be used for business logic only.

3.1.12 Existing applications and data node

Existing applications are run and maintained on nodes that are installed in the internal network. These applications provide for business logic that uses data maintained in the internal network. The number and topology of these existing application and data nodes is dependent on the particular configuration used by these legacy systems.

Where do you go from here?

In Chapter 2, “Choosing the application topology” on page 11, you were assisted in choosing the appropriate application topology for your environment.

If you chose application topology 1, continue here with 3.2, “Runtime topology A” on page 23.

If you chose application topology 2, skip the next section and go directly to 3.3, “Runtime topology B” on page 31.

3.2 Runtime topology A

Runtime topology A implements application topology 1. This topology consists of a basic topology and three variations. The basic topology and the first variation are considered to be “proven” topologies. Variations 2 and 3 are considered to be “emerging” topologies.

3.2.1 Proven basic topology

This runtime topology provides an initial implementation with an entry-level footprint. Or, to put it in simple words: “Start simple, grow fast.”

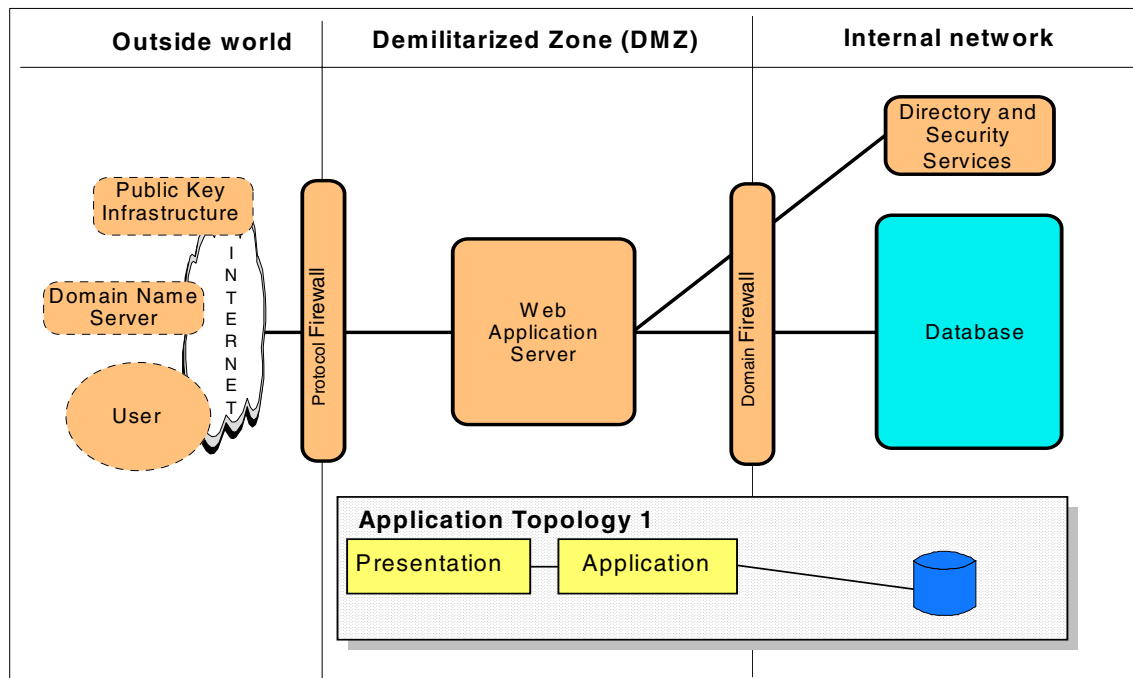


Figure 5. Runtime topology A (proven basic topology)

In this basic runtime topology, there is a single Web application server (a Web server and application server combined) residing in a Demilitarized Zone (DMZ).

The presentation logic and business logic are implemented on the Web application server. The data to be accessed from the business logic is behind the domain firewall in the internal network.

Access to the application server's resources is protected by the application server's security features. User information, needed for authentication and authorization, is stored in the directory and security services node behind the domain firewall in the internal network.

Benefits

This runtime topology offers the following benefits:

- This runtime topology can be closely modeled on a single developer workstation.
- All sensitive persistent data is stored behind the DMZ.

Limitations

This topology, in its pure form, has the following limitations:

- The topology has limited availability and failover capability.
- Horizontal scalability is not possible because there is no means of allowing more than one Web application server. See 3.2.2, “Proven variation 1” on page 25 for a solution.
- Only a limited amount of vertical scalability is possible. Vertical scalability can be achieved by adding memory or processors, and/or creating duplicates or clones of applications on the Web application server. See 4.1.2, “Clones running on application servers” on page 45 for more details.
- The number of clients that access the Web server simultaneously is limited by the capacity of the Web server. The actual numbers depend on the software and hardware platform used.
- Since the Web server is not separated from the application server, there is no additional security available and the business logic is protected only by the protocol firewall.

3.2.2 Proven variation 1

The number of Web servers is crucial when it comes to serving high volume sites. This is because each Web server can serve only a limited number of clients at the same time. As soon as this limit is reached, additional users who want to access that site will receive error messages that the site cannot be reached. But this information is incorrect because the Web server is connected. It just cannot serve the requests.

To overcome this problem you can install additional Web servers, which will then allow for a larger amount of Web traffic. Each of these Web servers will be connected to the corresponding application server.

This topology provides two approaches to horizontal scaling:

- Model/cloning - also supports process failover
- Load balancer - also supports machine failover

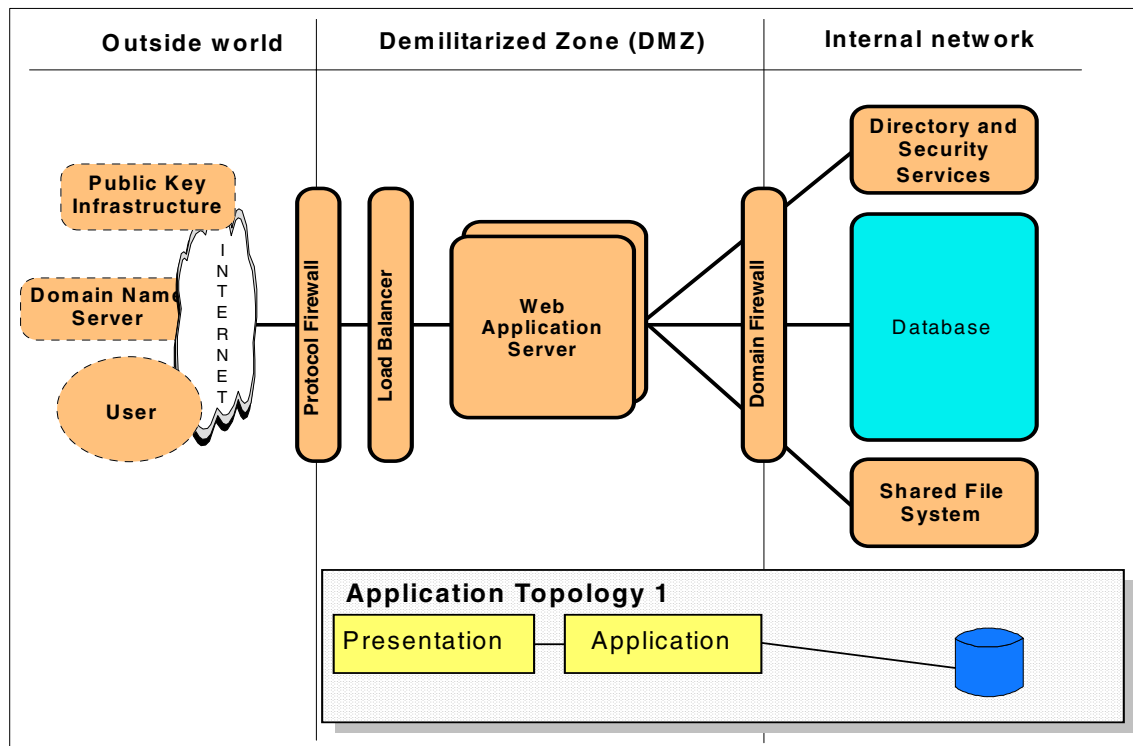


Figure 6. Runtime topology A (proven variation 1)

This variation allows a larger number of clients to access the Web site. The fundamental difference between this variation and the basic runtime topology is that the number of Web application servers is increased. A load balancer node is used to distribute (spray) the incoming requests to the Web application servers.

A shared file system may be installed in the internal network. This file system provides for shared access to information needed by all Web application servers.

The presentation logic and business logic are implemented on the Web application server. The data to be accessed from the business logic is behind the domain firewall in the internal network.

Access to the application server's resources is protected by the application server's security features. User information, needed for authentication and authorization, is stored in the directory and security services node behind the domain firewall in the internal network.

It is important that the application code on the Web application server be stateless, or that persistent sessions are shared among the pool of Web application servers. More information on session states can be found in 4.1.3, “Session sharing across servers” on page 46 and 4.1.4, “Achieving HTTP session affinity” on page 47.

Since there is more than one Web application server serving the clients’ requests, special care needs to be taken whenever scarce resources are accessed. An example of such a resource is a connection to the database. Since only a limited number of open connections can exist, connection pooling may be a proper way to address the problem.

A different approach to achieving horizontal scalability can be found in Part 3, “Application topology 1: a working example” on page 229, where workload management is done by cloning of applications.

Benefits

This runtime topology offers the following benefits:

- The topology has a high availability and failover capability.
- Horizontal scalability of the business logic is achieved by adding additional Web application servers to the topology; vertical scalability can be applied wherever possible or needed.

Limitations

This topology, in its pure form, has the following limitations:

- Since the Web servers are not separated from the application servers, there is no additional security available and the business logic is protected only by the protocol firewall.
- There can be drawbacks to increasing the number of Web application servers if other resources cannot handle the load. The throughput will not increase if the back-end systems have reached their limits. Overall, it is highly application dependent as to whether a performance improvement can be achieved by introducing additional application servers.

3.2.3 Emerging variation 2

Under certain circumstances, you may wish to separate the Web server from the Web application server. This may be necessary when you want to put the application server behind a DMZ so it is in a secure environment. The corresponding Web server can be placed inside the DMZ behind a firewall.

This separation of the Web server from the application server is done by means of the so-called *Web server redirector* (or redirector for short). The

Web server serves HTTP pages and the redirector forwards servlet and JSP requests to a dedicated server.

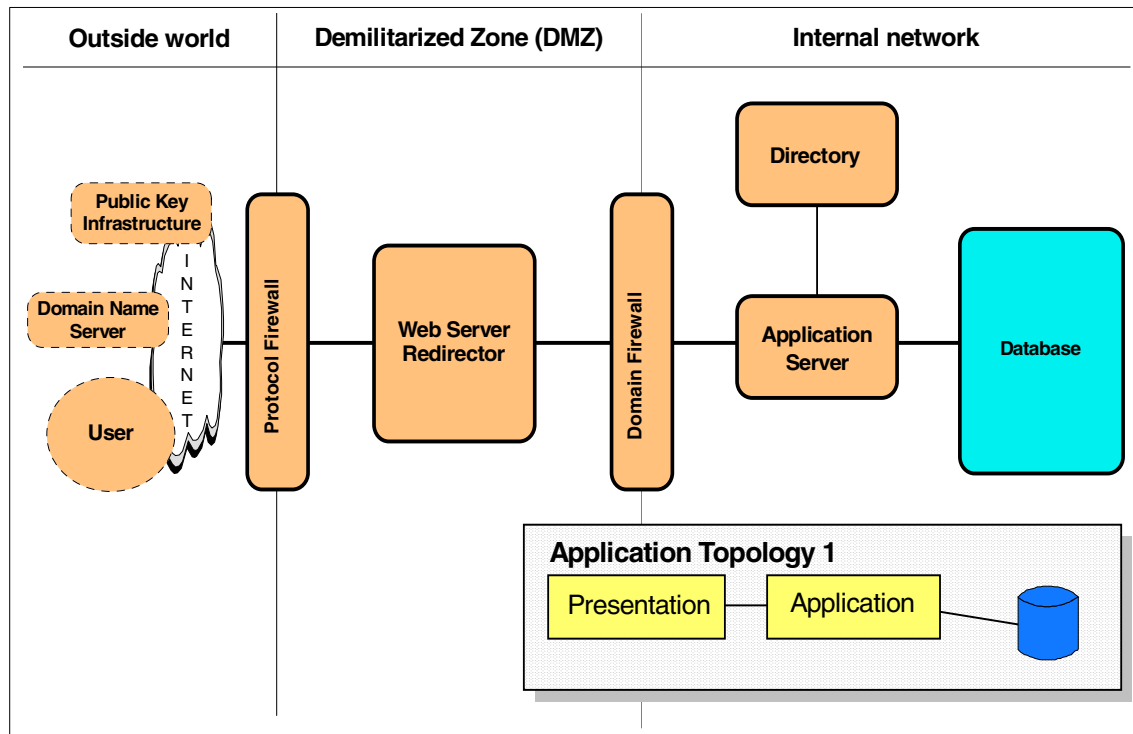


Figure 7. Runtime topology A (emerging variation 2)

This variation to the basic topology uses one Web server and one application server, effectively splitting the function of a Web application server across two machines. In this case the application server resides in the internal network to provide it with more security. The application server node will run both presentation and business logic.

The Web server remains in the DMZ and serves static pages. A Web server redirector is used to forward the requests from the Web server to the application server.

Access to the application server's resources is protected by the application server's security features. User authentication is implemented by the directory and security services node.

This topology is especially useful when you do not expect many clients to access the server simultaneously and when you want to have additional security by separating the Web server from the application server.

Benefits

Since the Web server is separated from the application server, additional security is available and the business logic is protected by both the protocol and the domain firewall.

Limitations

This topology has the following limitations:

- The topology has limited availability and failover capability.
- Horizontal scalability is not possible because there are no means to allow for more than one Web server or Web application server. See 3.2.2, “Proven variation 1” on page 25 for a solution.
- Only a limited amount of vertical scalability is possible. Vertical scalability can be achieved by adding memory or processors, creating duplicates or clones of applications on the application server. See 4.1.2, “Clones running on application servers” on page 45 for more details.
- Since only one Web server is used, only a limited number of clients can be served simultaneously. This is because each Web server has to allocate limited resources, such as memory, sockets, processes, or threads, etc., whenever it serves an incoming request.
- Since the requests to the application server need to be forwarded, you could see a noticeable performance degradation, depending on the redirector solution chosen. A reverse proxy is an alternative way to implement a redirector and has little performance degradation.
- In addition, a firewall needs to be configured to allow traffic from the redirector to the application server. But it is the firewall that delivers additional security.

3.2.4 Emerging multi-tier variation 3

Depending on the nature of the application logic and the customer’s security policy, you may need to locate the business logic behind the DMZ while maintaining the presentation logic on a Web application server in the DMZ.

Also, in some situations, you may want to design an application server that can be accessed by thick clients, perhaps using a protocol other than HTTP, and with a richer presentation logic at the client. Or you may have to integrate an existing application server that has been designed to be accessed with a protocol other than HTTP in the process of defining an architecture.

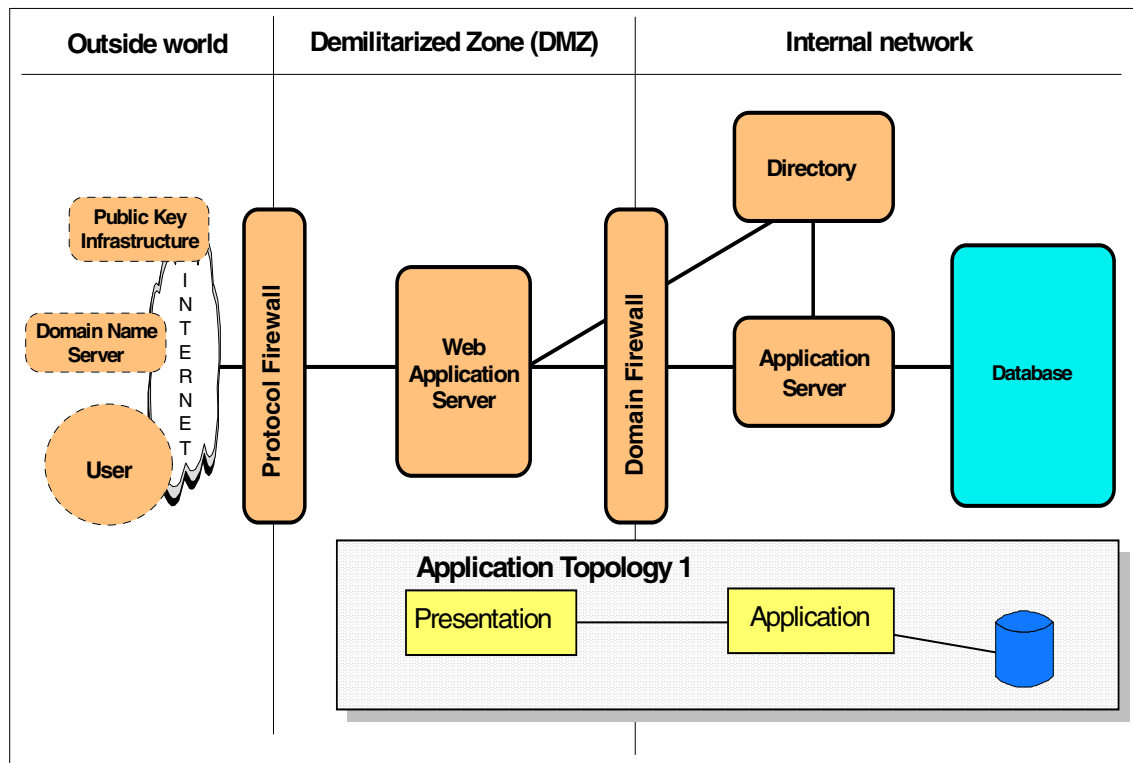


Figure 8. Runtime topology A (emerging multi-tier variation 3)

This variation of the basic topology adds an additional application server behind the DMZ to provide more security. A single Web application server within the DMZ is used for presentation logic, but all business logic is implemented within the application server behind the DMZ. This provides a clear layering between the presentation logic and the business logic.

This topology does not specify the choice of protocol between the Web application server and the application server. Depending on the application server it may be HTTP, RMI, or RMI/IIOP.

In this case the application server will be used for running business logic only. Which application server technology to use is an important implementation decision. Some candidates to consider are an EJB server, a traditional transaction monitor, or an RMI server. Using an RMI server is not recommended because session beans are a better choice. But in some situations, an RMI server may already be present.

Access to the application server's resources is protected by the application server's security features. User authentication is implemented by the directory and security services node.

Benefits

Since the Web application server that implements the presentation logic is separated from the application server that implements the business logic, additional security is available and the business logic is protected by both the protocol and the domain firewall.

Limitations

This topology can have the following limitations:

- Horizontal scalability is not possible because there are no means to allow for more than one Web application server. See 3.2.2, "Proven variation 1" on page 25 for a solution which, combined with this runtime topology, provides for optimal scalability.
- Only a limited amount of vertical scalability is possible. Vertical scalability can be achieved by adding memory or processors, and/or creating duplicates or clones of applications on the Web application server. See 4.1.2, "Clones running on application servers" on page 45 for more details.
- Since only one Web server is used, only a limited number of clients can be served simultaneously. This is because each Web server has to allocate limited resources, such as memory, sockets, processes, or threads, etc., whenever it serves an incoming request.
- Since the requests to the actual application server need to be forwarded, there is a slight performance degradation to be expected.

Where do you go from here?

The next section discusses runtime topology B, which applies to application topology 2. The only difference between runtime topology A and B is in the back-end connections, which are not covered in any significance here. Therefore, we recommend that you skip directly to 3.4, "Intranet vs. Internet runtime topologies" on page 40.

3.3 Runtime topology B

This runtime topology supports application topology 2. It has much in common with topology A. The difference is in the back-end applications.

Topology A assumes that no legacy or third-party data applications are required. Topology B provides for connections to existing applications.

This topology consists of a basic topology and three variations. The basic topology and the first variation are considered to be “proven” topologies. Variations 2 and 3 are considered to be “emerging” topologies.

3.3.1 Proven basic topology

The motivation for this runtime topology is to provide an extension of runtime topology A to integrate legacy or third-party systems.

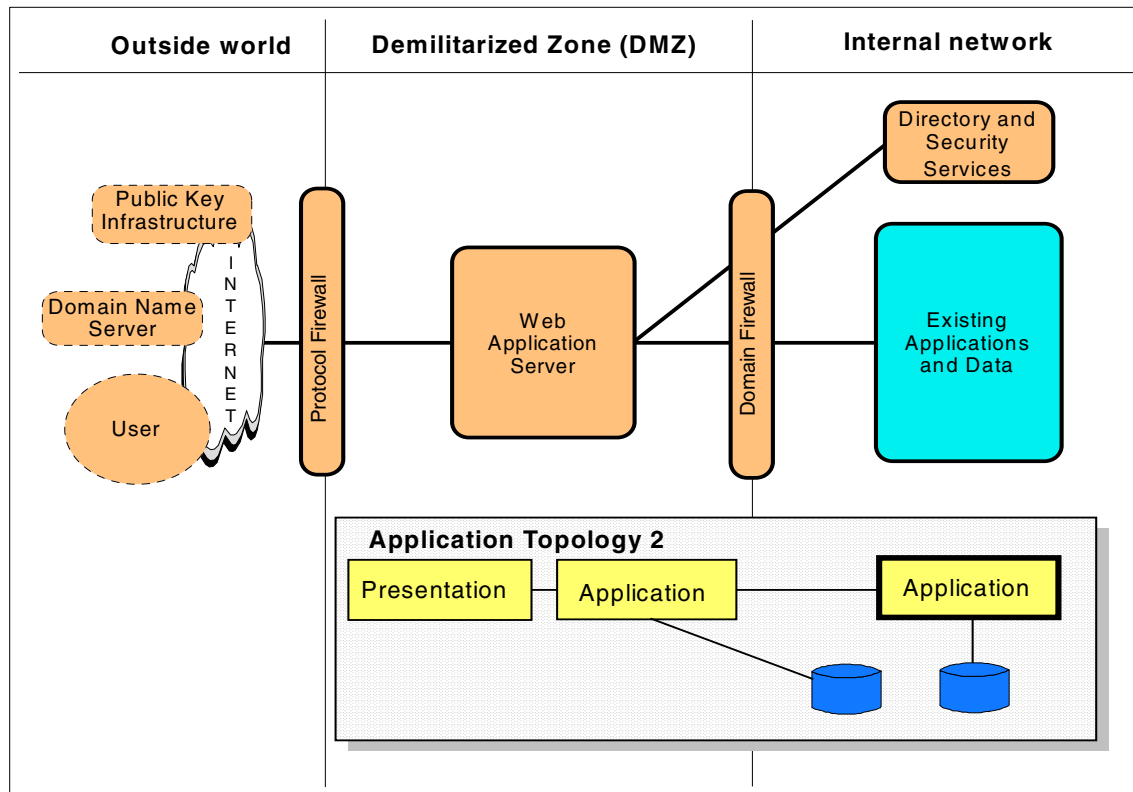


Figure 9. Runtime topology B (proven basic topology)

In this basic runtime topology, there is a single Web application server, a Web server and application server combined, residing in the Demilitarized Zone (DMZ).

Existing business logic in the internal network is augmented by business logic on the Web application server. The existing applications and data to be accessed from the business logic are behind the domain firewall in the internal network.

Access to the application server's resources is protected by the application server's security features. User information, needed for authentication and authorization, is stored in the directory and security services node behind the domain firewall in the internal network.

Benefits

This topology offers the following benefits:

- This runtime topology can be closely modeled on a single developer workstation.
- All sensitive persistent data is stored behind the DMZ.

Limitations

This topology can have the following limitations:

- The topology has limited availability and failover capability.
- Horizontal scalability is not possible because there is no means of allowing more than one Web application server. See 3.3.2, "Proven variation 1" on page 33 for a solution.
- Using workstations, only a limited amount of vertical scalability is possible. Vertical scalability can be achieved by adding memory or processors, and/or creating duplicates or clones of applications on the Web application server. See 4.1.2, "Clones running on application servers" on page 45 for more details. Using an inherently scalable hardware architecture, such as IBM's SP or OS/390, vertical scalability is not an issue.
- The number of clients that access the Web server simultaneously is limited by the capacity of the Web server. The actual numbers depend on the software and hardware platform used.
- Since the Web server is not separated from the application server, there is no additional security available and the business logic on the Web application server is protected only by the protocol firewall.

3.3.2 Proven variation 1

The number of Web servers is crucial when it comes to serving high volume sites. This is because each Web server can serve only a limited number of clients at the same time. As soon as this limit is reached, additional users who want to access that site will receive error messages that the site cannot

be reached. But this information is incorrect because the Web server is connected. It just cannot serve the requests.

To overcome this problem you can install additional Web servers, which will then allow for a larger amount of Web traffic. Each of these Web servers will be connected to the corresponding application server.

This topology provides two approaches to horizontal scaling:

- Model/cloning - also supports process failover
- Load balancer - also supports machine failover

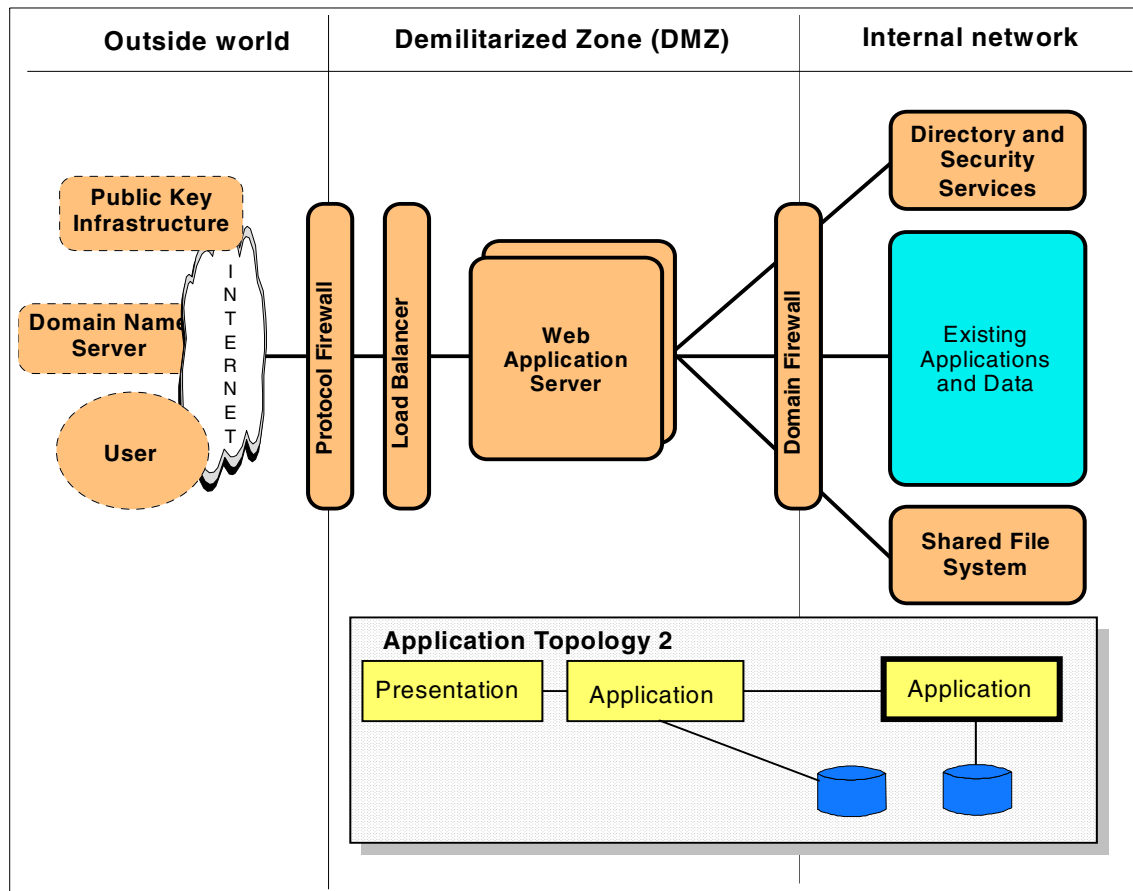


Figure 10. Runtime topology B (proven variation 1)

This variation allows a larger number of clients to access the Web site. The fundamental difference between this variation and the basic runtime topology is that the number of Web application servers is increased. A load balancer node is used to distribute (spray) the incoming requests to the Web application servers.

A shared file system may be installed in the internal network. This file system provides for shared access to information needed by all Web application servers.

Existing business logic in the internal network is augmented by business logic on the Web application server. The existing applications and data to be accessed from the business logic are behind the domain firewall in the internal network.

Access to the application server's resources is protected by the application server's security features. User information, needed for authentication and authorization, is stored in the directory and security services node behind the domain firewall in the internal network.

It is important that the application code on the Web application server be stateless, or that persistent sessions are shared among the pool of Web application servers. More information on session states can be found in 4.1.3, "Session sharing across servers" on page 46 and 4.1.4, "Achieving HTTP session affinity" on page 47.

Since there is more than one Web application server serving the clients' requests, special care needs to be taken whenever scarce resources are accessed. An example of such a resource is a connection to the database. Since only a limited number of open connections can exist, connection pooling may be a proper way to address the problem.

A different approach to achieving horizontal scalability can be found in Part 3, "Application topology 1: a working example" on page 229, where workload management is done by cloning of applications.

Benefits

This topology offers the following benefits:

- The topology has a high availability and failover capability.
- Horizontal scalability of the business logic is achieved by adding additional Web application servers to the topology; vertical scalability can be applied wherever possible or needed.

Limitations

This topology has the following limitations:

- Since the Web servers are not separated from the application servers, there is no additional security available and the business logic is protected only by the protocol firewall.
- There can be drawbacks to increasing the number of Web application servers if other resources cannot handle the load. The throughput will not increase if the back-end systems have reached their limits. Overall, it is highly application dependent as to whether a performance improvement can be achieved by introducing additional application servers.

3.3.3 Emerging variation 2

Under certain circumstances, you may wish to separate the Web server from the Web application server. This may be necessary when you want to put the application server behind a DMZ so it is in a secure environment. The corresponding Web server can be placed inside the DMZ behind a firewall.

This separation of the Web server from the application server is done by means of the so-called *Web server redirector* (or redirector for short). The Web server serves HTTP pages and the redirector forwards servlet and JSP requests to a dedicated server.

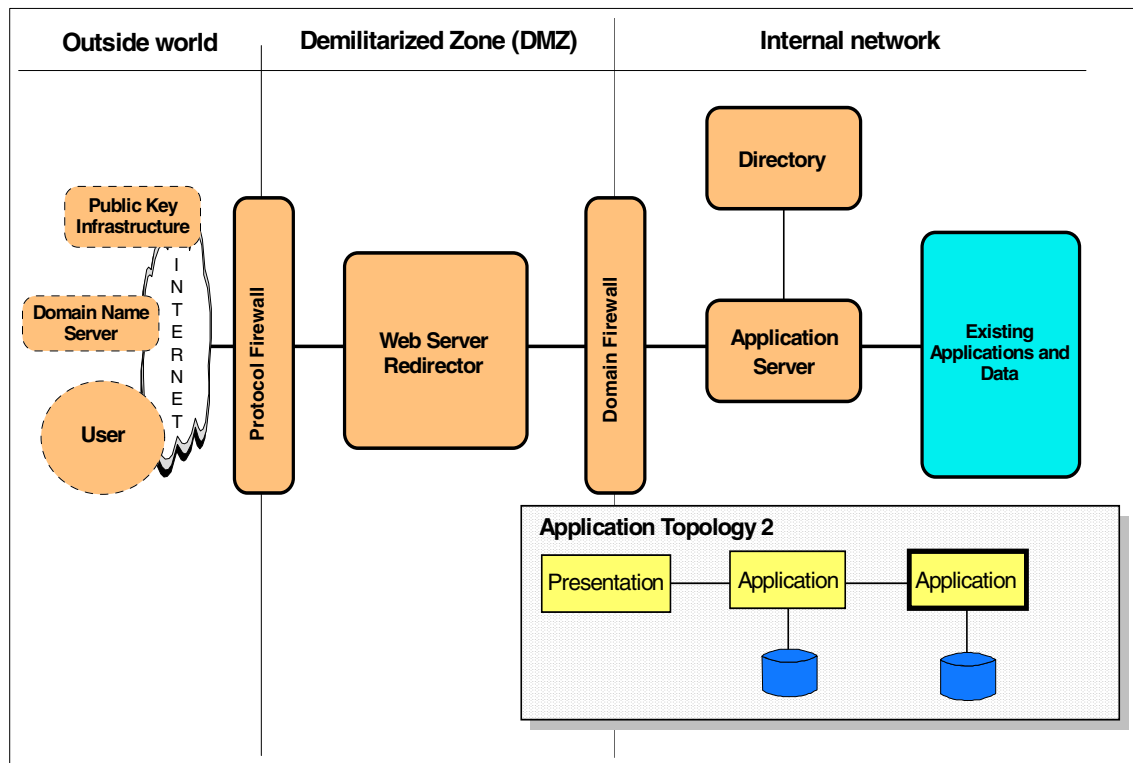


Figure 11. Runtime topology B (emerging variation 2)

This variation to the basic topology uses one Web server and one application server, effectively splitting the function of a Web application server across two machines. In this case the application server resides in the internal network to provide it with more security. The application server will run both presentation and business logic. There is also business logic in the existing applications.

The Web server remains in the DMZ and serves static pages. A Web server redirector is used to forward the requests from the Web server to the application server.

Access to the application server's resources is protected by the application server's security features. User authentication is implemented by the directory and security services node.

This topology is especially useful when you do not expect many clients to access the server simultaneously and when you want to have additional security by separating the Web server from the application server.

Benefits

Since the Web server is separated from the application server, additional security is available and the business logic on the application server is protected by both the protocol and the domain firewall.

Limitations

This topology has the following limitations:

- The topology has limited availability and failover capability.
- Horizontal scalability is not possible because there are no means to allow for more than one Web server or Web application server. See 3.3.2, “Proven variation 1” on page 33 for a solution.
- Only a limited amount of vertical scalability is possible. Vertical scalability can be achieved by adding memory or processors, creating duplicates or clones of applications on the application server. See 4.1.2, “Clones running on application servers” on page 45 for more details.
- Since only one Web server is used, only a limited number of clients can be served simultaneously. This is because each Web server has to allocate limited resources, such as memory, sockets, processes, or threads, etc., whenever it serves an incoming request.
- Since the requests to the application server need to be forwarded, you could see a noticeable performance degradation, depending on the redirector solution chosen. A reverse proxy is an alternative way to implement a redirector and has little performance degradation.
- In addition, a firewall needs to be configured to allow traffic from the redirector to the application server. But it is this firewall that delivers additional security.

3.3.4 Emerging multi-tier variation 3

Depending on the nature of the application logic and the customer’s security policy, you may need to locate the business logic behind the DMZ while maintaining the presentation logic on a Web application server in the DMZ.

Also, in some situations, you may want to design an application server that can be accessed by thick clients, perhaps using a protocol other than HTTP, and with a richer presentation logic at the client. Or you may have to integrate an existing application server that has been designed to be accessed with a protocol other than HTTP in the process of defining an architecture.

An example scenario could be comprised of a CORBA application server behind the DMZ and a Web application server that implements the presentation logic and augments the business logic on the CORBA server. In this scenario, you would use IIOP to communicate with the application server.

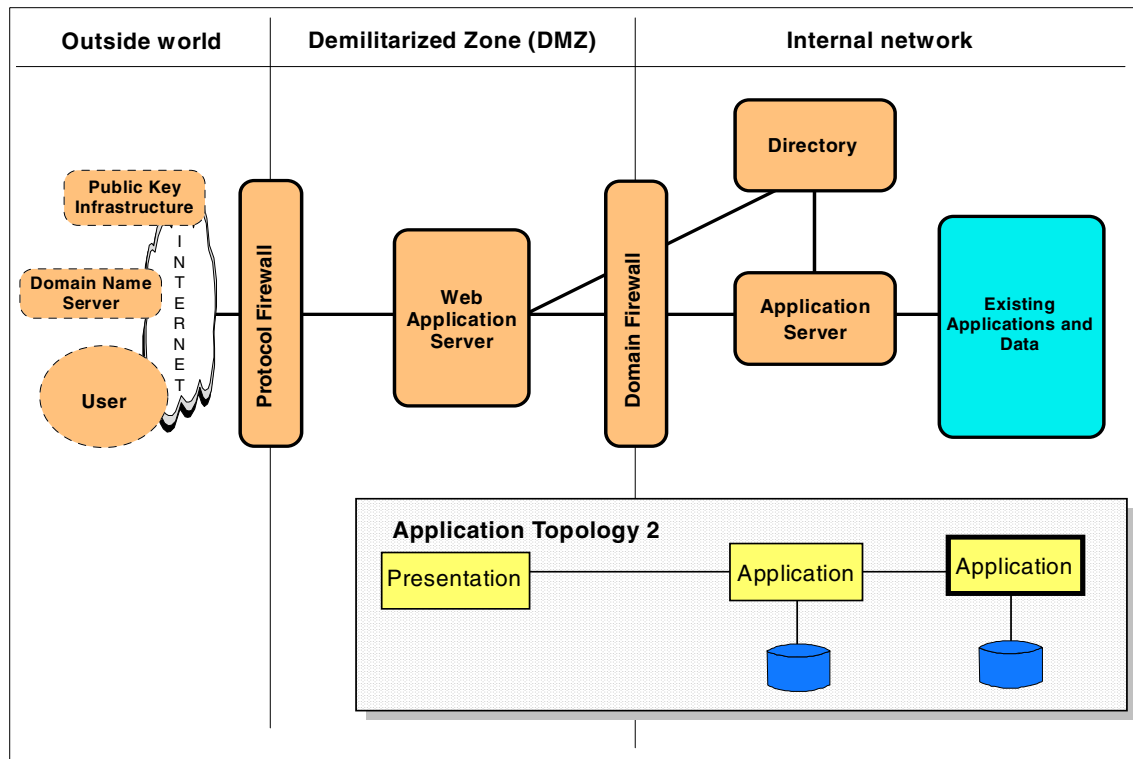


Figure 12. Runtime topology B (emerging multi-tier variation 3)

This variation of the basic topology adds an additional application server behind the DMZ to provide more security. A single Web application server within the DMZ is used for presentation logic, but all business logic is implemented within the application server behind the DMZ and in other existing applications. In this case the application server will be used for running business logic only (no presentation logic), providing a clear layering between the presentation logic and the business logic.

This topology does not specify the choice of protocol between the Web application server and the application server. Depending on the application server it may be HTTP, RMI, or RMI/IIOP.

The application server technology to use is an important implementation decision. Some candidates to consider are an EJB server, a traditional transaction monitor, a CORBA server, or an RMI server. Using an RMI server is not recommended because session beans are a better choice. But in some situations, an RMI server may already be present.

Access to the application server's resources is protected by the application server's security features. User authentication is implemented by the directory and security services node.

Benefits

Since the Web application server that implements the presentation logic is separated from the application server that implements the business logic, additional security is available and the business logic is protected by both the protocol and the domain firewall.

Limitations

This topology has the following limitations:

- Horizontal scalability is not possible because there are no means to allow for more than one Web application server. See 3.3.2, "Proven variation 1" on page 33 for a solution which, combined with this runtime topology, provides for optimal scalability.
- Only a limited amount of vertical scalability is possible. Vertical scalability can be achieved by adding memory or processors, and/or creating duplicates or clones of applications on the Web application server. See 4.1.2, "Clones running on application servers" on page 45 for more details.
- Since only one Web server is used, only a limited number of clients can be served simultaneously. This is because each Web server has to allocate limited resources, such as memory, sockets, processes, or threads, etc., whenever it serves an incoming request.
- Since the requests to the actual application server need to be forwarded, there is a slight performance degradation to be expected.

3.4 Intranet vs. Internet runtime topologies

The intranet runtime topologies do not differ significantly from the Internet runtime topologies. There are, however, certain issues that you should be aware of:

1. The bandwidth is usually higher than on the Internet. This allows you to use more bandwidth-consuming solutions, such as "thick clients". Downloading these clients may be impractical on the Internet.

Nonetheless, even within intranets, bandwidth may be limited, since not all employees may have equivalent access to the intranet. For example, think of someone working at home and connecting via telephone lines to the intranet. Also, when a company is widespread, some branches may have only telephone lines (say ISDN), to access the intranet.

2. Security may be less of an issue, as you have better control over who is accessing the network. But protecting the network is important even within the intranet, so protocol firewalls should not be left out and domain firewalls are important too, because you want to protect the back-end systems from unauthorized access.
3. Due to corporate rules, you may expect certain browsers to be used. Once you know which browsers are supported within the intranet, you can design and implement the solutions with these browsers in mind. This can be very productive, because you do not have to use the “least common denominator” but rather can utilize all available features.

This list of issues is not complete. They are listed here as examples of what you may have to consider.

Chapter 4. Product mapping

After having chosen a runtime topology option, you will then want actual products mapped against the runtime topology. It is suggested that you make the final platform recommendation based on the following considerations:

- Existing systems and platform investments
- Customer and developer skills available
- Customer choice

The platform chosen should fit into the customer's environment and ensure quality of service, such as scalability and reliability so that the solution can grow along with the e-business.

We have implemented a sample application topology 1 using IBM WebSphere Advanced Edition in a heterogeneous Microsoft Windows NT and IBM AIX environment. This chapter explains in more detail what products have been used in the process of implementing this application topology.

4.1 Runtime topology options

Before we start with the detailed description of the topologies, some options are discussed.

4.1.1 Implementing a redirector

The current version of WebSphere Advanced Edition (3.021) provides a servlet redirector feature. Another option in WebSphere is the OSE Remote feature. A third option is to use a reverse proxy.

In our samples, we used the WebSphere servlet redirector. If you are using WebSphere 2.0, then the reverse proxy is the only way for you to separate the Web server from the application server and put a firewall between them.

4.1.1.1 Using WebSphere's servlet redirector

The servlet redirector is a process that distributes servlet requests to machines remote to the Web server. With WebSphere Application Server 3.021 there are two configurations of the redirector available, base ("thick") and standalone ("thin").

Base ("thick") redirector

The base redirector runs as part of the administrative server, and therefore has the overhead of requiring the administrative server's infrastructure (the

database). The advantage of the base redirector is that it can be configured and managed using the WebSphere administrative console from either the redirector's or application server's node. The base redirector supports WebSphere security.

Standalone ("thin") redirector

The standalone redirector does not require the administrative server's infrastructure and therefore does not suffer from that overhead. It is especially suitable when you have limited resources or do not want to expose any information in the administrative database. Its disadvantage is that it cannot be configured and managed using the WebSphere administrative console. All configuration is done by means of scripts that need to be executed on a regular basis. This means that the redirector can be out of sync whenever the application server has changed and the changes have not been propagated to the Web servers. The standalone redirector does not support WebSphere security.

It depends on your requirements as to which version is a feasible alternative. If access to secured resources is not an issue, either one can be used to separate the Web server from the application server. If, however, resources are secured, then only the base redirector will be an option for you.

Additional information on the servlet redirector can be found on the Web at: <http://www.ibm.com/software/webservers/appserv/doc/v30/ae/web/help/model.htm#a9> as well as in Chapter 15, "Setting up a standalone servlet redirector" on page 297.

4.1.1.2 Using reverse proxy

An alternative way to implement a redirector is to use a so-called reverse proxy. A reverse proxy is a method of making a proxy server transparent to the client. A reverse proxy listens on port 80 for requests that have a certain format. It then forwards those requests to an HTTPServer that resides on the Web application server. The requests are then fulfilled and passed back to the reverse proxy to the client.

When a proxy server is configured for reverse proxy, it appears to the client to be the origin server. The client is not aware that the request is actually being sent to another server.

4.1.1.3 OSE Remote

WebSphere also offers the OSE Remote feature. Configuring the HTTP Web server for OSE Remote offers a significant performance improvement over the servlet redirector, but will not support SSL between the Web server and

application server. To use SSL you will need to use the servlet redirector. OSE Remote uses its own private protocol.

4.1.2 Clones running on application servers

In order to increase the vertical scalability of an application server, you can increase the number of processors installed on that particular server. Each of these processors can then execute in parallel. However, a performance improvement is only possible if either the software or the operating system can make use of these additional resources.

The IBM WebSphere Application Server allows you to leverage remaining processor capacity or additional processors by means of application "cloning". Cloning means that a given application is duplicated such that the clients cannot distinguish between the clones. In addition, each WebSphere Application Server (this is WebSphere's term for grouping a servlet engine and related resources) is running in its own JVM. This allows the use of more than one JVM with WebSphere.

Before cloning, you first have to create a model of that resource. Clones are created from a model. After you clone a resource, modifying the model automatically propagates the same changes to all of the clones. You can efficiently administer several copies of a server or other resource by administering its model.

You can also clone servlets, servlet engines, Web applications, EJB containers, and Enterprise Java beans. For details on cloning, please see the Web site at:

<http://www.ibm.com/software/webservers/appserv/doc/v30/ae/web/help/model.htm>.

There are, of course, situations where it is necessary to distinguish between the clones. For more information see 4.1.3, "Session sharing across servers" on page 46 and 4.1.4, "Achieving HTTP session affinity" on page 47.

The application server balances the workload of the clones running on a server automatically. Thus, you do not have to worry about the machine's utilization. All of that is done automatically by the application server.

There are circumstances where cloning is desirable even if you have only one processor installed. This can be the case if you have an application that is spending most of its time waiting for some resources. During this time, additional requests can be served by the application's clones. Also, if

automatic tasks such as Garbage Collection take too long to complete, cloning may prove a viable alternative on a single-processor machine.

In addition to all the above-mentioned advantages, cloning also increases the availability of a particular Web application as well as the failover capability. If any of the clones fails, the other clones take over the workload.

Overall, it is highly application dependent as to whether or not a performance improvement can be achieved by cloning of applications.

As with all alternatives there are also disadvantages that you have to consider:

1. If you do not require session affinity and want all session-related information to be transparent to the users, there is some performance penalty because all session information needs to be saved and retrieved from a database. Depending on the amount of data, this penalty may prove to be very expensive. Session affinity is discussed in 4.1.4, “Achieving HTTP session affinity” on page 47
2. If your application assumes that it is running on a dedicated machine (even on a dedicated JVM), cloning will not be an issue for you because it cannot be determined in advance on which machine your application will execute the next time a request is served.

4.1.3 Session sharing across servers

Whenever you distribute an application across several servers, either by means of cloning or duplication, you have to think about how the sessions will be handled by this configuration. Normally, there should be no problems when running such a distributed application. This is especially true if the application controller does not maintain any information or state between two consecutive client requests. For simple applications this should always be the case.

As soon as the application becomes more complex and requires that state be maintained by the controller, distribution can lead to unexpected results. This is because there is no guarantee that the same controller will be invoked the next time a client sends a request. And whenever the controller is making this assumption, it will fail.

To be able to overcome this problem when running multiple servers, the session state must be saved to a database and whenever an application accesses the current session's state, it must be reloaded from the database. This saving and loading takes time. And depending on the amount of data that is to be written, it can add up to a substantial amount.

If the volume of data to be written to the database is almost negligible, then the time to establish the database connection will be much larger than the actual time to put the data into the database and later on to fetch it from the database. In other words, if only small amounts of data are to be maintained, the overhead to access this data will become too large. On the other hand, if the amount of data to be written to the database is very large, say, several kilobytes, then the time to save and read the data can take up a substantial amount of the time spent in the controller.

These are the two extremes one has to consider. Of course, if the session state can be computed somehow, then persistent sessions are not needed and the state should always be re-computed whenever a request is served by the controller. As this may not always be possible, you have to be sure how much you are willing to pay for session sharing. Even if you have all clones running on the same physical machine, and sessions do not have to be stored in a database, it is up to the application server to make the decision as to whether or not it will need to access a database.

Session sharing has a tremendous advantage. Since the session state is not stored within a particular server, the availability of the service increases because each time a particular server fails, other servers can take over.

4.1.4 Achieving HTTP session affinity

The term “session affinity” applies whenever a client is always connected to the same server during a session. This server may have been determined in advance or during the first request of that session. In the latter case, this server will be serving the client’s subsequent requests.

There may be several reasons for session affinity to be a property that you want to leverage. For example, due to resource restrictions you cannot allow some other controller to continue when the next request comes in. Or maybe the overhead related to persistent sessions is too high for you and therefore the application performance degrades.

If you need session affinity, you must make sure that after serving a request, the next request is sent not only to the same machine but also to the same clone running on that machine. However, since clones cannot be distinguished, there is no way you can be sure a request is sent to some particular clone. This being the case, you cannot take advantage of cloning applications. However, you can still have several applications installed on that particular server that will utilize the server’s processing power. As for ensuring that all subsequent requests are served by the same server, you will have to use absolute paths such as:

`Title Page`, as opposed to relative paths such as `Title Page`.

A disadvantage of session affinity is that workload management is basically disabled after the initial connection. Workload management will occur at session instantiation if you are using something like the IBM SecureWay Dispatcher, but after that the client will connect directly to the same server.

Also, the availability of a service decreases with session affinity. Because a client is bound to a particular server, no other server can take over should the server fail.

4.2 Product mapping for basic runtime topology A

This mapping shows the products and platforms used in the implementation of the basic runtime topology A discussed in 3.2.1, “Proven basic topology” on page 23. Though we chose Windows NT for many of the nodes, keep in mind that these products are available on other platforms.

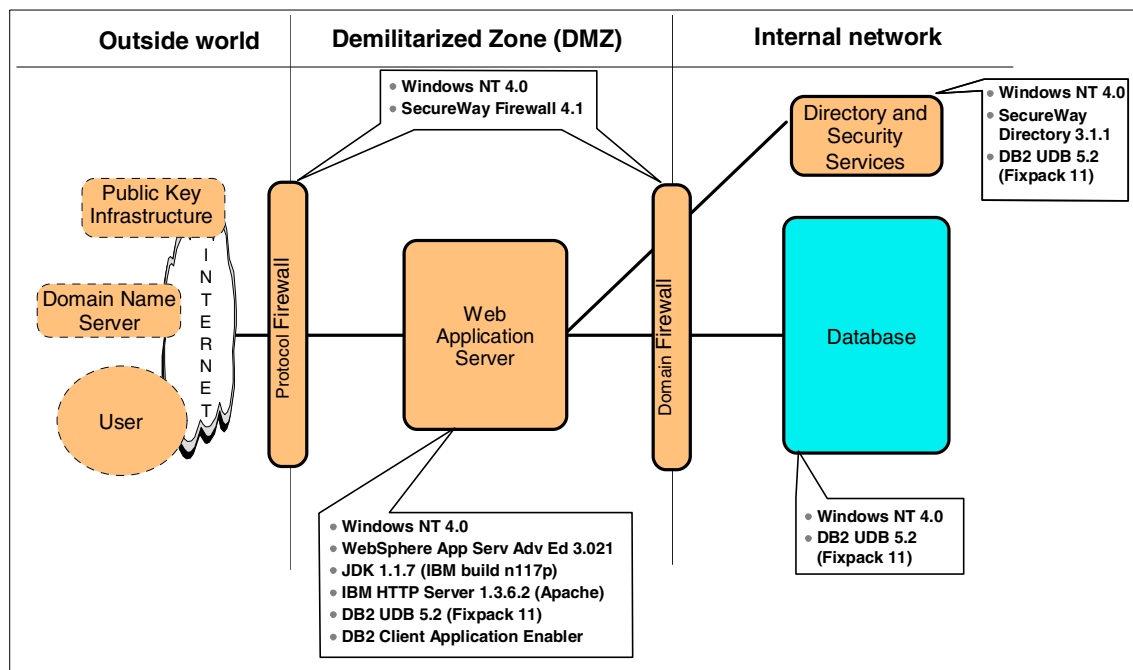


Figure 13. Product mapping, basic runtime topology A, Windows NT based

In this scenario:

- The protocol firewall was configured to be open on port 80 only. Thus, only HTTP traffic could flow from the Web browsers to the Web application server.
- The domain firewall was configured to be open on port 389 to access the LDAP server, and on port 50000 to access the database.
- The Web application server was hosting the business logic which, in our case, accessed data stored in the database within the internal network.
- User authentication was implemented using WebSphere's security features and users had to authenticate against the LDAP server.

An alternative for the Web application server is shown in Figure 14 using AIX for the WebSphere platform.

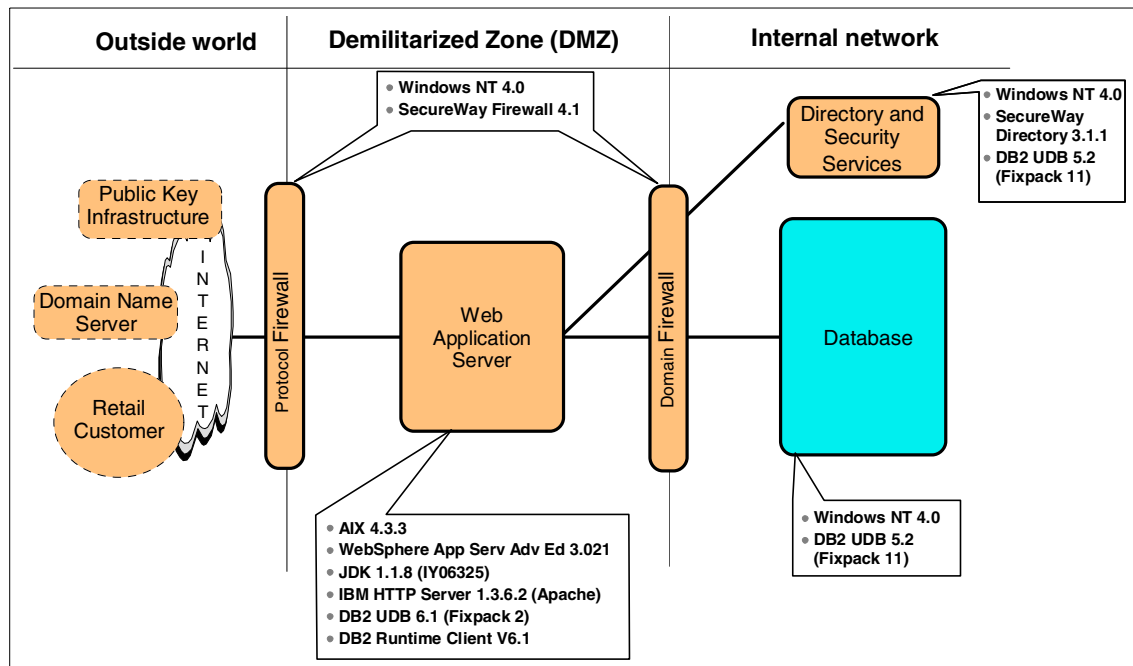


Figure 14. Product mapping, basic runtime topology A, AIX based

4.3 Product mapping for variation 1 of runtime topology A

This mapping shows the products and platforms used in the implementation of the first variation to runtime topology A discussed in 3.2.2, "Proven

variation 1” on page 25. Though we chose Windows NT for many of the nodes, keep in mind that these products are available on other platforms.

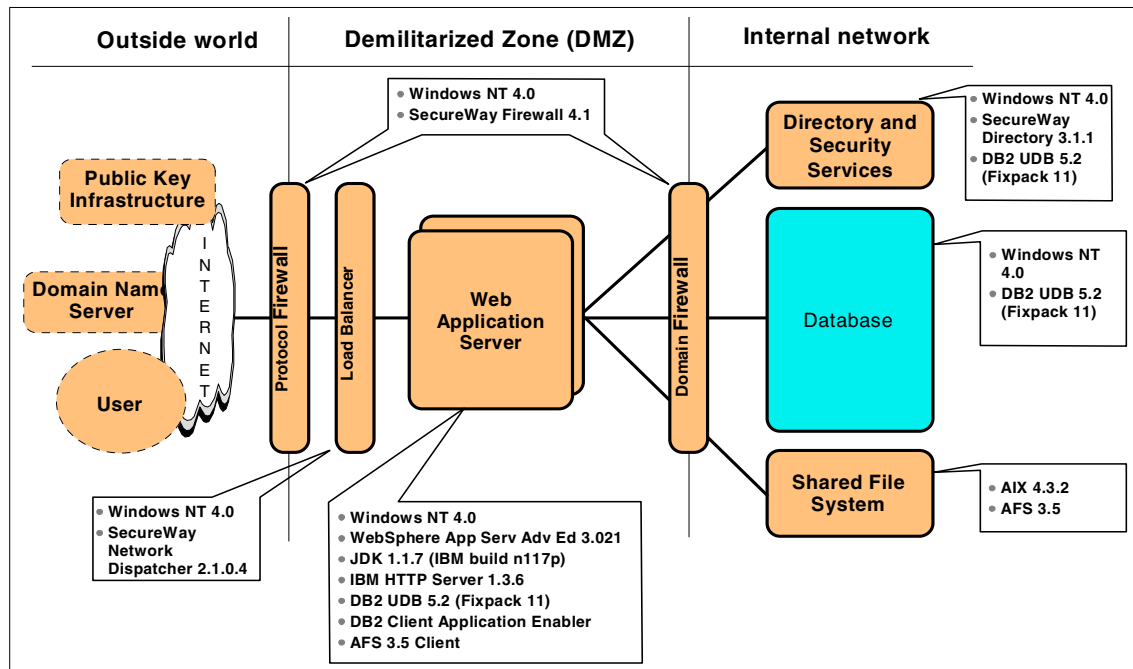


Figure 15. Product mapping, proven variation 1 of runtime topology A

This topology is an extension of the basic runtime topology, adding multiple Web application servers and load balancing capability. In this scenario:

- The protocol firewall was configured to be open on port 80 only. Thus, only HTTP traffic could flow from the Web browsers to the load balancer.
- The domain firewall was configured to be open on port 389 to access the LDAP server, and on port 50000 to access the database.
- The Web application server was hosting the business logic which, in our case, accessed data stored in the database within the internal network.
- User authentication was implemented using WebSphere's security features and users had to authenticate against the LDAP server.
- IBM's SecureWay Network Dispatcher was used to distribute (spray) incoming HTTP requests to several identical copies of the Web application server.
- Each of the Web application servers had the same data and code installed, so there was no need to install a shared file system in this

instance. However, if needed, a shared file system such as AFS could have been used.

- On each of the Web application servers in the DMZ, an alias needed to be included in the default host's alias list to include the cluster address (and/or cluster name) that has been created for the Network Dispatcher.

4.4 Product mapping for variation 2 of runtime topology A

This mapping shows the products and platforms used in the implementation of the second variation to runtime topology A discussed in 3.2.3, "Emerging variation 2" on page 27. Though we chose Windows NT for many of the nodes, keep in mind that these products are available on other platforms.

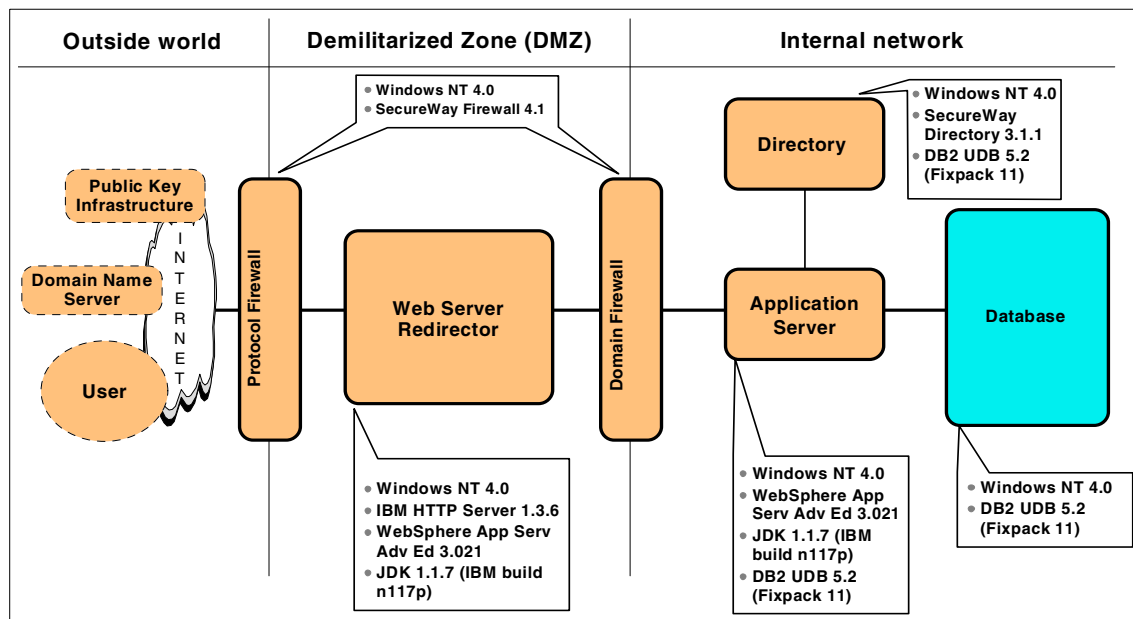


Figure 16. Product mapping, emerging variation 2 of runtime topology A

In this scenario a servlet redirector is installed in the DMZ. In this case, WebSphere was configured as a thin redirector on a machine, though a base redirector could have been used as well. The process used to set up a redirector is covered in Chapter 15, "Setting up a standalone servlet redirector" on page 297.

The application server (WebSphere Application Server) was moved to the secure network.

In this scenario:

- The protocol firewall was configured to be open on port 80 only. Thus, only HTTP traffic could flow from the Web browsers to the Web server.
- The domain firewall was configured to be open on ports needed to allow IIOP traffic to flow between the redirector and the application server. See Chapter 16, “Setting up firewalls” on page 305 for information on configuring the firewalls.
- The application server was hosting the business logic which, in our case, accessed data stored in the database within the internal network.
- User authentication was implemented using WebSphere’s security features and users had to authenticate against the LDAP server.

In Part 3, “Application topology 1: a working example” on page 229 we used the servlet redirector feature of WebSphere Advanced Edition 3.021 to implement variation 2 of runtime topology A. Another option to implement the redirector function in this topology would be the OSE Remote feature in the WebSphere Advanced product. For some development test cases, OSE Remote has demonstrated a performance advantage over servlet redirector but in its first implementation uses a private protocol, whereas servlet redirector uses RMI over IIOP. If SSL is required for the connection then the servlet redirector must currently be used.

Part 2. User-to-Business patterns: guidelines

Chapter 5. Performance guidelines

This chapter discusses the various aspects of performance and how the design or implementation of an e-business application can affect them. The majority of the information was extracted from the *Designing e-business Solutions for Performance* white paper, by Maggie Archibald and Mike Schlosser. The white paper contains much useful information and is definitely worth reading. This chapter contains only a small portion of the information. The white paper can be found at:

<http://www.ibm.com/software/developer/library/patterns/performance.html>

There are many specific guidelines that relate to various components of a solution that will be discussed later. There are also some general guidelines that apply to the designs of all solutions:

1. Pay attention to all components of the solution.
2. Understand in detail the interfaces and flows between the various components.
3. Plan for growth of the design.
4. Use the latest levels of infrastructure and system software.
5. Cache as much as possible.

5.1 Web server performance considerations

The Web server is responsible for serving up everything from static pages to invoking various applications through interfaces such as Common Gateway Interface (CGI), FastCGI, Web server Application Programming Interfaces (APIs) and servlets. Because of this varied workload, careful planning is needed to ensure that the Web server is processing the request in the most efficient manner. Segmenting the work into similar functions is critical to producing consistent response times. Another technique to improve performance is to use the workload classification to make prioritized decisions on how to shed workload if the server becomes overloaded. Specific items in a Web server that impact performance are:

- Threads

Some Web servers allow the configuration of both a minimum and maximum number of threads. Pick a reasonable, but large enough number of threads to handle your peak workloads, then set the maximum number of threads equal to the minimum number.

This avoids the overhead of creating and destroying threads during peak processing hours. Experiment to find the right number of threads on your

system. Experience has shown that picking too high a number can decrease your overall throughput on the system. Too low a number can even cause server failure. Remember in your calculations that the Web server uses some of the allocated threads for its own processing. In the case of the ICSS Web server version that was tested for instance, nine threads were used by the server for its own purposes.

Currently, IBM HTTP Server for Windows NT supports threads, but AIX does not.

- Limit the use of server-side includes (SSI)

Server-side includes (SSI) allow you to insert information into CGI programs and HTML documents that the server sends to the client. When server-side include processing is enabled, the Web server will parse each byte of every HTML file and CGI program searching for the existence of an SSI directive and, if found, process it. This is a great feature for processing dynamic content, but it requires a large amount of CPU processing.

SSI processing can be controlled by the use of the `imbeds` directive. If you do not use SSIs, set `imbeds` off in the `/etc/httpd.conf` file.

- URL links

URL suffix processing (multi-support) is overhead for any Web server. This occurs when a URL does not exactly match the templates on the `PASS` directive. For example, if the URL is `/oh_boy.html` and your file name is actually `/oh_boy.html.ascii`, the Domino Go Web server will do suffix processing and eventually return the file `oh_boy.html.ascii`. The Web server appends all known suffixes to the file name looking for a match. Eventually, the correct file will be returned, if it exists, but the process is CPU intensive.

- Caching

The addition of memory to a system almost always improves performance. This is because physical I/O is a relatively expensive operation in terms of latency. It makes intuitive sense then, that by dedicating memory in a Web server to store frequently accessed HTML pages and images, you will improve performance. As a rule of thumb, your Web server should have enough RAM to accommodate all network buffers, frequently used applications, images and HTML, including those mounted via DFS. This is especially important for dynamically generated pages that can be reused.

For the Web server there are several places you can cache your static items to help improve performance:

- Use a network router such as the IBM 2216 Nways Multiaccess Connector.

- Use a Web proxy cache such as the one found in WebSphere Performance Pack.
- Logging

Web server logs, in particular the access log, record important information about the use of the Web server. However, these logs can become very large and should be pruned and/or archived regularly. Logging can have a surprisingly large impact on response time and throughput. For this reason, you will often find that vendors turn logging off for bench marking purposes.
- CGI and server-side Java

The IBM Application Framework for e-business defines the infrastructure for developing e-business solutions. This includes the relationship between the Web server and server-side Java elements. IBM's analysis of the performance characteristics of this environment provides some useful guidelines for e-business solution designers, including:

 - a. Any of the techniques used to connect a Web server to application logic (CGI, in-process API, Java servlets, etc.) should be insignificant when compared to the time required to execute the application logic.
 - b. Most existing Web applications have been implemented using CGI. IBM's analysis indicates that Java servlets provide 4X to 10X better throughput compared to CGI.
 - c. Java servlets generally run faster if they are instantiated and preloaded in `servlet.properties`. Servlet invocation by class name rather than instance name is slower.
- Network

If there is no contention for either CPU or memory resources on your system, and you are experiencing performance problems, you may have a network issue to resolve. After all, while incoming Web requests may be relatively small, outgoing Web responses can contain large graphics, applets, video or audio files. It is important to make sure that the number and size of the TCP buffers be tuned appropriately on the Web server platform. Because the size of requests coming into the server is so different than the requests going out of the server, many sites have routed incoming requests along relatively "thin" network pipes while routing their output requests along relatively "fat" network pipes.

- Load balancing

The need to manage and scale Web sites that experience high “hit rates” over the Internet has led to the development of load balancing application server designs. Load balancing designs range from the relatively simple round robin Domain Name Systems (DNS) approaches to more sophisticated dispatchers such as the IBM SecureWay Network Dispatcher that include server-side monitors.

Unless the workload is very consistent across all the clients in terms of the server load required and the volumes requested, and the servers are identical in capacity, the round robin approach can cause some servers to bear more load than others. If some users generate requests for relatively more CPU intensive CGI calls or other types of dynamic pages versus some users who merely request static pages that may be in cache, the round robin approach usually does not balance the workloads. Because the round robin approach usually is not aware of current server workloads or the impact of the request being assigned, it just assigns the first request from a user to the next server in the rotation. This approach is more of a user allocation approach versus a true load balancing approach.

The most common approach to load balancing today is Web server clustering. A clustered environment in the Web world is a set of Web servers that appear to browsers on the Internet as a single server. A required element in a clustered environment is a load balancer, which is software or hardware that is responsible for making the set of Web servers appear to be a single server. For example, the SecureWay Network Dispatcher component of IBM's WebSphere Performance Pack provides this function.

- Segmenting workload

If the workload is not very consistent across all of the Web servers, as is the norm in most cases, then another approach needs to be implemented to normalize the response times to the clients. The best way to normalize response times to requests is to split up the workloads into groups of similar characteristics. For example, static HTML and images are good candidates for one group, while simple CGI/WSAPI programs belong in another group, and complex CGI/WSAPI programs belong in a third.

This segmenting of the workload can be done at the hostname level in the URL or you can use a load balancer to do content-based routing. It is generally recommended that you do URL-based segmentation since it is less of a performance hit on the load balancer.

5.2 Integration server performance considerations

Integration of Web application servers with back-end systems also raises several performance issues, including:

1. Pre-allocating and caching resource manager:

Some of the cost associated with accessing resource managers from middle-tier objects involves establishing connections to those resource managers. By pre-allocating and reusing connections, it is possible to greatly reduce the overhead of defining new ones.

2. Caching facilities local to the middle tier:

Caching state information accessed from back-end systems into the middle-tier environment can promote good performance. This is achievable through pre-fetch and look-aside algorithms that ensure that the back-end will only be accessed as often as is absolutely necessary. Note that it is common, especially when integrating with existing legacy information systems, to access back-end state information required by one object and, in the process, to encounter state information needed by one or more other, related objects. Caching this information as it becomes available may significantly improve overall system performance. Finally, a smart caching mechanism can ensure that updates are only made to a back-end system when the underlying state information of a given object has actually changed.

3. Optimistic locking mechanisms:

In some cases, applications place large numbers of unnecessary locks on resource managers. We say these locks may be unnecessary in the sense that no attempts to use the resource concurrently actually occur. In these situations it would be better to place no locks on the resource managers, and instead check for conflicting usage as part of transactional commit. Specifically, you should consider locking all affected resources only once: at the end of a given unit of work. Then compare the relevant resource manager values at that time with the values obtained when the unit of work began. In the absence of intervening changes, the current unit of work can be committed. Otherwise, it can be rolled back with an exception returned to the client. In either case, you may be able to reduce overall locking overhead and improve total system throughput and performance. Note that this “optimistic” locking strategy is inappropriate for some types of applications and domains. When updates to the same resource occur on a frequent basis, traditional (or “pessimistic”) locking mechanisms should be used. It’s also worth mentioning that multiple resources should typically be

accessed by multiple applications in the same sequence to reduce potential “deadlock” situations.

4. Implementation “pushdown”:

Object-based solutions that must coexist with legacy systems should complement (versus compete with) these systems. For example, a query invoked on a virtual collection of objects whose state maps to a relational database manager should first be translated into native SQL statements. These SQL statements, processed using the optimization technology provided by the database manager, can then return a result set whose values implicitly identify candidate objects satisfying the query. In this way, a minimal amount of processing is required in “object space”, leaving the bulk of the work to be handled instead by resource managers that have been perfecting high performance solutions over the course of many years. As a final point that is specific to our query example, it’s notable that an object-based implementation of query should also be able to return multiple elements back from a single method call, and that the query result set should be demand-driven (meaning that objects should only be activated on the server if a client actually requests them).

5.3 Java and Java Virtual Machines

Java is an interpreted language. Java source code must first be compiled into portable bytecodes that can then be interpreted in the Java Virtual Machine (JVM) on the local system. Naturally, when looking for performance improvements for your Java applications, the quality of the JVM is the place to start. For example, all JVMs implement an advanced feature called Garbage Collection (GC) to boost the programming productivity and to avoid the common pitfalls of traditional programming languages: memory leaks. However, GC can slow down the Java program execution because most GCs utilize a “stop and copy” technology. Due to ongoing research, GCs have been developed that do not exhibit any of the problems described above. These GCs run incrementally and take many characteristics of the Java language into account.

5.3.1 Just-In-Time compiler

A Just-In-Time (JIT) compiler converts bytecodes into native code on-the-fly with some optimization, and it should run much faster than just a Java Virtual Machine (JVM). A JIT works well for computationally intensive programs that execute the same segment of instructions repeatedly. But it does not yield significant improvement for programs that are I/O intensive or cause a lot of

garbage collection. This is because the program code takes up such a small amount of time and thus any optimizations would become negligible.

In addition, JITs are limited in the extent of optimizations they can do because their compile is a runtime cost. Also, since JIT compilers normally do not have the “global view” of the executed code, the code they generate is of poorer quality than the code generated by static compilers.

5.3.2 Adaptive compilers

Adaptive compilers try to overcome the weaknesses of JIT compilers with adaptive optimization techniques. The idea is to intelligently select, compile and optimize frequently used and/or resource intensive portions of the code, so called “spots.” Once these spots have been determined, optimization techniques known from traditional compilers are applied to compile a highly optimized version of the code. The compiled code is then stored in a cache, ready to be executed when the spot is executed again. Note that as with a JIT, results of compilation are not kept between runs/users.

5.3.3 Static compiler

A static compiler compiles Java source code into the underlying machine’s native code that is then executed without interpretation. This approach is similar to traditional program development where the code is written, compiled and, if necessary, debugged.

Static compilation can also be applied to the bytecode generated by some Java compilers. This approach is applicable when the original Java source code is not available. Static compilation of Java bytecodes is possible because this code is the same across Java implementations and builds the basis for the portability of the Java language.

However, it is important to note that the portability of the Java application will not be affected by static compilation.

5.3.4 Selecting JVMs

High-performance JVMs are critical for good performance. Here are some considerations when evaluating JVMs:

1. Java compiler and virtual machine technology changes rapidly. Today a JIT version of a compiler may provide the best performance for your solution, tomorrow it may be a static compiler, and next week there may be a new technique.

2. A JVM should be certified for portability by a recognized certification authority such as JavaSoft.
3. JVMs that have been optimized for a specific operating system tend to perform better than other JVMs.
4. Systems with symmetric multiprocessors (SMP) tend to perform better because of the thread support in Java. Use JVMs that effectively support SMP systems.
5. Evaluate the trade-off between the stability of the current release of a JVM and the performance enhancements available in the newest beta release and/or production version.

Generally speaking, the selection of a JVM should be of little concern to you. The performance of applications can often be better improved by improving the runtime characteristics of the algorithms used. However, there are circumstances, like computationally intensive programs, where an improvement in the performance of the underlying JVM will add to the solution's overall performance.

5.4 Where to find more information

White papers:

- Maggie Archibald, Mike Schlosser: *Designing e-business Solutions for Performance* white paper at:
<http://www.ibm.com/software/developer/library/patterns/performance.html>
- JavaSoft: *The Java HotSpot Performance Engine Architecture* white paper at:
<http://java.sun.com/products/hotspot/whitepaper.html>

IBM Redbooks:

- *WebSphere V3 Performance Tuning Guide for AIX*, SG24-5657

Web sites:

Documentation for IBM WebSphere Application Server, including the product library (which includes IBM HTTP Server documentation), hints and tips, white papers, and other support information can be found at:

- <http://www.ibm.com/software/webservers/appserv/support.html>

Information on IBM WebSphere Application Server, specifically for AS/400, including documentation, support, and performance considerations, can be found at:

- <http://www.as400.ibm.com/WebSphere>

Performance information for the Domino Go Webserver for OS/390 can be found at:

- <http://www.s390.ibm.com/oe/perform/dgwperf.html>

Chapter 6. Technology options

This chapter looks at the technologies that you should consider for Web applications based upon the open standards and Java-based programming model of the IBM Application Framework for e-business. We do not attempt to cover the many technologies that can be used in developing Web applications, such as Perl or server-side JavaScript. The recommended technologies are outlined in the *IBM Application Framework for e-business Architecture Overview* white paper at:

http://www.ibm.com/software/ebusiness/arch_overview.html

We will look at the technologies as they apply to both the client and the server side of the application. Some technologies such as Java and XML can apply to both. Also, the selection of client-side technologies used in your design will require consideration for the server-side such as to whether to store, or dynamically create, elements for the client-side.

The sections that follow detail a number of technologies that you will want to consider in your design.

We recommend the following as technologies that are central to the Application Framework and its programming model:

- HTML
- Java servlets and Java Server Pages
- XML
- Connectors
- Enterprise Java beans
- JDBC / SQLJ
- Additional enterprise Java APIs

These technologies are used in the context of the following logical model for an e-business application, which we will explore in more detail in Chapter 7, “Application design guidelines” on page 81 and Chapter 8, “Application development guidelines” on page 147. This model, which has similarities to the Model-View-Controller approach in GUI development, characterizes the presentation logic as consisting of interaction control (implemented by Java servlets) and page construction (implemented by Java Server Pages). The business logic may be implemented using Java beans and/or enterprise Java beans depending on the transactional characteristics of the application. The business logic may need to access external resources using the appropriate connector technology.

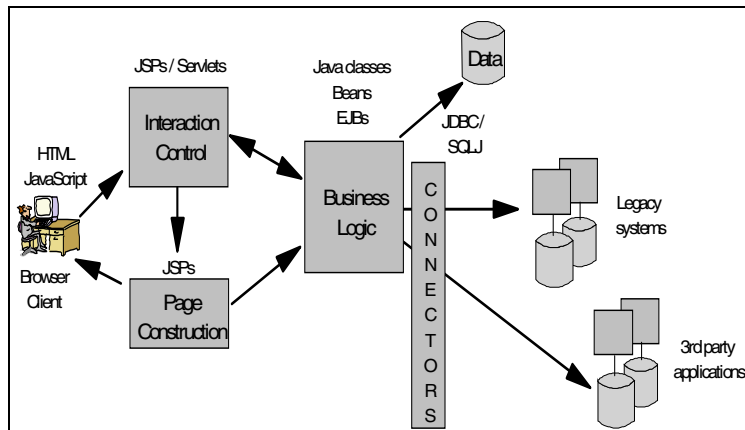


Figure 17. The logical structure of an e-business application using the recommended core technologies

We also include some discussion of the following technologies and the limitations involved in their usage:

- DHTML
- JavaScript
- Java applets

For more information, see the *IBM Application Framework for e-business Architecture Overview: Understanding Technology Choices* white paper, upon which significant portions of this chapter are based:

<http://www.ibm.com/software/ebusiness/buildapps/understand.html>

6.1 Web client

The Application Framework recommends the following technology model for a Web client.

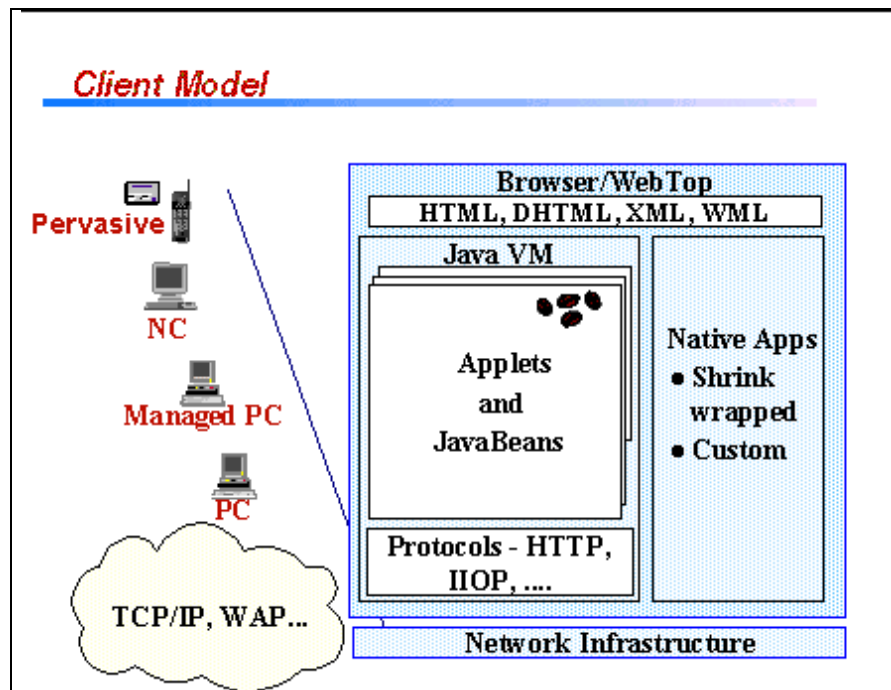


Figure 18. Web client technology model

The clients are “thin clients” with little or no application logic. Applications are managed on the server and downloaded to the requesting clients. The client portions of the applications should be implemented in HTML, dynamic HTML (DHTML), XML, and Java applets.

The following sections outline some of the possible technologies that you should consider, but remember that your choices may be constrained by the policy of your customer or sponsor. For example, for security reasons, only HTML is allowed in the Web client at some government agencies.

6.1.1 Web browser

A Web browser is a fundamental component of the Web client. For PC-based clients, the browser typically incorporates support for HTML, DHTML, JavaScript and Java. Some browsers are beginning to add support for XML as well. Under user control, there is a whole range of additional technologies which can be configured as “plug-ins”, such as RealPlayer from RealNetworks or Macromedia Flash.

As an application designer you must consider the level of technology you can assume will be available in the user's browser, or you can add logic to your application to enable slight modifications based upon the browser level. Regarding plug-ins, you need to consider what portion of your intended user community will have that capability.

For an e-business application that is to be accessed by the broadest set of users with varying browser capabilities, the client is often written in HTML with no other technologies. On an exception basis, limited use of other technologies, such as using JavaScript for simple edit checks, can then be considered based on the value to the user and the policy of the organization for whom the project is being developed.

The emergence of pervasive devices introduces new considerations to your design with regard to the content streams that the device can render and the more limited capabilities of the browser. For example, WAP (Wireless Application Protocol) enabled devices render content sent in WML (Wireless Markup Language).

6.1.2 HTML

HTML is a document markup language with support for hyperlinks, that is rendered by the browser. It includes tags for simple form controls. Many e-business applications are assembled strictly using HTML. This has the advantage that the client-side Web application can be a simple HTML browser, enabling a less capable client to execute an e-business application.

The HTML specification defines user interface (UI) elements for text with various fonts and colors, lists, tables, images, and forms (text fields, buttons, checkbooks, radio buttons). These elements are adequate to display the user interface for most applications. The disadvantage, however, is that these elements have a generic look and feel, and they lack customization. As a result, some e-business application developers augment HTML with other user interface technologies to enhance the visual experience, subject to maintaining access by the intended user base and compliance with company policy on Web client technologies.

Because most Web browsers can display HTML Version 3.2, this is the lowest common denominator for building the client-side of an application.

6.1.3 Dynamic HTML

DHTML allows a high degree of flexibility in designing and displaying a user interface. In particular, DHTML includes cascading style sheets (CSS) that enable different fonts, margins, and line spacing for various parts of the

display to be created. These elements can be accurately positioned using absolute coordinates.

Another advantage of DHTML is that it increases the level of functionality of an HTML page through a document object model and event model. The document object enables scripting languages such as JavaScript to control parts of the HTML page. For example, text and images can be moved about the screen, and hidden or shown, under the command of a script. Also, scripting can be used to change the color or image of a link when the mouse is moved over it, or to validate a text input field of a form without having to send it to the server.

Unfortunately there are several disadvantages with using DHTML. The greatest of these is that two different implementations (Netscape and Microsoft) exist and are found only on the more recent browser versions. A small, basic set of functionality is common to both, but differences appear in most areas. The significant difference is that Microsoft allows the content of the HTML page to be modified by using either JScript or VBScript, while Netscape allows the content to only be manipulated (moved, hidden, shown) using JavaScript.

Because of browser compatibility issues, DHTML is not recommended in environments where mixed levels and brands of browsers are present.

6.1.4 XML (client-side)

XML allows you to specify your own markup language with tags specified in a Document Type Definition (DTD). Actual content streams are then produced which use this markup. The content streams can be transformed to other content streams, by using XSL (eXtensible Stylesheet Language).

For PC-based browsers, HTML is well established for both document content and formatting. The leading browsers have significant investments in rendering engines based on HTML and a Document Object Model (DOM) based on HTML for manipulation by JavaScript.

XML seems to be evolving to a complementary role for active content within HTML documents for the PC browser environment.

For new devices, such as WAP-enabled phones and voice clients, the data content and formatting is being defined by new XML schema (DTD), WML for WAP phone and VoiceXML for voice interfaces.

For most Web application designs, you should focus your attention on the use of XML on the server-side. See 6.2.4, “XML” on page 75 for additional discussion of the server-side use of XML.

6.1.5 JavaScript

JavaScript is a cross-platform object-oriented scripting language. It has great utility in Web applications because of the browser and document objects that the language supports. Client-side JavaScript provides the capability to interact with HTML forms. You can use JavaScript to validate user input on the client and help improve the performance of your Web application by reducing the number of requests that flow over the network to the server.

ECMA, a European standards body, has published a standard (ECMA-262) which is based on JavaScript (from Netscape) and JScript (from Microsoft) called ECMAScript. The ECMAScript standard defines a core set of objects for scripting in Web browsers. JavaScript and JScript implement a superset of ECMAScript. You can find the ECMAScript Language Specification at:

<http://www.ecma.ch/stand/ECMA-262.htm>.

To address various client-side requirements, Netscape and Microsoft have extended their implementations of JavaScript in Version 1.2 by adding new browser objects. Because Netscape's and Microsoft's extensions are different from each other, any script which uses JavaScript 1.2 extensions must detect the browser being used, and select the correct statements to run.

The use of JavaScript on the server side of a Web application is not recommended, given the alternatives available with Java. Where your design indicates the value of using JavaScript, for example for simple edit checking, use JavaScript 1.1, which contains the core elements of the ECMAScript standard.

JavaScript: The Definitive Guide, Third Edition, by David Flanagan, is an excellent book on JavaScript which details the JavaScript objects and methods listing their origin and JavaScript level.

6.1.6 Java applets

The most flexible of the user interface (UI) technologies that can be run in a Web browser is offered by the Java applet. Java provides a rich set of UI elements that include an equivalent for each of the HTML UI elements. In addition, because Java is a programming language, an infinite set of UI elements can be built and used. There are many widget libraries available that offer common UI elements, such as tables, scrolling text, spreadsheets, editors, graphs, charts, etc.

A Java applet is a program written in Java that is downloaded from the Web server and run on the Web browser. The applet to be run is specified in the HTML page using an APPLET tag:

```
<APPLET CODEBASE="/mydir" CODE="myapplet.class" width=400 height=100>
  <PARAM NAME="myParameter" VALUE="myValue">
</APPLET>
```

For this example, a Java applet called myapplet will run. An effective way to send data to an applet is with the use of the PARAM tag. The applet has access to this parameter data and can easily use it as input to the display logic.

Java can also request a new HTML page from the Web application server. This provides an equivalent function to the HTML FORM submit function. The advantage is that an applet can load a new HTML page based upon the obvious (a button being clicked), or the unique (the editing of a cell in a spreadsheet).

A characteristic of Java applets is that they seldom consist of just one class file. On the contrary, a large applet may reference hundreds of class files. Making a request for each of these class files individually can tax any server and also tax the network capacity. However, packaging all of these class files into one file reduces the number of requests from hundreds to just one. This optimization is available in many Web browsers in the form of either a JAR file or a CAB file. Netscape and HotJava support JAR files simply by adding an `ARCHIVE="myjarfile.jar"` variable within the APPLET tag. Internet Explorer uses CAB files specified as an applet parameter within the APPLET tag. In all cases, executing an applet contained within a JAR/CAB file exhibits faster load times than individual class files. While Netscape and Internet Explorer use different APPLET tags to identify the packaged class files, a single HTML page containing both tags can be created to support both browsers. Each browser simply ignores the other's tag.

A disadvantage of using Java applets for UI generation is that the required version of Java must be supported by the Web browser. Thus, when using Java, the UI part of the application will dictate which browsers can be used for the client-side application. Note that the leading browsers support variants of the JDK 1.1 level of Java and they have different security models for signed applets.

A second disadvantage of Java applets is that any classes such as widgets and business logic that are not included as part of the Java support in the browser must be loaded from the Web server as they are needed. If these additional classes are large, the initialization of the applet may take from

seconds to minutes, depending upon the speed of the connection to the internet.

Because of the above shortcomings the use of Java applets is not recommended in environments where mixed levels and brands of browsers are present. Small applets may be used in rare cases where HTML UI elements are insufficient to express the semantics of the client-side Web application user interface. If it is absolutely necessary to use an applet, care should be taken to include UI elements that are core Java classes whenever possible.

6.2 Web application server

The Application Framework recommends the following technology model for a Web application server.

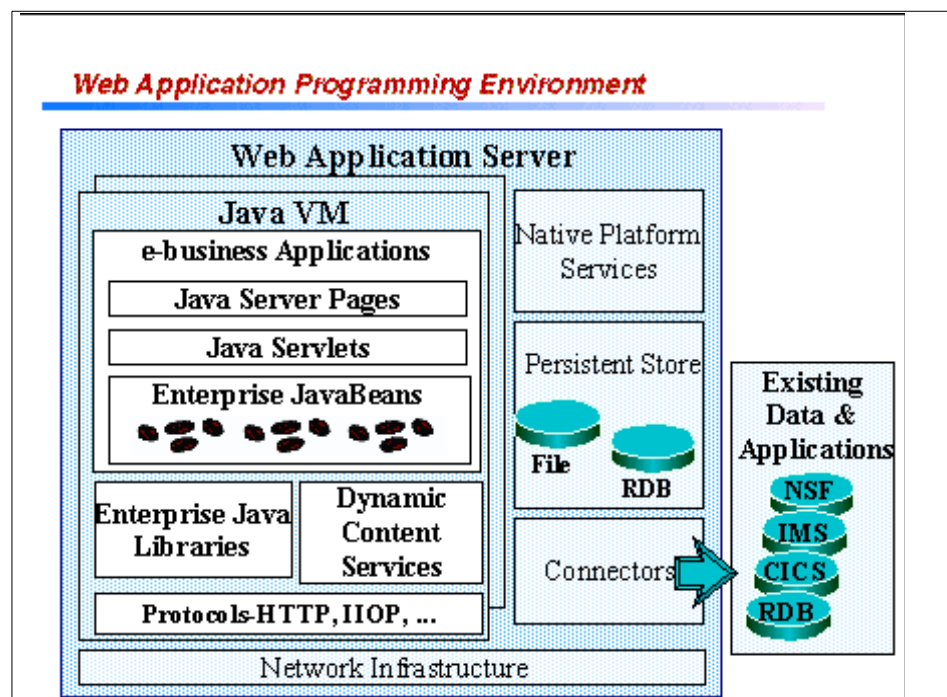


Figure 19. Web application server technology model

We will assume in this section that you will be using a Web application server and server-side Java. While there have been many other models for a Web application server, this is the one which is experiencing widespread industry

adoption. For more details on the Java APIs discussed in this section see *Java Enterprise in a Nutshell* by David Flanagan, Jim Farley, William Crawford and Kris Magnusson.

Before looking at the technologies and APIs available in the Web application programming environment, first a word about two fundamental operational components on this node, the HTTP server and the Java Virtual Machine (JVM). For production applications, these essential components should be chosen for their operational characteristics in areas such as robustness, performance and availability.

Later, in 7.3.1, “Model-View-Controller (MVC) design pattern” on page 94 we discuss the Model-View-Controller design structure so often used in user interfaces. For the Web application programming model:

- The View is generally best implemented using Java Server Pages.
- The Interaction Controller, which is primarily concerned with processing the HTTP request and invoking the correct business or UI logic, often lends itself to implementation as a servlet.
- The Model is represented to the View and Interaction Controller via a set of JavaBean components.

6.2.1 Java servlets

Servlets provide a replacement for CGI-based techniques in Web programming. Servlets are small Java programs that run on the Web application server. They interact with the servlet engine running on the Web application server through HTTP requests and responses, which are encapsulated as objects in the servlet.

One of the attractions of using servlets is that the API is a very accessible one for a Java programmer to master. The most current level of the servlet API is 2.1. To learn more about the servlet API visit:

<http://www.javasoft.com/products/servlet/>.

Servlets are a core technology in the Web application programming model. They are the recommended choice for implementing the “Interaction Controller” classes that handle HTTP requests received from the Web client.

For more details on the effective use of servlets see 7.2.1, “Java servlets” on page 86.

6.2.2 Java Server Pages (JSPs)

JSPs were designed to simplify the process of creating pages by separating Web presentation from Web content. In the page construction logic of a Web application, the response sent to the client is often a combination of template data and dynamically-generated data. In this situation, it is much easier to work with JSPs than to do everything with servlets.

The chief advantage JSPs have over Java servlets is that they are closer to the presentation medium. A Java Server *Page* is an HTML page. JSPs can contain all the HTML tags that Web authors are familiar with. A JSP may contain fragments of Java code which encapsulate the logic that generates the content for the page. These code fragments may call out to beans to access reusable components and back-end data. To learn more about JSPs visit <http://www.javasoft.com/products/jsp/>.

JSPs are compiled into servlets before being executed on the Web application server. The most current level of the JSP API is 1.0, which added significant changes from the 0.91 level.

JSPs are the recommended choice for implementing the “View” that is sent back to the Web client. For those cases where the code required on the page will be a large percentage of the page, and the HTML minimal, writing a Java servlet will make the Java code much easier to read and therefore maintain.

For more details on the effective use of JSPs see 7.2.2, “Java Server Pages (JSPs)” on page 90.

6.2.3 JavaBeans

JavaBeans is an architecture developed by Sun Microsystems, Inc. describing an API and a set of conventions for reusable, Java-based components. Code written to Sun’s JavaBeans architecture is called Java beans or just beans. One of the design criteria for the JavaBean API was support for builder tools that can compose solutions that incorporate beans. Beans may be visual or non-visual.

Beans are recommended for use in conjunction with servlets and JSPs in the following ways:

- As the client interface to the “Model Layer”. An “Interaction Controller” servlet will use this bean interface.
- As the client interface to other resources. In some cases this may be generated for you by a tool.

- As a component that incorporates a number of property-value pairs for use by other components or classes. For example, the Java Server Pages specification includes a set of tags for accessing JavaBean properties.

For more details on the effective use of beans see 7.2.3, “JavaBeans and Enterprise JavaBeans” on page 91.

6.2.4 XML

XML and XSL style sheets can be used on the server side to encode content streams and parse them for different clients, thus enabling you to develop applications for both a range of PC browsers and for the emerging pervasive devices. The content is in XML and an XML parser is used to transform it to output streams based on XSL style sheets.

This general capability is known as transcoding and is not limited to XML based technology. The appropriate design decision here is how much control over the content transforms you need in your application. You will want to consider when it is appropriate to use this dynamic content generation and when there are advantages to having servlets or JSPs specific to certain device types.

XML is also used as a means to specify the content of messages between servers, whether the two servers are within an enterprise or represent a business-to-business connection. The critical factor here is the agreement between parties on the message schema, which is specified as an XML DTD. An XML parser is used to extract specific content from the message stream. Your design will need to consider whether to use an event-based approach, for which the SAX API is appropriate, or to navigate the tree structure of the document using the DOM API.

For more detail on the use of XML in the server side of your Web applications, see *XML and Java: Developing Web Applications* by Maruyama, Hiroshi, Kent Tamura and Naohiko Uramoto.

6.2.5 JDBC and SQLJ

The business logic in a Web application will access information in a database for a database centric scenario. JDBC is a Java API for database-independent connectivity. It provides a straightforward way to map SQL types to Java types. With JDBC you can connect to your relational databases, and create and execute dynamic SQL statements in Java.

JDBC drivers are RDBMS specific, provided by the DBMS vendor, but implement the standard set of interfaces defined in the JDBC API. Given

common schemas between two databases, an application can be switched between one and the other by changing the JDBC driver name and URL. A common practice is to place the JDBC driver name and URL information in a property or configuration file.

There are four types of JDBC drivers from which you can choose, based on the characteristics of your application:

- Type 1: JDBC-ODBC bridge drivers. This type of driver, packaged with the JDK, requires an ODBC driver and was introduced to enable database access for Java developers in the absence of any other type of driver.
- Type 2: Native API Partly Java drivers. This type of driver uses the client API of the DBMS and requires the binaries for the database client software. This type of driver offers performance advantages but introduces native calls from the JVM.
- Type 3: Net-protocol All Java drivers. A generic network protocol is used with this type of driver. Portability is a major advantage of this type of driver, but it has the limitation that it requires intermediate middleware to convert the Net-protocol to the DBMS protocol.
- Type 4: Native-protocol All Java drivers. This type of driver is portable and uses the protocol of the DBMS. Type 3 and 4 drivers are well suited for applets that access a database server on an intranet, as they only require Java code to be downloaded.

An important technique used to enhance the scalability of Web applications is connection pooling, which may be provided by the application server. When application logic in a user session needs access to a database resource, rather than establishing and later dropping a new database connection, the code requests a connection from an established pool, returning it to the pool when no longer required.

SQLJ provides a simplified syntax for JDBC that allows you to write SQL-like statements directly in your Java source code. The SQLJ preprocessor generates static SQL providing better performance than dynamic SQL. SQLJ will also generate iterator Java classes. These iterators allow you to navigate query results using a very simple “get next” protocol.

JDBC is important to your design if you are implementing a solution based on application topology 1. As your design takes shape, based on its desired performance and sophistication you may see the need to investigate SQLJ or enterprise Java beans.

The most recent level of the JDBC specification is 2.0, but many JDBC drivers you use will still implement 1.0.

6.2.6 Enterprise JavaBeans

“Enterprise JavaBeans” is Sun's trademarked term for their EJB architecture (or “component model”). When writing to the EJB specification you are developing “enterprise beans” (or, if you prefer, “EJB beans”).

Enterprise JavaBeans are distinguished from Java beans in that they are designed to be installed on a server, and accessed remotely by a client. The EJB framework provides a standard for server-side components with transactional characteristics.

The EJB framework specifies clearly the responsibilities of the EJB developer and the EJB container provider. The intent is that the “plumbing” required to implement transactions or database access can be implemented by the EJB container. The EJB developer specifies the required transactional and security characteristics of an EJB in a deployment descriptor (this is sometimes referred to as declarative programming). In a separate step, the EJB is then deployed to the EJB container provided by the application server vendor of your choice.

There are two types of Enterprise JavaBeans:

- Session
- Entity

A typical session bean has the following characteristics:

- Executes on behalf of a single client.
- Can be transactional.
- Can update data in an underlying database.
- Is relatively short lived.
- Is destroyed when the EJB server is stopped. The client has to establish a new session bean to continue computation.
- Does not represent persistent data that should be stored in a database.
- Provides a scalable runtime environment to execute a large number of session beans concurrently.

A typical entity bean has the following characteristics:

- Represents data in a database.
- Can be transactional.
- Shared access from multiple users.
- Can be long lived (lives as long as the data in the database).

- Survives restarts of the EJB server. A restart is transparent to the client.
- Provides a scalable runtime environment for a large number of concurrently active entity objects.

Typically an entity bean is used for information that has to survive system restarts, while in session beans, the data is transient and does not survive when the client's browser is closed. For example, a shopping cart containing information that may be discarded uses a session bean, and an invoice issued after the purchase of the items is an entity bean.

An important design choice when implementing entity beans is whether to use Bean Managed Persistence (BMP), in which case you must code the JDBC logic, or Container Managed Persistence (CMP), where the database access logic is handled by the EJB container.

The business logic of a Web application often accesses data in a database. EJB entity beans are a convenient way to wrap the relational database layer in an object layer, hiding the complexity of database access. Because a single business task may involve accessing several tables in a database, modeling rows in those tables with entity beans makes it easier for your application logic to manipulate the data.

The latest EJB specification is 1.1. The most significant changes from EJB 1.0 are the use of XML-based deployment descriptors and the need for vendors to implement entity bean support to claim EJB compliance.

To learn more about Enterprise JavaBeans visit:

<http://www.javasoft.com/products/ejb/index.html>

6.2.7 Connectors

e-business connectors are gateway products that enable you to access enterprise and legacy applications and data from your Web application. Connector products provide Java interfaces for accessing database, data communications, messaging and distributed file system services.

IBM provides a significant set of e-business connectors with tool support, for CICS, Encina, IMS, MQSeries, DB2, SAP and Domino. IBM is basing its tool support on a Common Connector Framework (CCF). For resources on System 390, IBM is delivering native connectors based on CCF.

The command bean model allows you to code to the specific connector interface(s) of your choice while hiding the connector logic from the rest of the Web application. Command beans are discussed in 7.6.1, "Command beans" on page 118.

Connectors are particularly important if you are implementing application topology 2 in your solution.

6.2.8 Additional enterprise Java APIs

In developing a server-side application, you may also need to be familiar with the following enterprise Java class libraries:

- Java Naming and Directory Interface (JNDI). This package provides a common API to a directory service. Service provider implementations include those for LDAP directories, RMI and CORBA object registries. Sample uses of JNDI include:
 - Accessing a user profile from an LDAP directory
 - Locating and accessing an EJB Home
- Remote Method Invocation (RMI). RMI and RMI over IIOP are part of the EJB specification as the access method for clients accessing EJB services. RMI can also be used to implement limited function Java servers.
- Java Message Service (JMS). The JMS API enables a Java programmer to access message-oriented middleware such as MQSeries from the Java programming model. Such messaging middleware is a popular choice for accessing existing enterprise systems and is one of your options if you are implementing a solution based on application topology 2.
- Java Transaction API (JTA). This Java API for working with transaction services, is based on the XA standard. With the availability of EJB servers, you are less likely to use this API directly.

6.3 Where to find more information

For more information on topics discussed in this chapter see:

- *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00
- Flanagan, David, *JavaScript: The Definitive Guide*, Third Edition, O'Reilly & Associates, Inc., 1998
- Maruyama, Hiroshi, Kent Tamura and Naohiko Uramoto, *XML and Java: Developing Web Applications*, Addison-Wesley 1999
- Flanagan, David, Jim Farley, William Crawford and Kris Magnusson, *Java Enterprise in a Nutshell*, O'Reilly & Associates, Inc., 1999
- For information on the IBM Application Framework for e-business:
<http://www.ibm.com/software/ebusiness/>

- For information about the ECMAScript language specification:
<http://www.ecma.ch/stand/ECMA-262.htm>
- To learn more about Java technology:
<http://www.javasoft.com/products>

Chapter 7. Application design guidelines

e-business application design presents some unique challenges compared to traditional application design and development. The majority of these challenges are related to the fact that traditional applications were primarily used by a defined set of internal users, whereas e-business applications are used by a broad set of internal and external users such as employees, customers, and partners. Web applications must be developed to meet the varied needs of these end users. The following list provides key issues to consider when designing e-business applications:

- The user experience, look and feel of the site need to be constantly enhanced to leverage emerging technologies, attract and retain site users.
- New features have to be constantly added to the site to meet customer demands.
- Such changes and enhancements will have to be delivered at record speed to avoid losing customers to the competition.
- e-business applications in essence represent the corporate brand online. Developers have to work closely with the marketing department to ensure the digital brand effectively represents the company image. Such intra-group interactions usually present content management challenges.
- It is hard to predict the runtime load of e-business applications. Based on the marketing of the site, the load can increase dramatically over time. If the load increases, the design must allow such applications to be deployed in various high volume configurations. Runtime configurations are discussed in Chapter 3 “Choosing the runtime topology” on page 19 and Chapter 4 “Product mapping” on page 43. It is important to be able to move Web applications between these runtime configurations without making significant changes to the code.
- Security requirements are significantly higher for e-business applications compared to traditional applications. In order to execute traditional applications from the Web a special set of security-related software may be needed to access private networks.
- The emergence of the Personal Digital Assistant (PDA) market and broad-band Internet market will require the same information to be presented in various user interface formats. PDAs will require a light-weight presentation style to accommodate the low network band width. Broad-band users on the other hand will demand a highly interactive rich graphical user interface.

In order to meet these challenges it is critical to design Web applications to be flexible. This chapter helps you understand some of these design challenges and presents various design pattern solutions that promote loosely coupled design to provide a maximum degree of flexibility in a Web application.

Problem domain: In addition to presenting design pattern solutions, we demonstrate how these techniques can be applied by a sample. In this sample we use a simple problem domain where the Web application is responsible for displaying the average weather information of different planets in the solar system on a specified date. This problem domain is a simplified version of the problem domain implemented by the Pattern Development Kit (PDK). Further details on the PDK can be found at:

<http://www.ibm.com/developer/patterns>

For each of these examples we present and explain class diagrams, object interaction diagrams and code snippets. Class diagrams and object interaction diagrams use the standard UML notation and were created using the Rational Rose modeling tool. Chapter 8 “Application development guidelines” on page 147 explains how such UML work products can be developed and used throughout the application design and development process.

7.1 Application elements

The design guidelines outlined here primarily focus on User-to-Business Web applications. Before exploring these guidelines it is important to understand the overall structure of these types of Web applications. Chapter 3 “Choosing the runtime topology” on page 19 presents the overall topology of such e-business applications and explains the responsibilities of various nodes in the topology. Chapter 6 “Technology options” on page 65 presents various technology options available for implementing a User-to-Business Web application and recommends the use of server-centric Java-based technologies such as servlets, JSPs, JavaBeans, and EJBs for such implementations. Figure 20 identifies the key elements of such Web applications and explains the responsibilities of these elements.

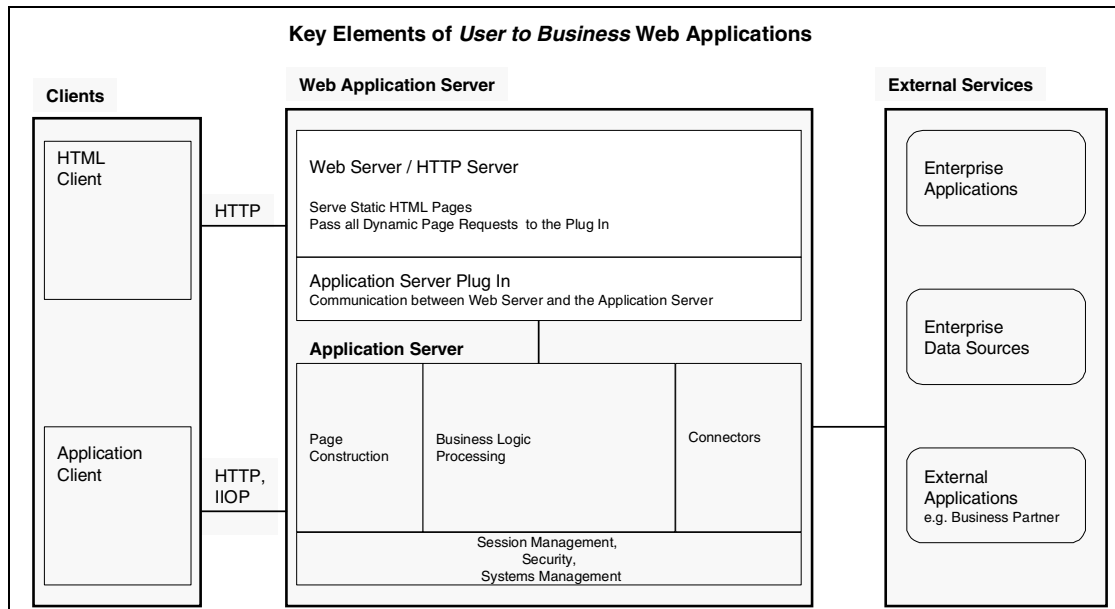


Figure 20. Key elements of User-to-Business Web applications

Clients are responsible for accepting and validating the user input, communicating the user inputs to the Web application server, and presenting the results received from the Web application server to the user. Clients may use HTTP, IIOP, TCP/IP, or other Internet standard protocols to communicate with the Web application server. These clients can be broadly classified into the following categories:

- HTML clients use HTTP protocol to communicate with the Web application server. These clients display HTML and DHTML Web pages. In addition, they are capable of processing client-side JavaScript for enhancing navigation to perform simple input validation and to handle simple errors. Furthermore, the majority of HTML clients can display small Java applets to enhance the GUI.
- Application clients are primarily large Java applets or Java applications. These clients provide rich graphical user interfaces compared to HTML clients. They may communicate with the Web application server over a number of protocols including HTTP, IIOP, MQ, etc. Application clients communicate with the Web application server primarily to receive data rather than pre-formatted HTML pages. These clients use the data received to format and render the user interface. All of the user interface

processing is performed on the client side. In addition, under this model, some parts of the business logic can also be processed on the client side.

WebSphere Application Server supports both of these client models. However, HTML clients provide the following benefits compared to application clients:

- The majority of the presentation logic and all of the business logic will reside on the server. Hence it is easy to make the necessary changes to support a broad range of client devices including Personal Digital Assistants (PDAs), WebTV, etc.
- The client part of the application is lightweight and downloads quickly.
- It is easier to secure, scale and maintain presentation and business logic that reside on the server.
- Client applications that use Java applets and Java applications require a particular version of Java to be supported by all clients, thus limiting the universal accessibility of the applications.
- Client applications that use Java applets and Java applications are downloaded to a user's local machine and can be de-compiled to view the business logic. This poses security concerns. Hence critical business logic should not be directly coded in these downloadable applications.

For these reasons, the majority of Web applications being designed today are zero maintenance HTML clients. The guidelines outlined in this chapter primarily focus on designing and developing HTML client Web applications. Further details on these client models can be found at:

<http://www.ibm.com/developer/features/framework/framework.html>

Web application servers are the focal point of the Web application topology. Their responsibilities include: receiving requests from the clients, selecting and executing the appropriate business logic based on these requests, coordinating with *External Services* to retrieve data and execute external applications, and finally formulating the response and dispatching it back to the client. To meet these requirements, Web application servers provide a range of dynamic page construction, business logic processing, data access, external application integration, session management, load balancing, and fail-over services. At a high level, one can identify the following sub-components in the majority of Web application servers including WebSphere Application Server:

- Web Server/HTTP Server - are responsible for receiving and responding to HTTP requests. They are capable of serving static HTML pages without help from other sub-components of the Web application server. However,

they pass all dynamic page requests to the application server plug-in. In the case of WebSphere Application Server, the HTTP server is usually configured to pass all requests for servlet and JSP execution to the WebSphere plug-in.

- Application server plug-in provides the connection between the HTTP server and the page construction services of the Web application server.
- Page Construction services - WebSphere Application Server supports Java-based technologies such as Java servlets and Java Server Pages (JSPs) for dynamic page construction.
- Business logic services provide a robust environment for processing business logic independent of the user interface client types. WebSphere Application Server supports components based on technologies such as JavaBeans and Enterprise JavaBeans (EJBs) for programming business logic.
- Connectors are components that support the communication between the application server and the external services such as databases, legacy applications and business partner applications. WebSphere Application Server Advanced Edition provides a number of connectors including JDBC drivers, JNDI class libraries, CICS connectors, MQ connectors, and IMS connectors.

Further details of these connectors, class libraries, and related tools can be found at:

<http://www.ibm.com/developer/features/framework/framework.html>.

The PDK provides some examples of using these connectors.

- Session management services are necessary because inherently HTTP is a stateless protocol. In order to address the issues of a stateless protocol, a number of HTTP session management techniques have been developed. WebSphere Application Server supports a number of these techniques and provides a simple HttpSession API to handle the session information. This API is based on the standard Java Servlet API.
- Security services include authentication, authorization, data integrity, privacy (encryption) and non-repudiation services. WebSphere Application Server provides these services by supporting industry standard protocols such as SSL, LDAP, etc.
- System management services provide a robust runtime environment for the application hosted in the Web application server. These services allow load balancing, fail-over support, remote monitoring, etc. WebSphere Application Server supports these features and further details can be

found in Chapter 9 “System management products and guidelines” on page 197.

- **External Services** include enterprise data sources, existing or new enterprise applications (for example ERPs, financial systems, etc.), and business partner systems.

7.2 Understanding supporting technologies

Before presenting some of the design pattern solutions, it is important to establish a common understanding of the key technologies involved. Chapter 6 “Technology options” on page 65 discusses various technology options available for Web application development and recommends the use of HTML, JavaScripts, Java servlets, JSPs, JavaBeans, and EJBs for implementing user-to-business applications. This section introduces you to server-side Java technologies, namely servlets, JSPs, JavaBeans, and EJBs. However in order to understand the design guidelines you are expected to be familiar with HTML and client-side JavaScripts. This is because dynamically generated Web pages contain client side JavaScripts that perform page navigation and simple edits. A good introduction to HTML can be found at <http://www.w3.org/MarkUp/> and to JavaScripts found at <http://www.ecma.ch/stand/ECMA-262.htm>.

7.2.1 Java servlets

Servlets are protocol and platform independent server-side Java components. They implement a simple request and response framework for communication between the client and the server. The Java servlet API is a set of Java classes that define a standard interface between Web clients and the Web application server. The API is composed of the following two packages:

- `javax.servlet`
- `javax.servlet.http`

The `javax.servlet` package implements the generic protocol-independent servlets. The `javax.servlet.http` package extends this generic functionality to include specific support for the HTTP protocol. In this section we explore the key classes and methods of the `javax.servlet.http` package. The subsequent sections introduce other classes and methods of this package if such services are exploited by the topic under consideration.

HttpServlet is an abstract class that provides methods for handling various HTTP requests. Typically, all servlets that respond to HTTP traffic would

extend this class and override some of the methods such as `doGet()` to handle GET requests and `doPost()` to handle POST requests. WebSphere Application Server provides ways to register these servlets with the Web server. Upon receiving an HTTP request the Web server determines if it needs to be handled by a servlet and if so, it passes such requests to WebSphere. WebSphere in turn calls the `service()` method which then calls the HTTP-specific method based on the type of HTTP request. The following are some of the key methods provided by the `HttpServlet` class:

- `init()` - The purpose of this method is to perform necessary servlet initialization. It is guaranteed to be the first method to be called on any servlet instance. The servlet implementer may choose to override this method to perform custom servlet initialization.
- `service(HttpServletRequest req, HttpServletResponse resp)` - The Web application server invokes the `servlet.service()` method upon receiving an HTTP request targeted towards that servlet. This method in turn invokes the appropriate HTTP-specific method based on the type of request. `HttpServletRequest` is an input parameter and contains the HTTP protocol- specified header information. `HttpServletResponse` is an output parameter and contains an HTTP protocol-specific header and returns data to the client. Further details on these classes can be found below.
- `doGet(HttpServletRequest req, HttpServletResponse resp)` - The `service()` method invokes the `doGet()` method if the HTTP request type is GET. The servlet implementer overrides the `doGet()` method if the servlet is intended to handle GET requests. HTTP GET requests are expected to be safe and read-only. They are suitable for queries.
- `doPost(HttpServletRequest req, HttpServletResponse resp)` - The `service()` method invokes the `doPost()` method if the HTTP request type is POST. The servlet implementer would override the `doPOST()` method if the servlet is intended to handle POST requests. HTTP POST requests are not expected to be either safe or read-only. They are suitable for requests that result in updates to stored data.
- `destroy()` - This method is called just before unloading a servlet. It is overridden only if there is a need to perform some cleanup operations such as closing connections or files before unloading a servlet.
- `getServletContext()` - This method returns a *ServletContext* object that contains information about the environment in which the servlet is executing. This method is particularly useful in dispatching control from a servlet to a JSP. Developers are never expected to override this method.

HttpServletRequest represents a communication channel from the client and is passed as an input parameter into the `HttpServlet.service()` method,

which in turn passes it to the appropriate HTTP-specific methods such as `doGet()` and `doPost()`. The servlet can invoke this object's methods to get information about the client environment, the server environment, and any HTTP protocol-specified header information that is received from the client. The following are some of the key methods provided by this interface:

- `getParameter(java.lang.String name)` - returns a string containing the value of the specified parameter.
- `getParameterValues(java.lang.String name)`, on the other hand, returns the parameter value as an array of strings. These methods can be used to retrieve the HTML FORM information set by GET and POST methods.
- `getSession(boolean create)` - returns the current valid `HttpSession` associated with this request. If `create` is true and a current valid `HttpSession` does not exist then a new session is created.
- `setAttribute(java.lang.String key, java.lang.Object o)` - is used to store an attribute in the request context. Attributes are stored as name-value pairs in a hash table and can be retrieved by the `getAttribute()` method. This method may be used to pass data between various servlets and JSPs. It is important to note that the attributes are reset between requests.

HttpServletResponse represents a communication channel back to the client and is passed as an output parameter into the `HttpServlet.service()` method. It provides a number of set methods that allow servlets to manipulate HTTP protocol-specified header information and to set the response data to be returned to the client. The following are some of the key methods provided by this interface:

- `getWriter()` - returns a print writer object. The print writer object is an output stream to which servlets write dynamically generated text such as HTML and XML. Upon completion all the text that is written to the print writer will be sent to the client, for example sent to the browser to be displayed.
- `setContentType(java.lang.string type)` - sets the MIME content type to be sent back to the client, for example "text/html". The content type must be set before obtaining the print writer or writing to the output stream.
- `sendRedirect(java.lang.string type)` - sends a temporary redirect response to the client using the specified redirect location URL. The URL must be absolute (for example, `https://hostname/path/file.html`). Relative URLs are not permitted here.

HttpSession represents an association between an HTTP client and an HTTP server. By design, HTTP is a stateless protocol. Over the years, a number of approaches have been developed to maintain application sessions

between HTTP requests. HttpSession are used to maintain application state and user identity across several page requests from the same user. The following are some of the key methods provided by this interface:

- putValue(java.lang.String name, java.lang.Object value) - can be used to store application-specific state data as a name-value pair. The application server is responsible for storing this information in a hash table and allows servlets to access this data across HTTP requests.
- getValue(java.lang.String name) - is used to retrieve the application-specific value from the hash table by referencing the name-value pair by its name.
- removeValue(java.lang.String name) - is used to remove the name-value pair from the hash table. In order to manage scarce server resources, it is important to remove state information that is no longer required.

ServletContext object contains information about the environment in which the servlet is executing. A servlet can obtain its context by calling the `getServletContext()` method. The following is the key method provided by this interface:

- getRequestDispatcher(java.lang.String urlpath) - takes the URL path of resources such as other servlets and JSPs as input and returns a RequestDispatcher object that implements a wrapper around a server resource. RequestDispatcher is responsible for locating such resources and also forwarding any requests made by the servlet to the appropriate resource. As shown below we use them to forward requests from the servlets to JSPs.

```
RequestDispatcher rd;  
rd = getServletContext().getRequestDispatcher("Sample.JSP");  
rd.forward(req, res);
```

In this example, the servlet retrieves the ServletContext. Using this context, it obtains the RequestDispatcher to a JSP. Finally it forwards the request to the JSP by calling the forward() method on the RequestDispatcher. This invokes the service() method on the target JSP.

WebSphere Application Server V3 supports the Java servlets API 2.1. Further details of this API can be found at:

<http://java.sun.com/products/servlet/2.1>.

WebSphere Studio and VisualAge for Java Servlet and JSP Programming, SG24-5755-00 provides further details and examples of servlet programming.

7.2.2 Java Server Pages (JSPs)

JSPs provide a simple yet powerful mechanism for inserting dynamic content into Web pages. The JSP specification achieves this by defining a number of HTML-like tags that allow developers to insert server-side Java logic directly into HTML or XML pages that are sent to HTTP clients.

It is important to note that JSP technology is an extension of the Java Servlet API. In fact, application servers compile JSPs into `HttpServlet`s before executing them. Essentially, the JSPs represent the `service()` method of the automatically generated `HttpServlet`. Hence JSPs automatically get access to certain implicit objects without having to declare them. The following table lists some of the key implicit objects:

Table 2. JSP implicit objects

Object Name	Object Type	Description
request	<code>javax.servlet.HttpServletRequest</code>	Represents the HTTP request received from the client.
response	<code>javax.servlet.HttpServletResponse</code>	Represents the HTTP response to be sent to the client.
session	<code>avax.servlet.HttpSession</code>	Represents the session object created for the requesting client (if any).
out	<code>javax.servlet.jsp.JspWriter</code>	Represents the output stream to be sent to the client as part of the response.

For the purposes of this chapter we focus on the following JSP Specification 1.0 tags:

- JSP scriptlet

Syntax: `<% code %>` .

Scriptlets can be used to insert any code fragment in the specified scripting language into a JSP. By default, the scripting language is assumed to be Java.

- JSP expression

Syntax: `<%= expression %>`.

The expression here stands for any valid operation that can be executed at runtime. The output from such an operation is converted to into a string and is emitted into the output stream *out*.

- **jsp:useBean**

Syntax: `<jsp: useBean id="beanInstanceName"
scope="page|request|session|application" typespec/>`

The server uses `id` and `scope` to look for the bean. If it exists it is accessed. If not, it is created. The `typespec` is used to specify the bean type.

The `jsp:useBean` tag is used to declare a JavaBean you would like to use within a JSP.

- **jsp:getProperty**

Syntax: `<jsp:getProperty name="beanName" property="propertyName" />`

The `<jsp:getProperty>` converts the value of the specified property into a string and inserts this string into the implicit `out` object. `getProperty` on a bean is usually called after declaring the bean instance using the `jsp:useBean` tag.

- **jsp:setProperty**

Syntax: `<jsp:setProperty name="beanName" property="propertyName"
value="propertyValue" />`

`<jsp:setProperty>` is used to set the bean property value.

WebSphere Application Server V3 supports JSP Specification 1.0 and JSP Draft Specification 0.91. You have to choose a particular specification level for a JSP page. We recommend using JSP Specification 1.0. Further details on these specifications can be found at <http://www.java.sun.com/products/jsp/>. In addition, WebSphere supports WebSphere specific tags that allow you to connect to databases and perform repeats, etc. Details on the WebSphere extension to the JSP specification can be found in the WebSphere Library at:

<http://www.ibm.com/software/webservers/appserv/library.html>

7.2.3 JavaBeans and Enterprise JavaBeans

The JavaBeans architecture defines reusable software components that can be manipulated visually by builder tools. Code written to this architecture, called Java beans, or beans, are specialized Java classes that support the following features: properties, methods, events, introspection, and persistence. Basically, properties are attributes that have set and get methods. Methods are simple Java methods that can be called from other components. Events define a framework for one component to notify the other when something noteworthy happens. The JavaBean specification defines a set of conventions for defining properties, methods, and events. Using this convention, builder tools can analyze the bean to allow for visual

manipulation. In addition, beans should implement the persistence mechanism allowing the customized JavaBean state to be stored and retrieved when necessary. Beans can be either visual or non-visual; however they should all allow manipulation by visual builder tools. This is usually done by using Java introspection techniques. Further details on the JavaBean Specification can be found at:

<http://java.sun.com/beans/index.htm>

The Enterprise JavaBeans (EJBs) architecture extends the reusable software component model of JavaBeans to multi-tier, distributed, transactional computing environments. For the sake of simplicity the design examples outlined in this chapter primarily use simple beans. However, these guidelines can be easily extended to Web applications that use EJBs instead of simple Java beans. WebSphere Application Server V3 provides full support for the Enterprise JavaBeans™ (EJB) 1.0 specification. Further details on the EJB Specification can be found at:

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

The previous two sections have established a common set of terminology and a common understanding of the key technologies involved in a User-to-Business Web application. Based on this understanding, the subsequent sections of this chapter will:

- Introduce a design challenge.
- Discuss the recommended design pattern solution.
- Explain the motivation for using the recommended design pattern solution.
- Document the implications of the chosen approach on the runtime topology.
- Provide an example that applies the recommended design technique using class diagrams, interaction diagrams, and sample code.
- Document the advantages and disadvantages of the proposed solution.

7.3 Application structure

A User-to-Business Web application can be viewed as a set of interactions between the browser and the Web application server. The interaction begins with an initial request by the browser for the welcome page of the application. This is usually done by the user typing in the welcome page URL on the browser. All subsequent interactions are initiated by the user either by clicking on a button or a link. This causes a request to be sent to the Web application

server. The Web application server processes the request and dynamically generates a result page and sends it back to the client along with a set of buttons and links for the next request.

A closer examination of these interactions reveals a common set of processing requirements that need to be considered on the server side. These interactions can be easily mapped to the classical Model-View-Controller design pattern as shown below. As outlined in the book *Design Patterns: Elements of Reusable Object-Oriented Software* the relationship between the MVC triad classes are composed of Observer, Composite, and Strategy design patterns. For further details on these detailed design patterns please refer to the above mentioned design patterns book.

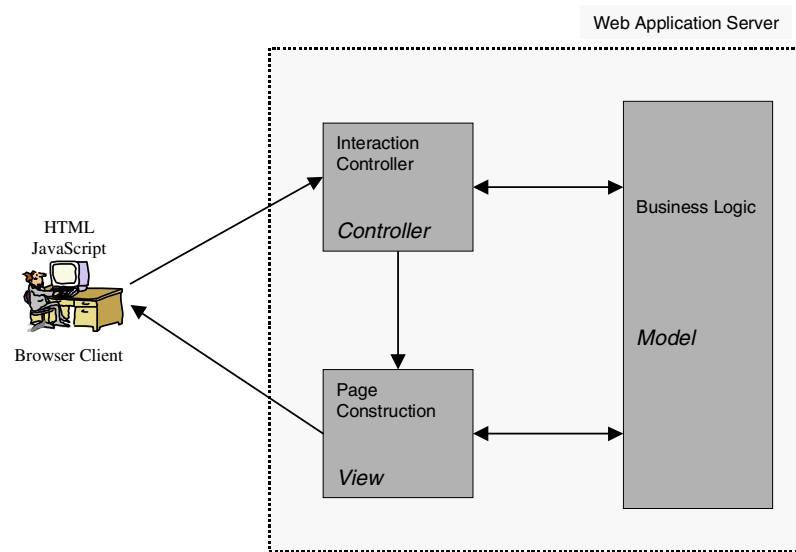


Figure 21. The Structure of user-to-business Web applications

Model represents the application object that implements the application data and business logic. The *View* is responsible for formatting the application results and dynamic page construction. The *Controller* is responsible for receiving the client request, invoking the appropriate business logic, and based on the results, selecting the appropriate view to be presented to the user. A number of different types of skills and tools are required to implement various parts of a Web application. For example the skills and tools required to design an HTML page are vastly different from the skills and tools required to design and develop the business logic part of the application. In order to

effectively leverage these scarce resources and to promote reuse we recommend structuring Web applications to follow the Model-View-Controller design pattern.

7.3.1 Model-View-Controller (MVC) design pattern

Over the years, a number of GUI-based client/server applications have been designed using the MVC design pattern. This powerful and well-tested design pattern can be extended to support the user-to-business Web applications as shown by Figure 21 on page 93. Throughout this chapter, *Model* is often referred to as Business Logic, *View* is referred to as Page Constructor or Display Page, *Controller* is referred to as Interaction Controller. This section further outlines the responsibilities of each of these components and discusses what technologies could be used to implement the same.

Interaction Controller (*Controller*) - The easiest way to think about the responsibility of the interaction controller is that it is the piece of code that ties protocol independent business logic to a Web application. This means that the interaction controller's primary responsibility is mapping HTTP protocol-specific input into the input required by the protocol-independent business logic (that might be used by several different types of applications), scripting the elements of business logic together and then delegating to a page construction component that will create the response page to be returned to the client. Here's a list of typical functions performed by the interaction controller:

- Validate the request and session parameters used by the interaction.
- Verify that the client has the necessary privileges to access the requested business task.
- Transaction demarcation.
- Invoke business logic components to perform the required tasks. This includes mapping the request and session parameters to the business logic component's input properties, using the output of the components to control logic flow and correctly chain the business logic.
- Call the appropriate page construction component based on the output of one or more of the business logic commands.

Interaction controllers can be implemented using either Java servlets or JSPs. It is important to note that interaction controller code is primarily Java code and Java code is easy to develop and maintain using Java Integrated Development Environments (IDE) such as VisualAge for Java. Since servlets are also Java classes it is possible to leverage such IDEs to write, compile, and maintain servlets. JSPs on the other hand provide a simple script-like

environment. Even though JSPs were primarily developed for dynamic page construction, they can be used to code interaction controller logic. Due to their simplicity, JSPs have a broad appeal to script programmers. However tools available today for JSPs are primarily targeted toward dynamic page construction.

Finally, it is up to the project team to decide whether to use JSPs, servlets or both for coding interaction controller logic. What is much more important is to recognize the need for the separation between interaction controller logic and page construction logic. Such a separation is necessary under the following conditions:

- One display page needs to be reusable because it can be called by multiple interaction controllers. For example an error page may be called by more than one interaction controller. Under such a scenario if we were to combine the error page construction logic and the interaction controller logic then the error page logic would need to be duplicated in several places throughout the application.

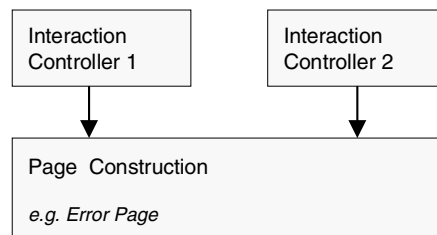


Figure 22. One display page component being called by multiple interaction controllers

- The interaction controller is required to do page selection. There are several reasons for this, such as the need to include different display pages depending on runtime results, national language support, different client browser types, different customer types, etc. For example, the figure below shows an interaction controller calling either the admin or normal page constructor, based on the used type.

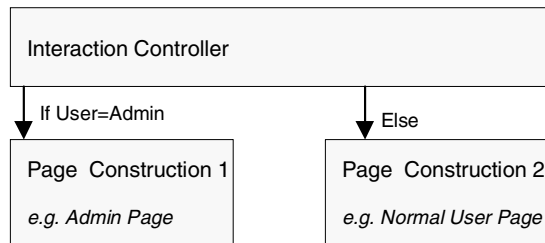


Figure 23. One interaction controller calling multiple page constructors

- If there is a need to use different tools and skills for coding interaction controllers and display pages then it is good to separate the two to simplify the development process. Failing to do so could result in multiple people having to write different parts of the same file thus complicating the version control and code management process.

Note

Hence it is recommended that servlets should be used in most cases to implement interaction controllers. However, for simple applications where there is no conditional or transactional logic involved, it is possible to combine the interaction controller and page construction logic into one component. Under such conditions, a JSP would be the best choice.

Another common design issue to consider is the relationship between interactions and interaction controllers. The following is an overview of the options:

- One interaction to one interaction controller: For every unique interaction, there is a unique interaction controller. For example, login is handled by loginServlet, logoff is handled by logffServlet.
- One interaction group to one interaction controller: A group of related Web interactions are all handled by the same interaction controller. The interaction controller for the group is passed a parameter to differentiate which interaction within the group is being performed. For example, login and logoff both could be handled by authenticateServlet which can get a parameter called action type that could be either set to login or logoff.
- All interactions to one interaction controller: This approach extends the interaction group to all interactions and builds a monolithic interaction controller.

We recommend the first option in most cases. Occasionally it might be appropriate to choose the second option and implement one interaction controller that supports a group of related interactions.

Page Constructor (*View*) - The page constructor is responsible for generating the HTML page that will be returned to the client; it is the view component of the application. Like interaction controllers, WebSphere allows display pages to be implemented as either Java servlets or Java Server Pages. JSPs allow template pages to be developed directly in HTML, with scripting logic inserted for dynamic elements of the page. Hence, JSP is the best choice for implementing page construction components. In addition, there are number of visual tools such as WebSphere Page Designer that can be used to develop dynamic display pages using JSP technology.

In many cases, the interaction controller will pass the dynamic data as JavaBeans to the display page for formatting. In other cases, the display page will invoke business logic directly to obtain dynamic data. It makes sense to have the interaction controller pass the data when it has already obtained it and when the data is an essential component of the contract between the interaction controller and the display page. In other cases, the data needed for display is not an essential part of the interaction and can be obtained independently by inserting calls to business logic directly in the display page. However such direct access to business logic from the page construction component increases the complexity of the display page, since the page designer must know the details of the business logic methods. For this reason, care must be taken to minimize such direct access to business logic from the display pages.

Once the page constructor has obtained the dynamic data, either from the interaction controller or via its own logic, it will typically format the data. This can be done in two ways. The simplest mechanism is to format the data using simple scripting inside the page constructor. An alternative is to develop reusable formatting components called Formatter beans that will take a data set and return formatted HTML. 7.5, “Application output formatting” on page 115 further elaborates this concept.

Business Logic (*Model*) - The business logic part of a Web application is the piece of code ultimately responsible for satisfying client requests. As a result, business logic must address a wide range of potential requirements which include ensuring transactional integrity of application components, maintaining and quickly accessing application data, supporting the coordination of business workflow processes, and integrating new application components with existing applications. To address these requirements, WebSphere supports business logic written in and using the full facilities of

the Java runtime including support for Java servlets, Java beans, EJBs, JDBC, CORBA, LDAP, MQ, and connectors to CICS, IMS and other enterprise services.

While a discussion on how business logic should be developed is beyond the scope of this chapter, it is valuable to consider the interface between the Web parts of the interaction (interaction controllers and the display pages) and the business logic. We recommend that the business logic be wrapped with JavaBeans or EJBs. Such a separation of business logic from the Web-specific interaction controller and display page logic isolates the business logic from the details of Web programming, increasing the reusability of the business logic in both Web and non-Web applications. Further details on how such a wrapping can be done can be found in 7.6, “Application business logic granularity” on page 117.

7.3.2 MVC design pattern example

Let us assume that we are interested in designing a simple Web application that accepts a date from the user and displays the average temperature of the planet Mars on that day. Figure 24 on page 99 identifies the key classes necessary to implement the application using the MVC design pattern. They are:

- **RetrievePlanetTemperatureServlet**: represents the interaction controller. This class extends the `HttpServlet` and implements the `doGet()` method.
- **PlanetBean**: represents the business logic. It is a simple Java bean that has a property called `temperature` and implements the `getTemperature(date)` method. This method takes a date as a parameter and returns the average temperature of the planet on the specified date. In a real implementation the `getTemperature(date)` method may have to read the temperature data from a database or obtain it from some other application. In order to hide the complexity we don't implement these details in this example.
- **PlanetTemperatureJSP**: represents the display page. It is a JSP that constructs the response page to display the temperature of the planet on the specified date.
- **HttpServletRequest**: represents the request received from the browser with the HTTP-specified data. It is passed into the `doGet()` method of the servlet as an input parameter.

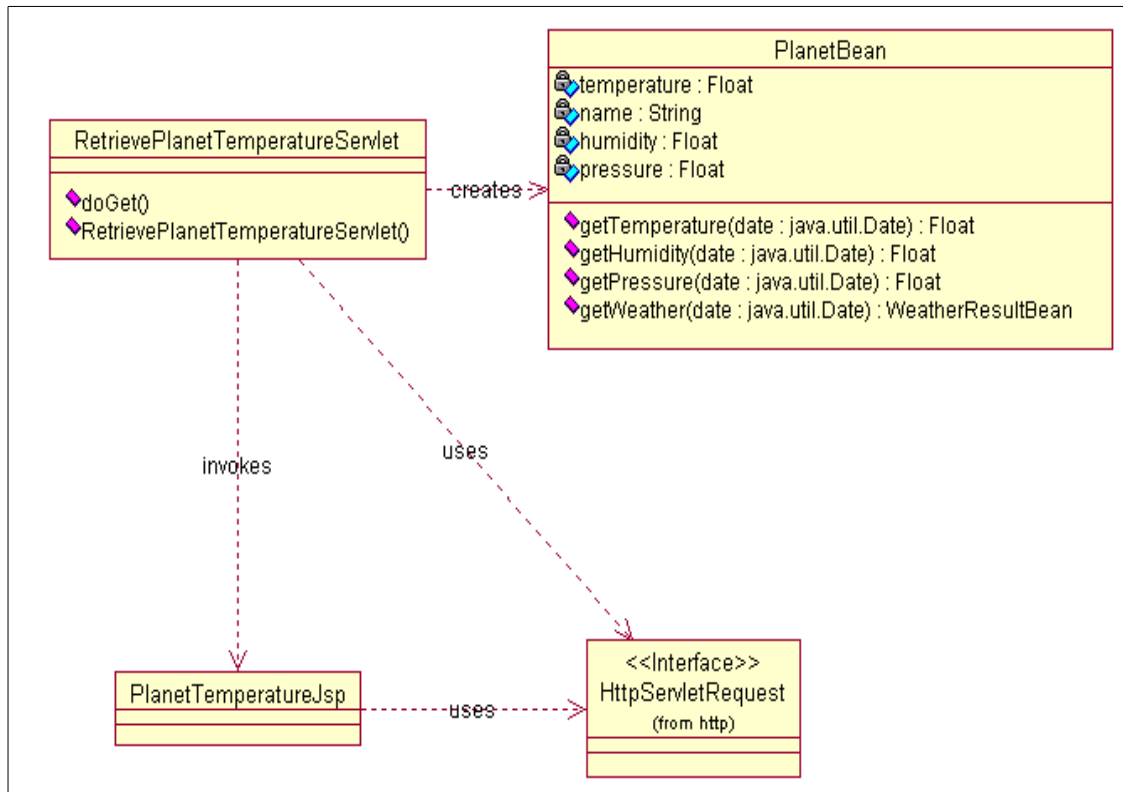


Figure 24. Class diagram - Model-View-Controller design pattern example

Let us assume that the interaction begins by the user requesting a static HTML page that allows the user to enter the date for which he or she would like to get the average temperature of Mars. The following HTML code shows a typical HTML form that accomplishes this activity:

```

<FORM METHOD="GET" ACTION="RetrievePlanetTemperatureServlet">
Date: <INPUT NAME="Date" TYPE="TEXT" SIZE=12> <BR>
<INPUT TYPE="SUBMIT">
</FORM>

```

When the user clicks on the submit button this form sends an HTTP GET request to the Web server. The Web server recognizes that the call is targeted towards a servlet and hands the request over to the WebSphere Application Server. WebSphere in turn calls the `doGet()` method on the

RetrievePlanetTemperatureServlet. The following object interaction diagram shows the subsequent key interactions between the various components.

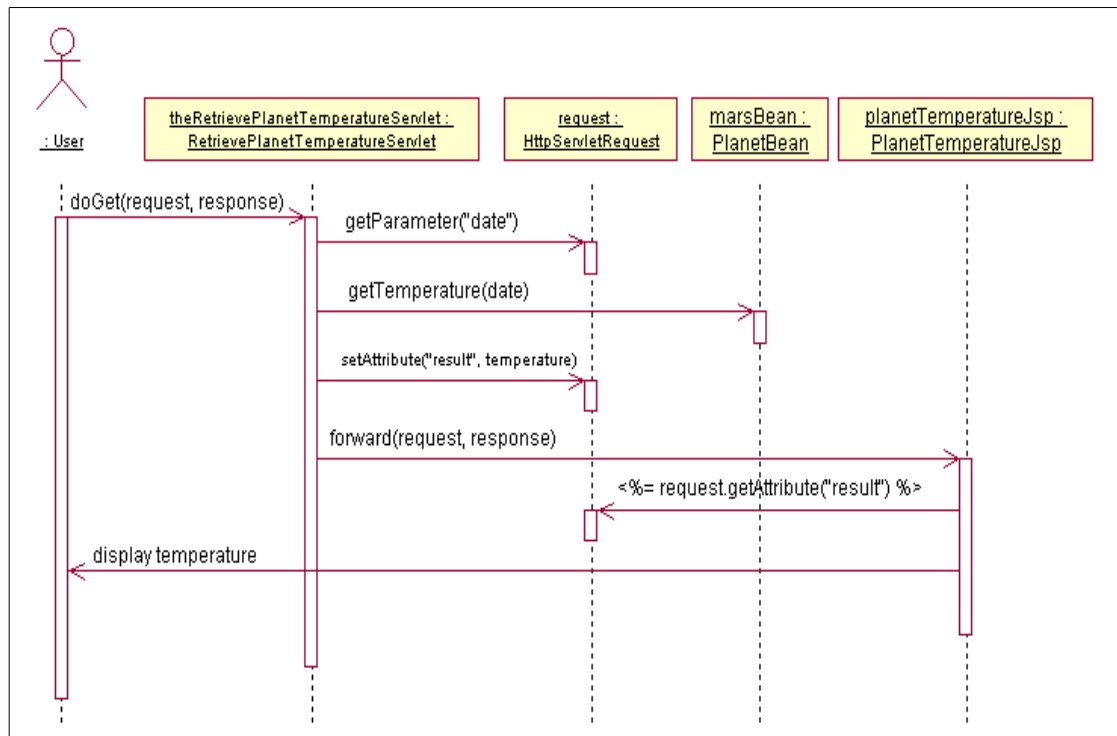


Figure 25. Object interaction diagram - Model-View-Controller example

- The servlet implements the doGet() method that represents the *Interaction Controller* logic. It retrieves the user entered value (a date) by calling getParameter("date") method on the request.
- Subsequently, the servlet instantiates the marsBean of type PlanetBean and calls the getTemperature() method on the marsBean, where marsBean represents the *business logic*.
- The servlet stores the temperature retrieved from the marsBean in the request object.
- Finally, the servlet calls the forward() method to pass control to planetTemperatureJsp, which represents the *display page*.
- This JSP inserts the dynamically retrieved Mars temperature for the specified date by using the following JSP expression:
`<%=request.getAttribute("result") %>.`

The following code segments show how the MVC interaction outlined above can be implemented.

```
package itso.solarsystem.weatherinformation;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;

public class RetrievePlanetTemperatureServlet extends HttpServlet {
    public void doGet (HttpServletRequest request,
        HttpServletResponse response)
        throws javax.servlet.ServletException, java.io.IOException {

        String dateString = request.getParameter("date");
        Date date = new Date(Date.parse(dateString));

        PlanetBean planetBean = new PlanetBean("Mars");
        Float temperature = planetBean.getTemperature(date);

        request.setAttribute("result", temperature);

        RequestDispatcher rd;
        rd =getServletContext().getRequestDispatcher("temperature.jsp");
        rd.forward(request, response);
    }
}
```

Figure 26. *RetrievePlanetTemperatureServlet* - controller source code

```
<HTML>
<HEAD><TITLE>Solar System Web application</TITLE></HEAD>
<BODY BGCOLOR="#FFFFFF"><H1>Solar System</H1>

Planet: Mars <BR>
Date: <%= request.getParameter("date") %><BR>
Temperature: <%= request.getAttribute("result") %>

</BODY>
</HTML>
```

Figure 27. *temperature.jsp* - view source code

```

package itso.solarsystem.weatherinformation;
import java.util.Date;

public class PlanetBean {
    private Float temperature = new Float(0.0);
    private String name;
    private Float humidity = new Float(0.0);
    private Float pressure = new Float(0.0);

    public PlanetBean(String newPlanetName) {
        super();
        name = newPlanetName;
    }

    public Float getHumidity(java.util.Date date) { return humidity; }
    public String getName() { return name; }
    public Float getPressure(java.util.Date date) { return pressure; }
    public Float getTemperature(java.util.Date date) { return temperature; }

    public WeatherResultBean getWeather(java.util.Date date) {
        return new WeatherResultBean(temperature, pressure, humidity);
    }
}

```

Figure 28. PlanetBean - model source code

This simple MVC example can be extended to implement more complex Web applications. For example, based on the results received from the business logic, a servlet may choose to call a different view page, etc.

7.3.3 Advantages and disadvantages of the MVC design pattern

To summarize, the MVC design pattern recognizes various types of program logic involved in implementing a typical Web application and advocates the separation of business logic, page construction logic, and interaction controller logic. We recommend using Java servlets for implementing interaction controllers, JSPs for implementing dynamic display pages, and simple Java beans and/or EJBs for implementing business logic. Such a separation provides the following advantages compared to a monolithic implementation:

- Leverage different skill sets:

As discussed earlier, the skill sets required to design an HTML page are vastly different from the skill sets required to code the business logic in Java. The separation of concerns outlined here allows for the effective use of skilled resources.

- Increase reusability:

In a non-trivial application there are usually display pages that can be called from multiple interaction controllers. For example, an error page may be called as a result of many interactions. Similarly, based on some conditions there might be a need to perform page selection. For example, one might have to display different pages for admin users vs. normal users. Finally, the business logic could be used by several interactions or applications. For example, you may have to display the current weather information on multiple pages of a Web application. Under these conditions a clear separation of concerns would increase the potential for reuse.

- Can support multiple user interfaces:

e-business applications often support multiple user interfaces such as HTML clients for the Internet, application clients for the call center, wireless handheld PDAs, and voice response units. Separating the presentation logic from the business logic allows reuse of the business logic component among these user interface environments. In addition to providing higher reusability, such a separation also ensures the consistency of the business logic across these applications.

- Improved maintainability of the site:

Under this scenario it is easy to make changes to the user interface without affecting the business logic and vice versa. For example, the user interface can be changed to leverage a new HTML standard such as CSS without affecting the business logic components making it easy to respond to the demands of the business in record speed.

- Reduced complexity:

Any non-trivial application implemented without clear separation of concerns could result in large and complex code. For such applications the MVC separation reduces the complexity.

On the other hand, it is important to recognize the following disadvantages of the MVC design pattern:

- Could be an overkill for small applications:

MVC design pattern can introduce extra artifacts that may not be necessary for very simple cases and in fact increase the complexity of the

application. However, if the application is likely to evolve over time, then it still may be beneficial to “pay now versus pay later” to gain the flexibility provided by the MVC design pattern.

- High level of communication requirements between various groups:

Since various groups would be typically responsible for implementing the various parts of the application, there is a need for a defined communication plan. For example, interaction control developers need to know display page names and vice versa. They have to agree upon a naming convention for various parameters and attributes and interaction control developers need to know the business logic, etc.

7.4 Application component contracts

The previous section outlined issues related to the structure of Web interactions. Now we turn our focus towards issues related to passing the data between the various model-view-controller components. In essence, these are the guidelines for defining the contract between the interaction controller, business logic, and display pages.

In the previous MVC example, the application always returned the average temperature of planet Mars for the specified date. In that case the interaction controller expected a single field back upon executing the business logic `marsBean.getTemperature(date)`. This single result field was returned as a simple string. If the interaction controller expects more than one field as a result of executing the business logic, then we recommend returning a data bean that wrappers all the result fields. Since this data bean represents the result of executing business logic, we call it a *Result bean*. A *Result bean* effectively defines the contract between a particular piece of business logic and a particular interaction controller.

Now let us turn our attention towards the contract between the interaction controller and the page constructor. For simple interactions the *Result bean* itself can be used to define such a contract. However in some cases there might be a reason to introduce a *View bean*. A *View bean* is responsible for combining the result data and the display-specific attributes. For example, let us assume that some users of the solar system Web application would like to see the temperature in Fahrenheit and others would like to see it in Celsius. The model, *PlanetBean*, should not be bothered with such unit conversion details. The `PlanetBean.getTemperature()` method can be designed to return the temperature in a base unit such as Kelvin. The conversion from Kelvin to Fahrenheit or Kelvin to Celsius can be done by the *View bean* based on the user preference.

Such Result beans and View beans collect multiple result fields, minimize the number of calls, and simplify the contract between various components. The figure below demonstrates the relationship between MVC components, Result beans and View beans. It is important to note that in simple Web interactions, both a Result bean and a View bean could be implemented by the same Java bean.

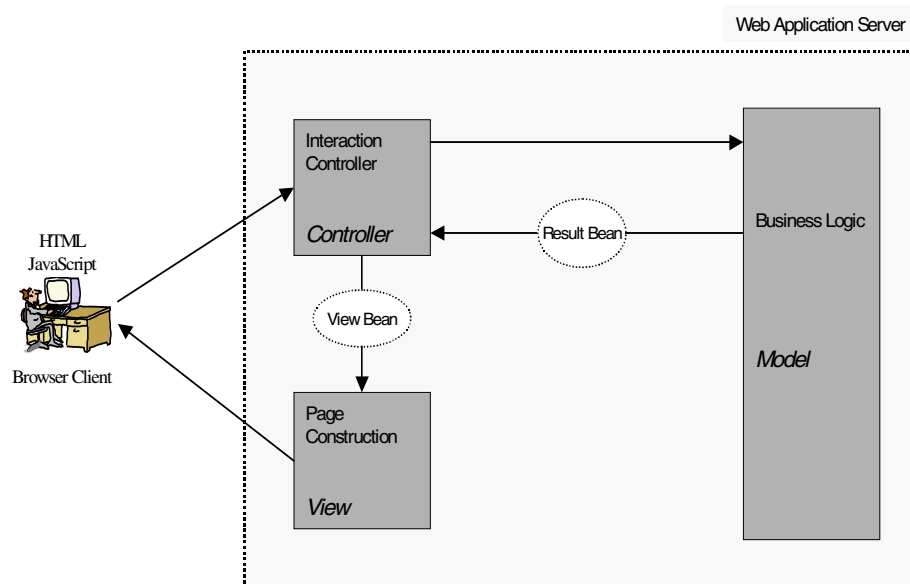


Figure 29. Result bean and View bean design pattern

7.4.1 Result beans and View beans design pattern

Result bean - defines the contract between the interaction controller and the business logic. It wrappers all the return values the interaction controller expects to receive upon executing a piece of business logic. Hence it provides an easy communication mechanism between the developers of these two components. Result beans are usually implemented as simple Java beans. Since Java beans by definition are serializable they can be passed by value between EJB-based business logic implementations. Whenever possible, Result bean properties should be implemented as read-only properties. A constructor could be used to initialize these properties during instantiation. This prevents the interaction controllers and page constructors from inadvertently updating the data.

It is important to note that a Result bean can be reused by multiple business logic methods or objects. For example, based on some condition, the

controller may decide to call two different business logic methods. If it is appropriate, both methods could use the same Result bean to return the results. The reverse can also be true. Multiple controllers may call the same business logic that returns the same Result bean to all controllers. In our solar system Web example, the weather Result bean can be returned either by a PlanetBean.getWeather() or a MoonBean.getWeather()

View bean - defines the contract between the controller and the view. It lists all the attributes the JSP can display. The main benefit of defining such a View bean is to make it easy for the JSP page designer to get all the required data in one place. The display page often contains the data from the following sources:

- Result bean properties (returned by the business logic)
- HTTP request data (including attributes, parameters, cookies, URL string)
- Session state
- Servlet context

The controller is responsible for instantiating the View bean and initializing all the properties of the bean. View beans can be designed to be responsible for view-specific transformations. For example, a View bean can be responsible for converting the monetary values into the user preferred currency. Such a View bean can have two properties, the monetary value in a base currency and the currency display type. The controller can initialize both of these properties. The View bean can use this information to call a reusable currency conversion library and get the monetary value in the appropriate format.

Usually, View beans are tightly coupled with a JSP since its primary purpose is to provide all the properties the JSP designer would need in one place. However, under special circumstances one can reuse the View beans by inheritance.

For example, let us assume that there are two types of users of a banking system, namely customers and customer service representatives (CSR). A particular screen in the system allows customers to see the transaction history. The same screen allows the CSRs to see transaction history and transaction details, allowing CSRs to answer any customer questions about the transaction such as where it was conducted, which bank representative was involved, etc. Since the CSR screen has all the data that the customer is able to see and has additional information, the CSRTransactionHistoryViewBean could be inherited from CustomerTransactionHistoryViewBean. This would ensure that the customers

and CSRs see the same basic information, with the CSRs seeing additional details. The following figures depict this scenario:

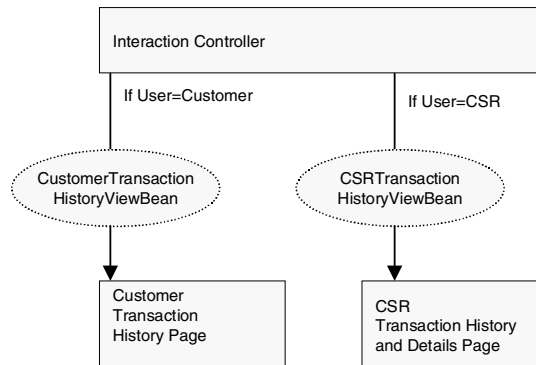


Figure 30. One interaction controller calling multiple display pages

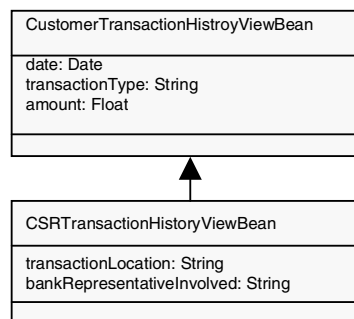


Figure 31. Class diagram - View bean inheritance example

7.4.2 Result bean and View bean design pattern example

In the MVC example, the solar system Web application always returned the average temperature of the planet Mars on the specified date. We would like to extend this application and allow users to get the average weather information including temperature, humidity, and pressure of planet Mars on the specified date. In addition, we would like to allow users to select the measurement system of their choice for displaying these results. For example, a user from Germany may want to see these results in a metric measurement system, whereas someone from the UK may want to see these results in imperial measurement system.

Under this scenario, the system needs to retrieve all the weather-related information and convert it to the specified measurement system before

displaying it. For example, the temperature can be obtained in Celsius and may need to be converted to Fahrenheit based on user preference.

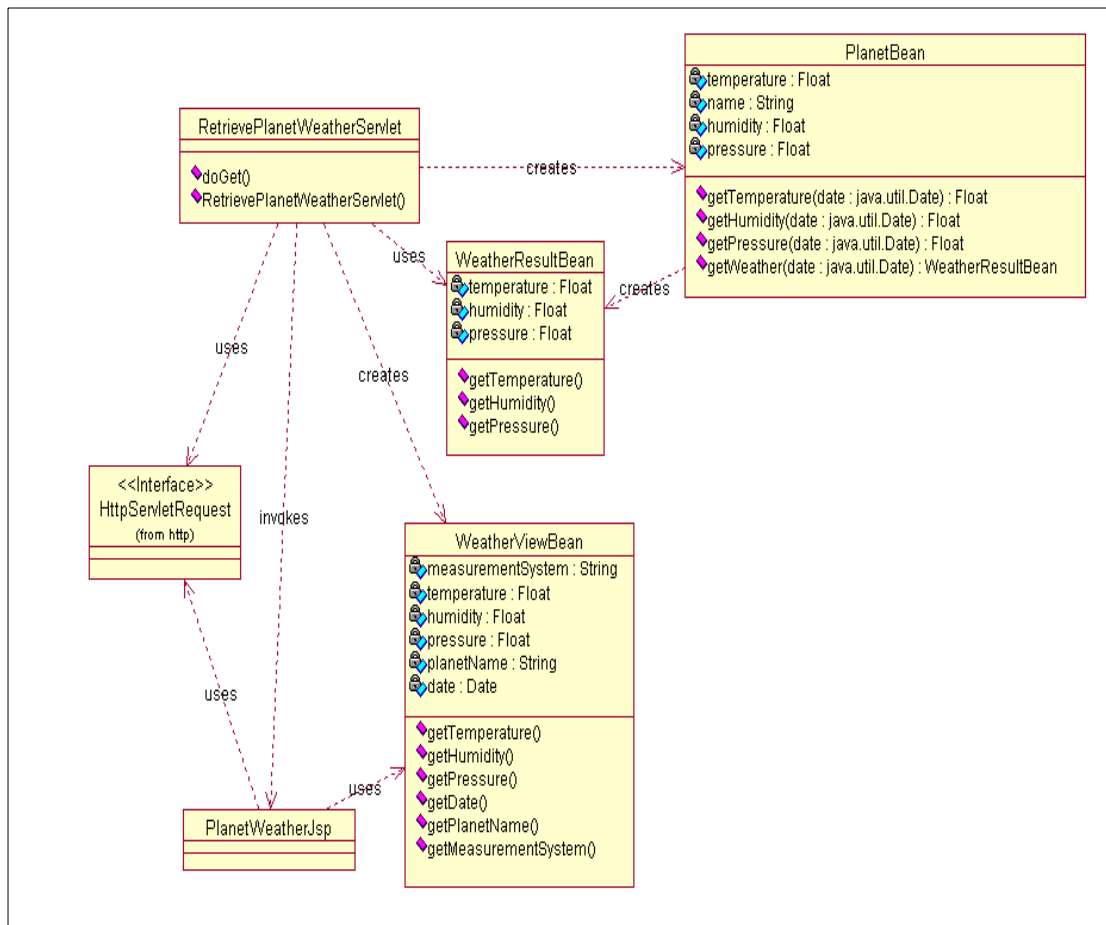


Figure 32. Class diagram - Result bean and View bean example

The class diagram in Figure 32 identifies the key classes required to implement this application using the MVC, Result bean, and View bean concepts. The classes are:

- **RetrievePlanetWeatherServlet**: represents the interaction controller. This class extends the `HttpServlet` and implements the `doGet()` method.
- **PlanetBean**: represents the business logic. It is a simple Java bean and has temperature, humidity, and pressure properties. For each of these properties it implements getter methods. In addition, it implements a

getWeather(date: Date) method. This method returns all the weather information wrapped in a Result bean called WeatherResultBean.

- WeatherResultBean: represents the Result bean that the servlet expects to receive upon executing the business logic. The PlanetBean method getWeather() creates this bean and populates its properties.
- WeatherViewBean: represents the View bean and lists all the properties that the JSP expects to display. Its properties include the results received from the getWeather() information and also the parameters received from the HTTP request such as measurement system and date. The servlet is responsible for creating this View bean and populating its properties.

It is important to note that the get methods of a View bean such as getTemperature() are responsible for converting the temperature from one measurement system to the other based on user preference. As discussed earlier, in doing so these methods may internally use a reusable class library of conversion methods. 7.5, “Application output formatting” on page 115 provides some guidelines for designing reusable Formatter beans.

- PlanetWeatherJSP: represents the display page. It is responsible for composing the response page that displays the weather information. In doing so it uses the WeatherViewBean to get the dynamic attributes of the page. The servlet is responsible for forwarding the request to this JSP.
- HttpServletRequest: represents the request received from the browser with the HTTP-specified data. It is passed into the doGet() method of the servlet as an input parameter. The doGet() method retrieves the user-entered parameters from this object using the getParameter() method.

Let us assume that the interaction begins by a user requesting a static HTML page. The page allows the user to enter a date and to choose the preferred measurement system for displaying the weather information of Mars. The following HTML code shows a sample HTML FORM that accomplishes this activity:

```
<FORM METHOD="GET" ACTION="RetrievePlanetWeatherServlet">  
Date: <INPUT NAME="date" TYPE="TEXT" SIZE=12> <BR>  
Measurement System: <SELECT NAME="measurementSystem" SIZE=1>  
<OPTION>Metric Measurement System  
<OPTION>Imperial Measurement System  
</SELECT>  
<INPUT TYPE="SUBMIT">  
</FORM>
```

After entering the required data the user clicks on the submit button that sends an HTTP GET request to the Web server. That in turn triggers the doGet() method on the PlanetWeatherServlet object.

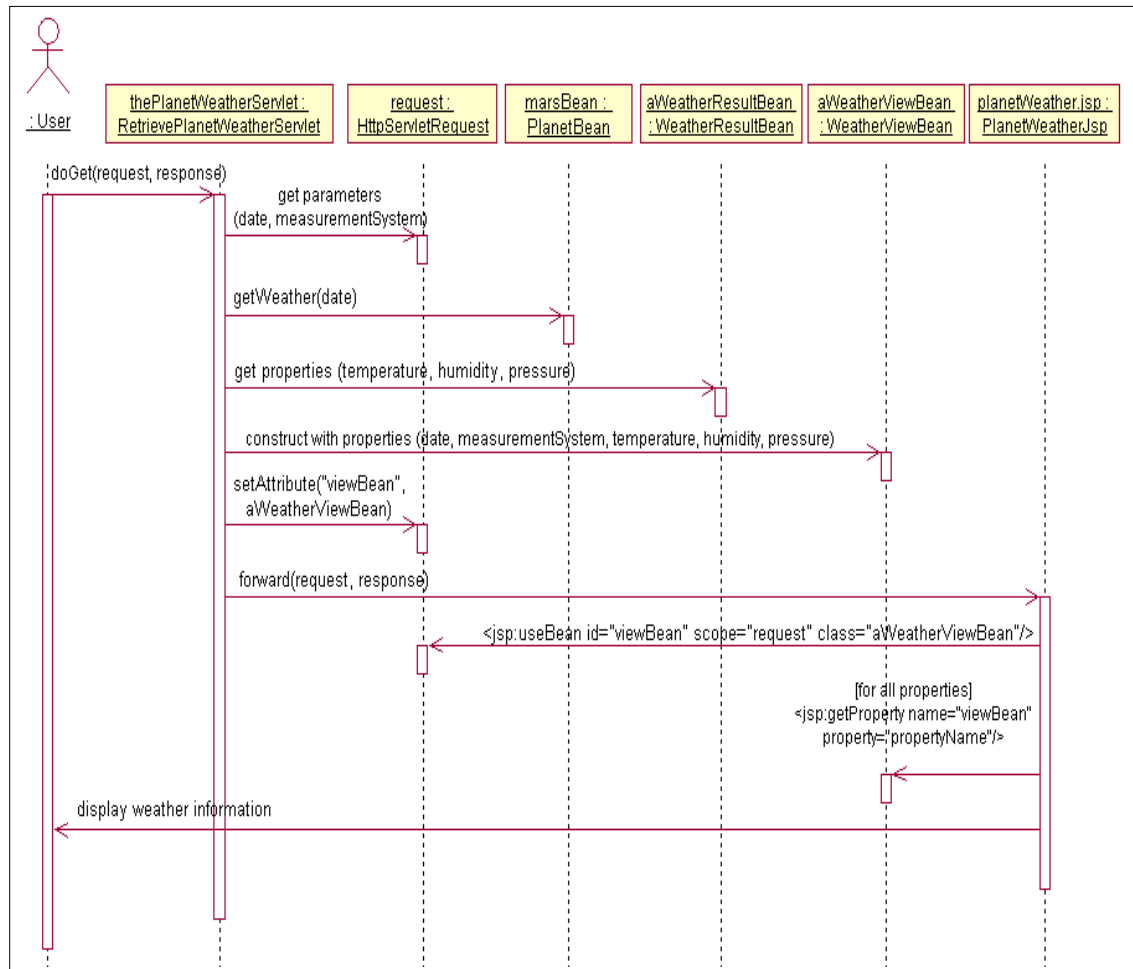


Figure 33. Object interaction diagram - Result beans and View beans

The object interaction diagram in Figure 33 shows the subsequent key interactions between various components.

- The RetrievePlanetWeatherServlet.doGet() method retrieves the user entered parameters namely “date” and “measurementSystem” from the HttpServletRequest object using the getParameter() method.

- Subsequently, the servlet instantiates the marsBean object of type PlanetBean and sends the getWeather() message by passing the date as an input parameter.
- The marsBean.getWeather(date) method is responsible for retrieving the weather information for the specified date from an external source. In doing so it may use one of the connectors provided by WebSphere. Finally, this method creates a WeatherResultBean, populates its properties, and returns this object back to the servlet.
- The servlet, retrieves all the properties (temperature, humidity, and pressure) from the Result bean using get methods.
- Then the servlet constructs a WeatherViewBean and populates all of its properties including date, measurementSystem, temperature, humidity, and pressure.

It is important to note that the servlet populates the View bean with all the fields needed by the JSP for display. In doing so, it combines the parameters received with the HTTP GET request such as date and measurement system and the results received from the Result bean such as temperature, humidity, and pressure.

The View bean uses the measurementSystem property to decide if the weather-related information needs to be returned by the get methods in the metric measurement system or the imperial measurement system.

- The servlet then stores the WeatherViewBean in the request context using the setAttribute() method.
- Finally, the servlet calls the forward() method to pass control to PlanetWeatherJsp, which represents the display page.
- The PlanetWeatherJSP inserts the dynamically retrieved weather information into the response page using <jsp:useBean> and <jsp:insertProperty> tags.

The following code segments show how the Result bean and View bean example outlined above can be implemented. PlanetBean code under this example remains the same the PlanetBean code under the MVC example. Therefore we don't repeat the PlanetBean code below. Please refer to Figure 28 on page 102 for the PlanetBean implementation.

```

public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException {
    PlanetBean planetBean = null;
    WeatherResultBean resultBean = null;
    WeatherViewBean viewBean = null;

    String dateString = request.getParameter("date");
    Date date = new Date(Date.parse(dateString));
    String measurementSystem = request.getParameter("measurementSystem");
    planetBean = new PlanetBean("Mars");
    resultBean = planetBean.getWeather(date);

    viewBean = new WeatherViewBean("Mars",
        date,
        measurementSystem,
        resultBean.getTemperature(),
        resultBean.getHumidity(),
        resultBean.getPressure());
    request.setAttribute("viewBean", viewBean);
    RequestDispatcher rd;
    rd = getServletContext().getRequestDispatcher("weatherInfo.jsp");
    rd.forward(request, response);
}

```

Figure 34. RetrievePlanetWeatherServlet.doGet() - controller source code

```

public class WeatherResultBean {
    private Float temperature;
    private Float humidity;
    private Float pressure;
    public WeatherResultBean(Float newTemperature,
Float newPressure, Float newHumidity) {
        temperature = newTemperature;
        pressure = newPressure;
        humidity = newHumidity;
    }
    public Float getHumidity() {return humidity;}
    public Float getPressure() {return pressure;}
    public Float getTemperature() {return temperature;}
}

```

Figure 35. WeatherResultBean - Result bean source code

```

public class WeatherViewBean {
    // define attributes
    //define a constructor

    //define get methods
    public String getDate() {return date.toString();}
    public String getMeasurementSystem() {return measurementSystem;}
    public String getPlanetName() {return planetName;}

    public String getPressure() {
        //Convert the pressure from the base unit to the required
        //measurement system and return the computed value. }

    public String getTemperature() {
        //Convert the temperature from the base unit to the required
        //measurement system and return the computed value. }

    public String getHumidity() {
        //Convert the humidity from the base unit to the required
        //measurement system and return the computed value. }
    }
}

```

Figure 36. WeatherViewBean - View bean source code

As discussed earlier, the conversion of values from one unit system to the other can be done by View beans. This hides the formatting complexity from JSP pages. In implementing such output formatting, View beans can use a set of reusable objects called Formatter beans. 7.5.1, “Formatter beans” on page 116 discusses this concept further.

Figure 37 on page 114 shows the corresponding JSP page. It is important to note that, since we use a View bean in this example, JSP can get all the required information from one bean. This eliminates the need to insert Java code directly inside JSPs. When you compare the JSP code below to temperature.jsp in Figure 27 on page 101 you can see how the use of the View bean could make the JSP page look more like tag-based HTML code.

```

<HTML>
<HEAD><TITLE>Solar System Web application</TITLE></HEAD>

<BODY BGCOLOR="#FFFFFF">
<jsp:useBean id="viewBean" scope="request"
class="itso.solarsystem.weatherinformation.WeatherViewBean"/>

<H1>Solar System</H1>
Planet: <jsp:getProperty name="viewBean" property="planetName"/><BR>
Date: <jsp:getProperty name="viewBean" property="date"/><BR>
Measurement system:
<jsp:getProperty name="viewBean" property="measurementSystem"/><BR>
Temperature: <jsp:getProperty name="viewBean" property="temperature"/>
<BR>
Pressure: <jsp:getProperty name="viewBean" property="pressure"/><BR>
Humidity: <jsp:getProperty name="viewBean" property="humidity"/><BR>

</BODY>
</HTML>

```

Figure 37. *weatherInfo.jsp* - View source code

7.4.3 Advantages and disadvantages of Result beans and View beans

In summary, Result beans define the contract between controllers and model and View beans define the contract between controllers and view. Both Result and View beans are implemented using simple Java beans. In some cases it may make sense to use the same bean as both the Result and the View bean.

Advantages of Result beans

- Clearly define what the controller expects back from the business logic.
- Once a Result bean is clearly defined the developers of the controller and the model can develop their components independently. This simplifies and optimizes the development process and allows for parallel development.
- Since Result beans can be serialized, they can be sent to remote servers or received from remote servers such as EJB-based distributed applications. In addition, they can be stored on a file for persistence purposes.
- The Result bean data structure can be reused by multiple business logic objects and interaction controller objects.

Advantages of View beans

- Clearly define all the fields a view (JSP) can display.
- Once a View bean is clearly defined the developers of the controller and the view can develop their components independently. This simplifies and optimizes the development process and allows for parallel development.
- The view (JSP) designer can get all the dynamic data from one View bean and use `<jsp:useBean>` and `<jsp:getProperty>` tags to insert these values. This allows the JSP designer to concentrate on the look and feel of the page rather than worrying about gathering data from various sources (for example, sessions, cookies, Result beans, etc.) and coding complex view-specific Java code. View beans effectively hide these complexities from the display page designer.
- Complete separation of the view specific logic from the business logic, for example, currency conversion based on the user preference.
- Using inheritance as outlined above one can promote View bean reuse and ensure similar information is received by all users, for example, CSRs and customers.
- View beans can be used with tools such as WebSphere Page Designer, which allows a developer to insert JavaBean properties into JSPs.

On the flip side, it is important to recognize the following disadvantages of introducing Result beans and View beans:

Disadvantages of View beans

- View beans are tightly coupled with display pages and interaction controllers. This implies that any changes to the dynamic content of the display page will require a change to the interaction controller. We depend on the interaction controller to gather all the required information in one View bean.
- For small applications the introduction of Result beans and View beans could result in too many individual pieces of code and increase the complexity of application management.
- The number of artifacts to be coded, managed, and maintained may be greatly increased.

7.5 Application output formatting

The View bean concept described above tries to minimize the need for inserting Java code directly inside a display page. For the most part, by using

View beans the display page designer can use `<jsp:useBean>` and `<jsp:setProperty>` tags to insert dynamic content. In doing so one can use tools such a WebSphere Page Designer that allows you to insert JavaBean properties into JSPs.

In order to insert complex tables or drop down lists, complex conditional loops may be needed. JSP API 1.0 does not define repeat tags that allow for looping through an indexed JavaBean property. In order to overcome this limitation, consider the following options:

- Implement the complex table or drop down list generation code in Java and wrapper it inside the bean.
- Insert complex conditional loop Java code inside the JSP.
- Use WebSphere specific `<tsx:repeat>` tag.

Among these options we recommend the first one since it hides the complexity from the JSP designer and promotes reuse. This allows non-programmers to design the display page easily.

7.5.1 Formatter beans

It is possible for a number of display pages to share common methods that dynamically generate table bodies, drop down lists, and radio button lists based on the values in an indexed field. In addition, it is possible to build reusable beans that convert values from one unit to the other. We call such reusable beans *Formatter beans*.

7.5.2 Formatter bean example

Let us assume that we would like to display the average temperature of Mars for every day of this year on one display page. To do this we implement an `AllYearMarsTemperatureJSP` and `AllYearMarsTemperatureViewBean`. This View bean has a method called `getAllYearTempratureTable()` that loops through the indexed properties and prints the dynamic content to an out stream. The JSP can simply retrieve this table by calling this get method and directing the output from the method to the implicit JSP object *out*.

In doing so, View beans may internally use a table generation Formatter bean that takes the indexed fields as input and based on the content returns an HTML table body. Further description and some sample Formatter beans can be found at:

<http://oss.software.ibm.com/developerworks/opensource/jsp/index.html>

7.5.3 Advantages and disadvantages of Formatter beans

To summarize, a Formatter bean is a bean that wraps reusable HTML formatting logic inside a method.

Advantages of Formatter beans

- Eliminate the need for inserting complex scripting logic inside a JSP.
- Promote reusability of the formatting logic.
- Hide the complexity of scripting logic from view designers.
- Promote the ability to drop common information on multiple pages such as displaying the current weather information and current stock price on all the pages of the Web application.

Disadvantages of Formatter beans

- Some of the display page functionality that is best expressed in a JSP is now moved into Java code.

7.6 Application business logic granularity

This section primarily focuses on the design issues related to the *Model* part of the MVC design pattern. From our discussions earlier, we recognize that the business logic part of a Web application is the piece of code ultimately responsible for satisfying client requests. Hence, it must address a wide range of potential requirements including transactional integrity, application data access, workflow support, and integration of new and legacy applications. In order to achieve this, business logic components may use various protocols including JDBC, JNDI, IIOP, RMI, DCE, messaging, etc. to communicate with enterprise applications, enterprise data sources, and external applications. The *Model* is not only responsible for implementing the business logic, but also for hiding the details of the data and application access protocols. This can be achieved by wrapping the model with a Java bean.

In implementing such business logic components one can choose between the following levels of granularity:

- One business logic bean per task:

This approach implements one business logic bean per business task, offering the lowest granularity. We call these beans *Command beans*. 7.6.1, “Command beans” on page 118 explains this concept.

- One business logic bean that groups a related set of tasks:

This approach implements multiple methods, each representing a piece of the business logic, and groups related methods under one bean. In doing so, usually all methods related to a *State* in the underlying business process are wrapped together by a bean. Hence these beans are called *State beans*. Since only related methods are grouped together, this style has a medium level of granularity. 7.6.4, “An alternative approach” on page 126 explains this concept further.

- One business logic bean that groups all the tasks:

This approach implements multiple methods, each representing a business task and groups them all under one bean. This implementation has the highest level of granularity among all the options listed here. Such a monolithic approach complicates system development and any maintenance processes. Hence this style should be avoided.

Among these, we recommend the first two approaches. Both Command and State bean approaches have advantages over the last approach.

7.6.1 Command beans

Command beans can be defined as Java beans that provide a standard way to invoke business logic. The following are the key characteristics of Command beans:

- Each Command bean corresponds to a single business logic task, such as a query or an update task.
- All Command beans inherit from a single command interface. In essence they implement the command interface.
- The inherited Command bean defines business domain specific properties such as account numbers.
- Command execution results are stored as properties of a Command bean, therefore, Command beans also act as Result beans.
- Commands have a simple, uniform usage pattern:
 - a. Create an instance of the Command bean.
 - b. Initialize the bean by setting its properties.
 - c. Cause the bean to execute its action by calling one of its methods.
 - d. Access the results of command execution by inspecting its properties.
 - e. Discard the command object.
- Commands can be serialized.

The figure below shows how such a Command bean would interact with the other components of the Web application.

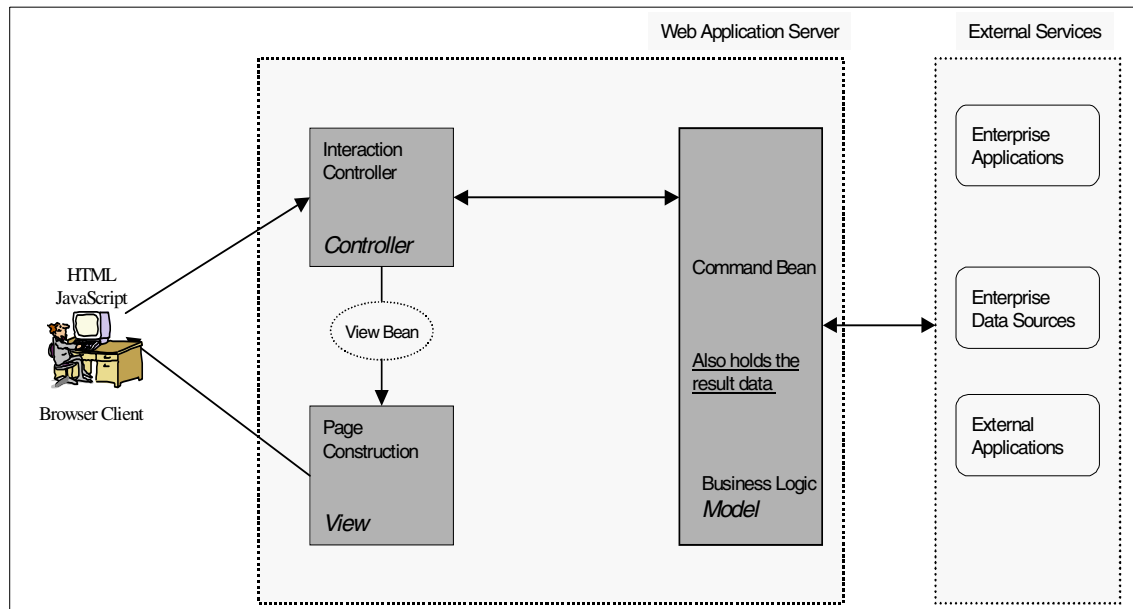


Figure 38. Command Beans

No need for Result beans

It is important to note that the Command bean contains the command execution results; hence there is no need to introduce another Result bean. In essence, the Command bean encapsulates both the business logic and result data.

7.6.2 Command bean example

Let us say that we would like to implement the example in 7.4.2, “Result bean and View bean design pattern example” on page 107 using the Command bean approach. The application allows the user to get the average weather information including temperature, humidity, and pressure of Mars on the specified date. In addition, the user can select the measurement system of choice for displaying these results.

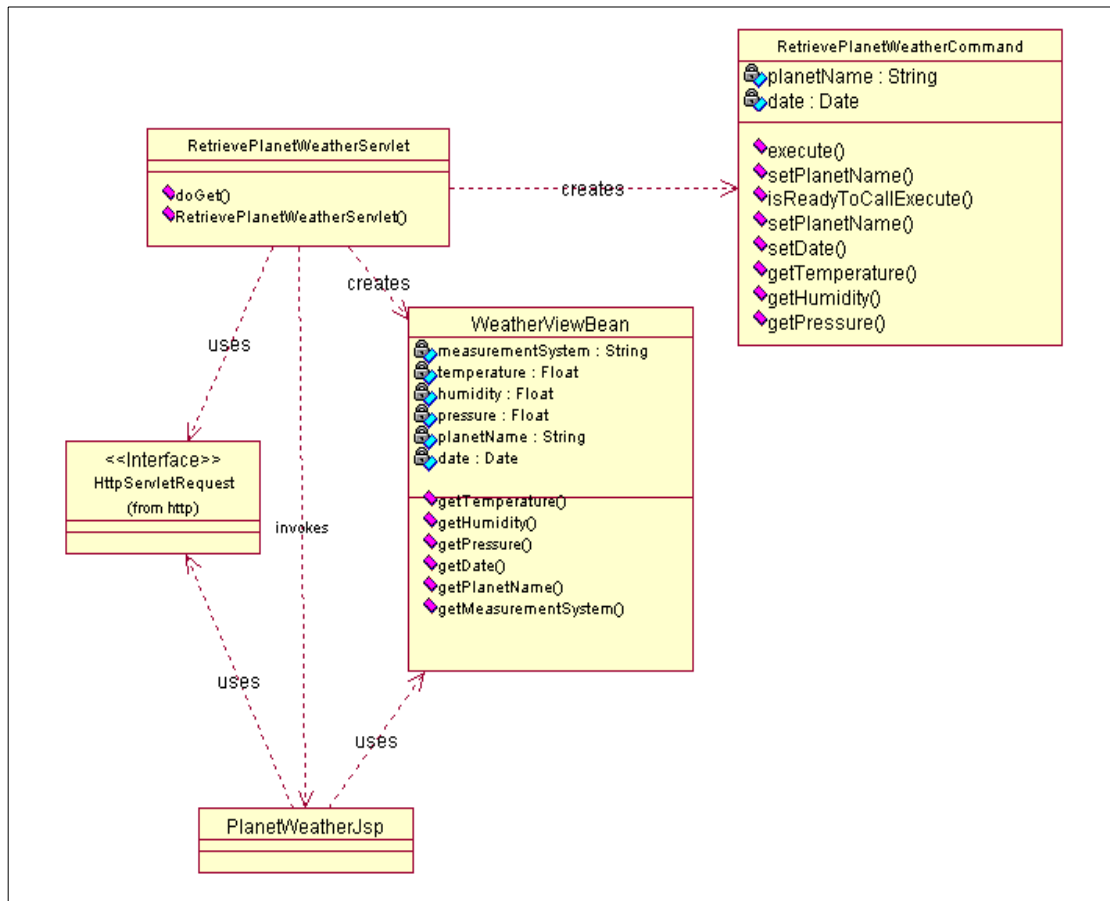


Figure 39. Class diagram - Command bean example

The above class diagram identifies the key classes required to implement this application using the MVC, Result bean, and View bean concepts. They are:

- **RetrievePlanetWeatherServlet**: represents the interaction controller. This class extends the `HttpServlet` and implements the `doGet()` method.
- **RetrievePlanetWeatherCommand**: represents the business task. It is a simple Java bean that implements the command interface. It defines planet name and date properties that represent the input parameters for the query. In addition, it defines temperature, humidity, and pressure properties that represent the results received from the query. For each of these properties it implements getter methods. In addition, it implements the `execute()` method that retrieves the data, in our case from a database, and sets the appropriate result values.

- **WeatherViewBean:** represents the View bean and lists all the properties that the JSP expects to display. Its properties include the results received from the command `execute()` method and also the parameters received from the HTTP request such as measurement system and date. The servlet is responsible for creating this View bean and populating its properties.

It is important to note that the `get` methods of the View bean such as `getTemperature()` are responsible for converting the temperature from one measurement system to the other based on user preference. In doing so, these methods may internally use a reusable class library of conversion methods. 7.5, “Application output formatting” on page 115 provides some guidelines for designing reusable `Formatter` beans.

- **PlanetWeatherJSP:** represents the display page. It is responsible for composing the response page that displays the weather information. In doing so it uses the `WeatherViewBean` to get the dynamic attributes of the page. The servlet is responsible for forwarding the request to this JSP.
- **HttpServletRequest:** represents the request received from the browser with the HTTP-specified data. It is passed into the `doGet()` method of the servlet as an input parameter. The `doGet()` method retrieves the user-entered parameters from this object using the `getParameter()` method.

Let us assume that the interaction begins by a user requesting a static HTML page that allows the user to enter a date and choose the preferred measurement system for displaying the weather information of Mars. The following HTML code shows a sample HTML FORM that accomplishes this activity:

```
<FORM METHOD="GET" ACTION="RetrievePlanetWeatherServlet">
<INPUT NAME="date" TYPE="TEXT" SIZE=12> <BR>
<SELECT NAME="measurementSystem" SIZE=1>
<OPTION>Metric Measurement System
<OPTION>Imperial Measurement System
</SELECT>
<INPUT TYPE="SUBMIT">
</FORM>
```

After entering the required data the user clicks on the submit button that sends an HTTP GET request to the Web server. That in turn triggers the `doGet()` method on the `RetrievePlanetWeatherServlet`.

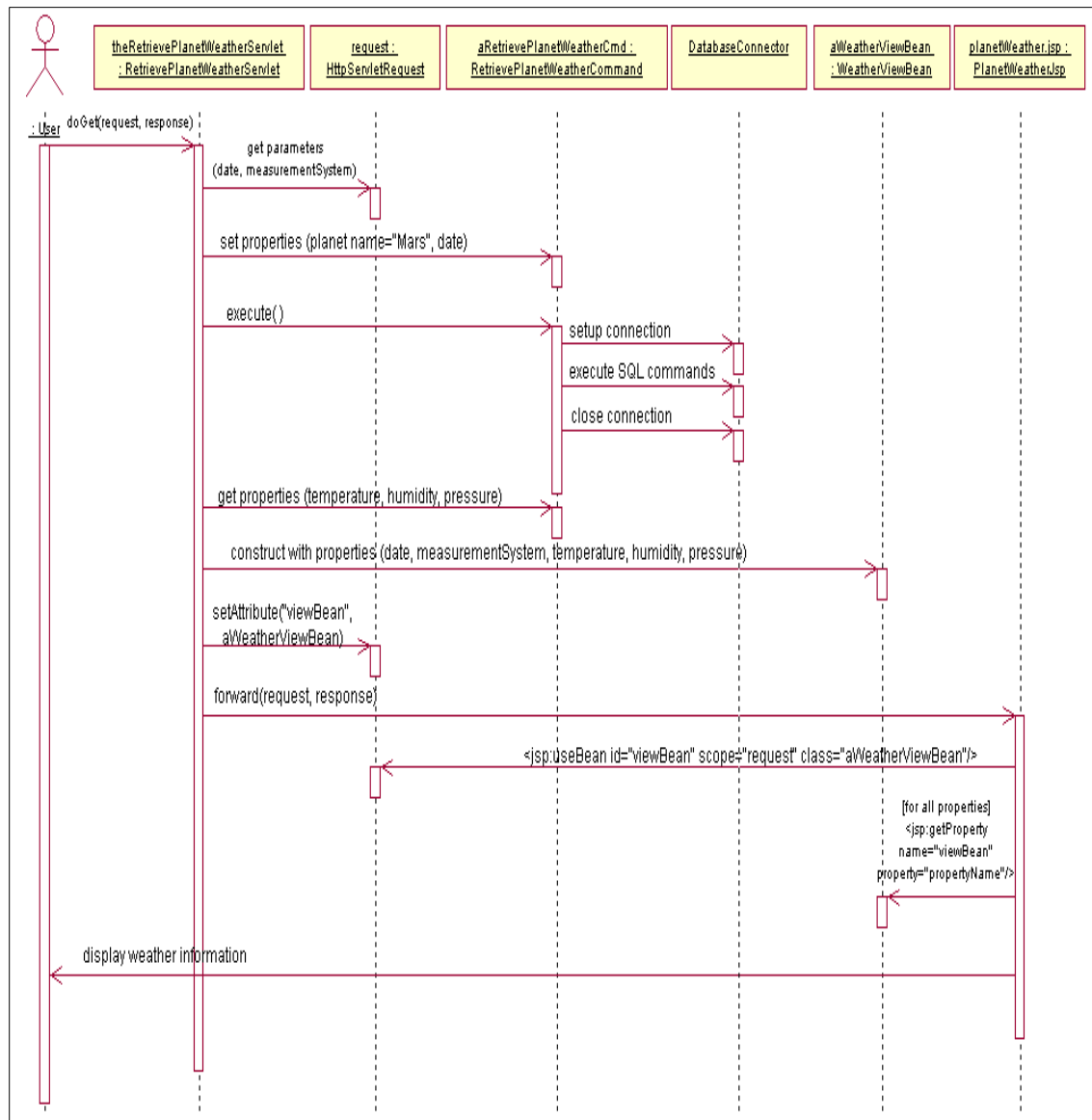


Figure 40. Object interaction diagram - Command beans

The object interaction diagram in above shows the subsequent key interactions between various components.

- The `RetrievePlanetWeatherServlet.doGet()` method retrieves the user entered parameters namely “date” and “measurementSystem” from the `HttpServletRequest` object using the `getParameter()` method.
- Subsequently, the servlet instantiates the `RetrievePlanetWeatherCommand` bean and sets the planet name and date values.
- Then it calls the `execute()` method on the Command bean.
- The `execute()` method is responsible for retrieving the weather information for the specified date from an external source. In doing so it may use one of the connectors provided by WebSphere. In the example above this method obtains a connection with the database driver and sends an SQL string for execution. Then it releases the database connection. Finally, this method populates the result properties (temperature, humidity, and pressure) and returns control to the servlet.
- The servlet retrieves all the properties (temperature, humidity, and pressure) from the Command bean using get methods.
- Then the servlet constructs a `WeatherViewBean` and populates all of its properties including date, measurementSystem, temperature, humidity, and pressure.

It is important to note that the servlet populates the View bean with all the fields needed by the JSP for display. In doing so, it combines the parameters received with the HTTP GET request such as date and measurementSystem with the results received from the Result bean such as temperature, humidity, and pressure.

The View bean uses the measurementSystem property to decide if the weather-related information needs to be returned by the get methods in the metric measurement system or the imperial measurement system.

- The servlet then stores the `WeatherViewBean` in the request context using the `setAttribute()` method.
- Finally, the servlet calls the `forward()` method to pass control to `PlanetWeatherJsp`, which represents the display page.
- The `PlanetWeatherJSP` inserts the dynamically retrieved weather information into the response page using `<jsp:useBean>` and `<jsp:insertProperty>` tags.

The following code segments show how the Command bean example outlined above can be implemented. `RetrievePlanetWeatherServlet`, `RetrievePlanetWeatherCommand`, `WeatherViewBean`, `weatherInfo.jsp` are the key classes under consideration. This example implements the same use

case as the Result bean and View bean example. For that reason, WeatherViewBean and weatherInfo.jsp code remains the same for both examples. We don't repeat those code segments below. Please refer to Figure 36 on page 113 for WeatherViewBean source code and Figure 37 on page 114 for weatherInfo.JSP source code.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws javax.servlet.ServletException, java.io.IOException {
    PlanetBean planetBean = null;
    WeatherResultBean resultBean = null;
    WeatherViewBean viewBean = null;

    String dateString = request.getParameter("date");
    Date date = new Date(Date.parse(dateString));
    String measurementSystem = request.getParameter("measurementSystem");

    RetrievePlanetWeatherCommand command;
    command = new RetrievePlanetWeatherCommand();
    command.setPlanetName("Mars");
    command.setDate(date);
    try {
        command.execute();
    } catch (itso.solarsystem.command.CommandException cmdEx) {

        //handle error - e.g. forward the control to an displayError.jsp
        return;
    }

    viewBean = new WeatherViewBean("Mars",
        date,
        measurementSystem,
        resultBean.getTemperature(),
        resultBean.getHumidity(),
        resultBean.getPressure());
    request.setAttribute("viewBean", viewBean);
    RequestDispatcher rd;
    rd = getServletContext().getRequestDispatcher("weatherInfo.jsp");
    rd.forward(request, response);
}
```

Figure 41. Modified RetrievePlanetWeatherServlet.doGet() method - Command bean example

The key difference between this servlet and the servlet in the Result bean and View bean example is that this servlet calls a command object to execute the business logic.

```
import java.util.Date;
import itso.solarsystem.command.*;

public class RetrievePlanetWeatherCommand implements Command {
    private java.util.Date date = null;
    private PlanetBean planet = null;
    private java.lang.String planetName = "";

    public Float getHumidity() {return planet.getHumidity(date);}
    public Float getPressure() {return planet.getPressure(date);}
    public Float getTemperature() {return planet.getTemperature(date);}

    public void setDate(java.util.Date newDate) {date = newDate;}
    public void setPlanetName(String newPlanetName) {planetName =
        newPlanetName;}

    public void reset() {planetName = null;date = null;planet = null;}

    public boolean isReadyToCallExecute() {
        //If required fields are not initialized, return null
        return (planetName != null) && (date != null);}

    public void execute() throws CommandException {
        if (isReadyToCallExecute()):
            //Connect to the data base and get the weather information for the
            //specified plant and date.  }
        else {
            throw new UnsetInputPropertiesException("required property date");
        }
    }
}
```

Figure 42. *RetrievePlanetWeatherCommand - Command bean source code*

The `RetrievePlanetWeatherCommand` implements the command interface. The `execute()` method is responsible for retrieving the required information from a data source. A downloadable version of the command interface can be found at:

<http://www.ibm.com/software/ebusiness/buildapps/understand.html>

7.6.3 Advantages and Disadvantages of Command beans

In summary, Command beans are stylized beans that provide a common interface for executing business logic. In addition to implementing the business logic, they hide the complexity of data and back-end application access.

Advantages of Command beans

- Provide a common interface for executing business logic.
- This common interface makes it easier to implement application development tools. Further releases of WebSphere development tools are expected to provide significant support based on this framework.
- This common interface can be further extended to support cross-tier communication and remote execution using the Command Manager pattern. WebSphere V3.5 is expected to implement this framework and provide the necessary API for cross-tier communication and remote execution of commands.
- This structure also makes it easy to implement automatic data caching. WebSphere V4.0 is expected to further extend the Command Manager pattern to implement caching.
- It hides the data and application access protocol details from the interaction controllers and page construction components. This abstraction layer allows for changing the data access mechanism without altering other components in the MVC design pattern.

Command beans are ideal where the Web application server acts primarily as a hub that receives and forwards requests to back-end applications where the complex business logic is executed. If there is a need to implement complex business logic locally on the Web application server node, then one could consider the approach described below.

7.6.4 An alternative approach

This approach provides the medium level of business logic granularity and implements one business logic bean that groups a related set of tasks. State beans are an example of this approach. State beans can be defined as business domain specific objects that group related business tasks associated with a state in the business process. It is critical to note that the State bean methods are responsible for accessing data from external services and should hide any protocol-specific details. Such an implementation will make it easy to change the back-end interface without

affecting interaction controllers and display pages. State beans differ from the Command beans primarily in their granularity.

The solar system weather information problem domain used throughout this chapter does not easily lend itself for use as a good example for this approach. For that reason, we use a discount brokerage application example to discuss how State beans can be designed. Let's say that this discount brokerage application allows users to enter a trade request, execute the trade, and view the status of the trade. Such an application may have the following states:

- **Trade Entry:** This allows the user to get a stock quote and research a stock, enter the details of the trade including stock ticker symbol, quantity, buy or sell, limit price, etc., update details of the trade, confirm the trade or cancel the trade.
- **Trade Execution:** Here there may be a need for the internal department to intervene if the required funds are not available. If things flow smoothly, the trade gets executed. This state may also allow the customer to see the status of the trade and to cancel the trade if the trade has not been executed yet.
- **Trade Completed:** Here the customer will be able to see the trade execution price, quantity purchased or sold, etc. This state can also allow the user to query a history of completed trades.

Under the State bean approach one would have three objects, each representing the state in the business process namely TradeEntryBean, TradeExecutionBean, and CompletedTradesBean. Each of these beans would have multiple methods each representing a business task. For example, TradeExecutionBean may have the following methods cancelTrade(), getAllOpenTrades() and getTradeStatus(). Each of these methods can take required parameters as input and return a Result bean.

Designing such a business logic model is very specific to a problem domain. Such a design follows the traditional object-oriented analysis and design process. We will not go into detail on object-oriented principals or design processes. Object-oriented designers are already familiar with modeling such business logic objects. Currently the Pattern Development Kit does not provide examples for implementing State beans.

The primary disadvantage of this approach is the lack of a standard interface for invoking business logic. Hence it does not lend itself to automatic caching techniques that can be implemented by extending the Command bean approach. However, the State bean approach is suitable for applications that require complex modeling on the Web application server node.

7.7 Application session management

For the purposes of this discussion, a session is defined as a series of requests originating from the same user, and the same browser. As discussed earlier, HTTP is a stateless protocol. Over the years a number of techniques have been developed to maintain the application state across multiple HTTP requests from the same user. Cookies and URL rewriting are the most commonly used techniques. WebSphere Application Server implements the HttpSession API that hides the complexity of these techniques from the Web application programmer. In addition, an HttpSession object internally implements a hash table of name-value pairs. Using the putValue() and getValue() methods one can store and retrieve application-specific information in this hash table.

WebSphere Application Server provides various configuration options for session management. These configuration options can influence the application behavior, performance, and failover capabilities. Hence we urge you to consider these options early in the design phase. We discuss some of these design issues in 7.7.2, “Session management design considerations” on page 137.

The following figure shows how the Command bean example can be extended to exploit the session management feature.

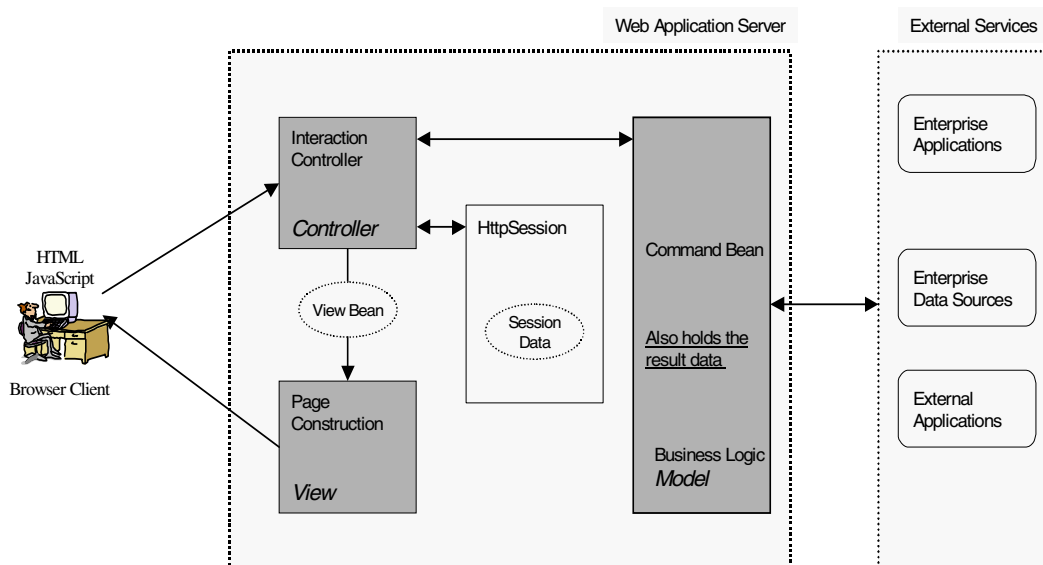


Figure 43. Session management

7.7.1 Session management example

The example in 7.6.2, “Command bean example” on page 119 expects the user to enter both the date and the preferred measurement system for every request. In order to demonstrate the session management issues we modify this use case as follows. Let us say that the user enters the date and preferred measurement system during the first request in a session. During subsequent requests in that session the user enters only the date and expects the system to use the measurement system entered during the first request to format the results.

Under this scenario, the system needs to save and use the measurement system across multiple requests. In order to implement this functionality one can exploit the HttpSession API.

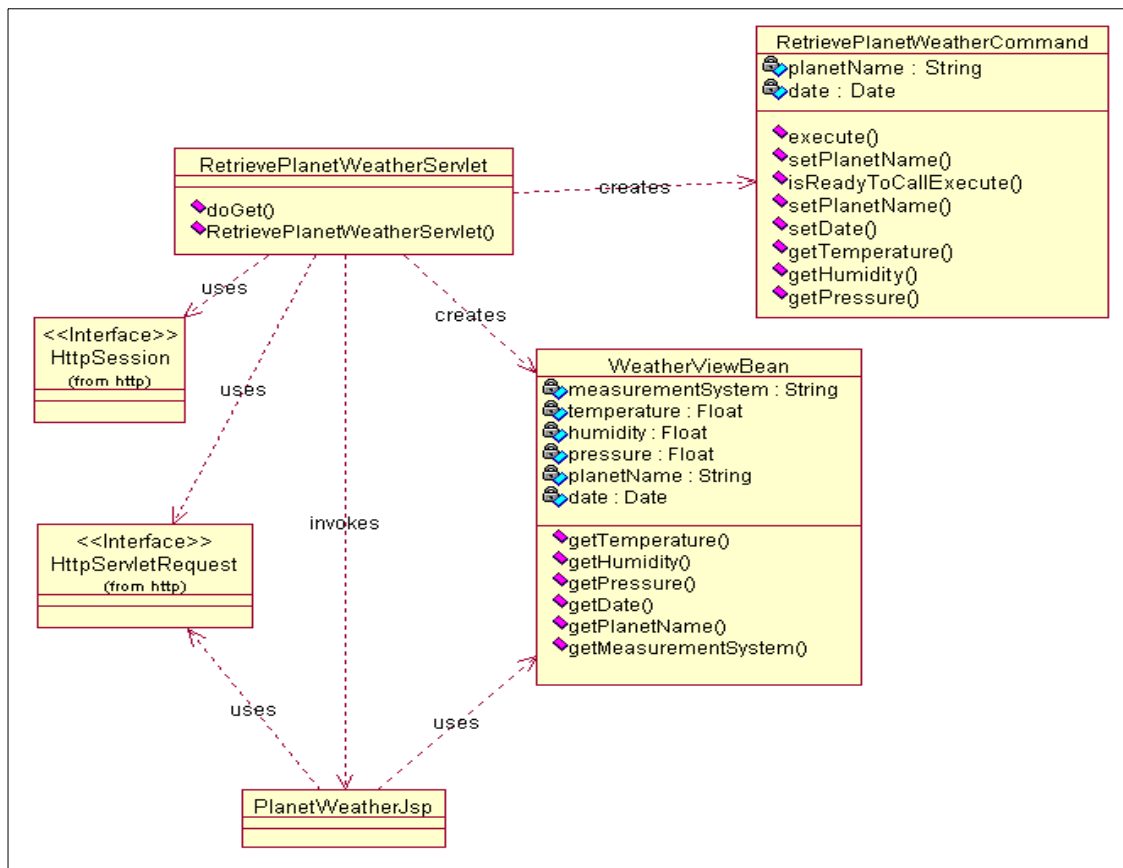


Figure 44. Class diagram - session management

The class diagram above identifies the key classes required to implement this application. The only difference between this and the Command bean example class diagram in Figure 39 on page 120 is that it uses the HttpSession object. Hence we only explain the HttpSession class below. For details on all other classes please refer back to 7.6.2, “Command bean example” on page 119.

- HttpSession: represents the session object. During the first request in a session the doGet() method creates this session object and stores the measurement system in the session hash table. During subsequent requests, doGet() retrieves this information and uses it to format the response page.

Let us assume that the interaction begins by a user requesting a static HTML page that allows the user to enter a date and choose the preferred measurement system for displaying the weather information of Mars. The following HTML code shows a sample HTML FORM that accomplishes this activity:

```
<FORM METHOD="GET" ACTION="RetrievePlanetWeatherServlet">
<INPUT NAME="date" TYPE="TEXT" SIZE=12> <BR>
<SELECT NAME="measurementSystem" SIZE=1>
<OPTION>Metric Measurement System
<OPTION>Imperial Measurement System
</SELECT>
<INPUT TYPE="SUBMIT">
</FORM>
```

After entering the required data, the user clicks on the Submit button that sends an HTTP GET request to the Web server. That in turn triggers the doGet() method on the RetrievePlanetWeatherServlet.

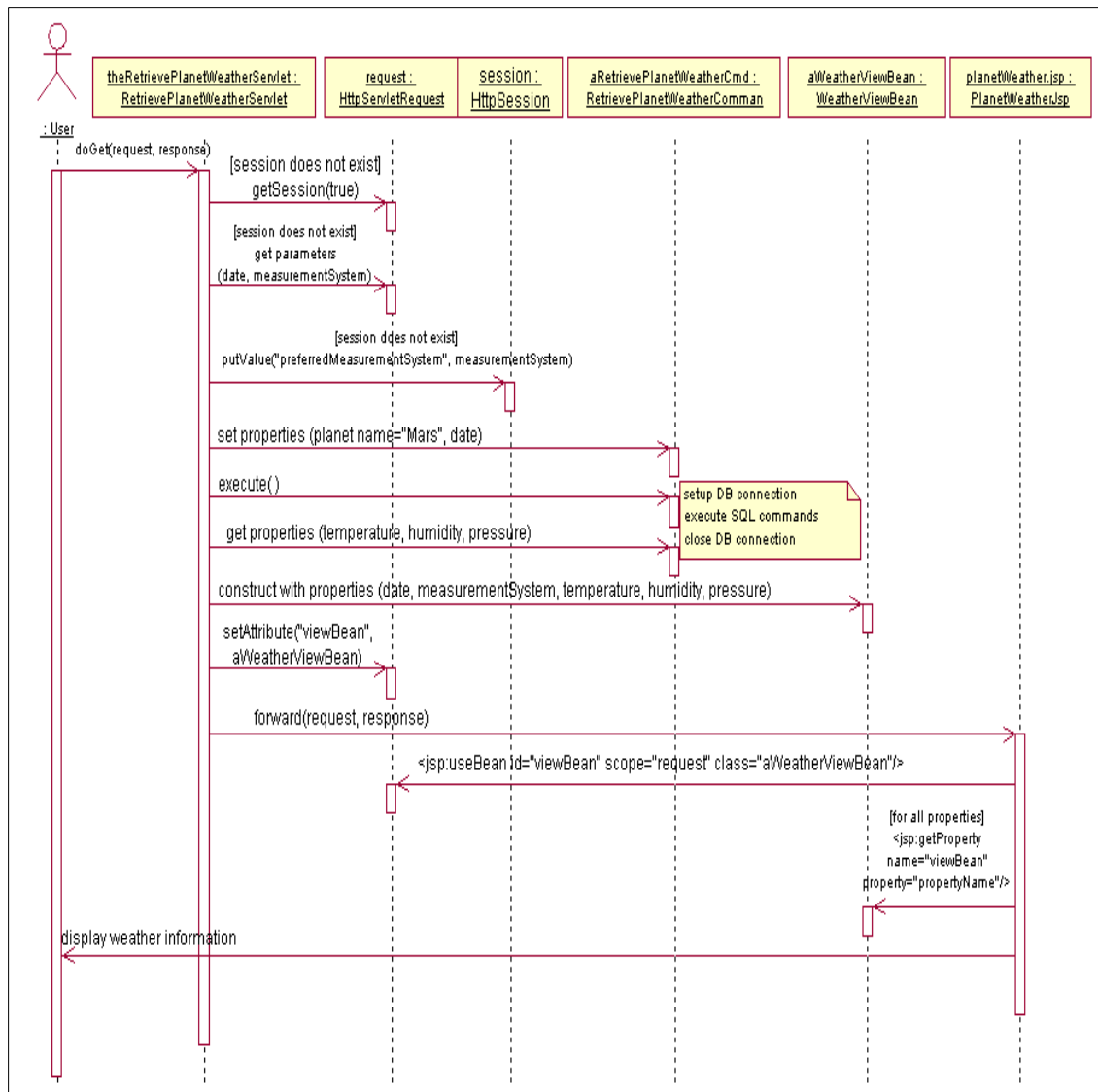


Figure 45. Object interaction diagram - session management - first request from the user

The above object interaction diagram captures the key interactions between various components during this *first request from the user*.

- The RetrievePlanetWeatherServlet.doGet() method checks to see if the session object exists by calling getSession(false). This method returns NULL if the session object does not exist.

- Since the session object does not exist, the servlet creates a session object for this user by calling `getSession(true)`.
- The servlet retrieves the user entered parameters namely “date” and “measurementSystem” from the `HttpServletRequest` object using the `getParameter()` method.
- Then it stores the `measurementSystem` in the session object by calling `putValue(“preferredMeasurementSystem”, measurementSystem)`.
- Subsequent interactions follow the same execution path as the Command bean example. The servlet instantiates the Command bean, sets its parameters, calls the `execute()` method on the Command bean. Then the servlet retrieves all the properties (temperature, humidity, and pressure) from the Command bean using get methods. The servlet constructs the View bean and passes control to the JSP that represents the display page.

The display page presents the results and allows the user to enter a different date for which he or she would like to see the weather information. The key difference between this entry page and the initial one is that the system does not ask the user to enter the measurement system. Once the user enters a new date and clicks on the Submit button, the `doGet()` method on the servlet is invoked again.

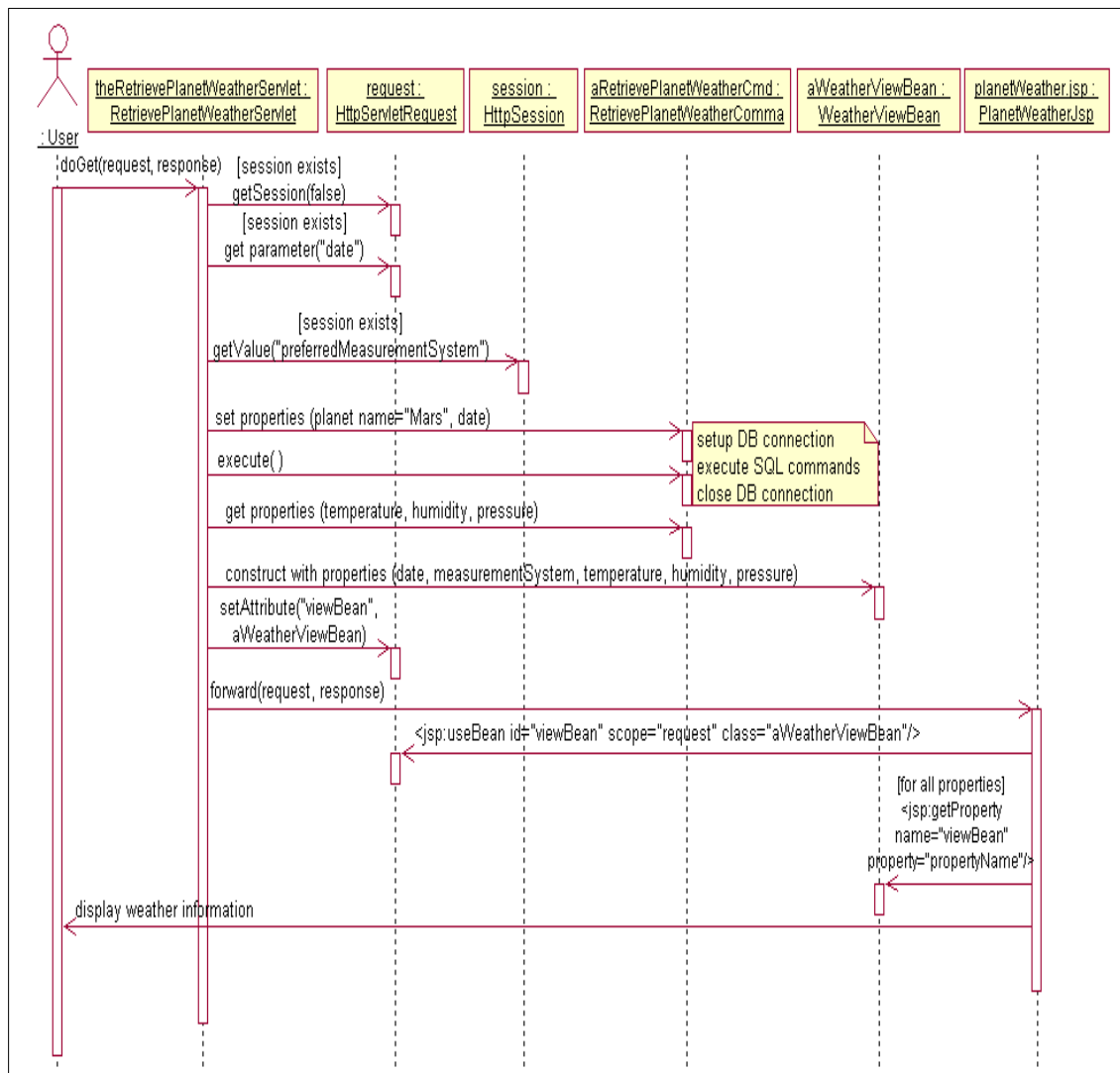


Figure 46. Object interaction diagram - session management - subsequent requests from the user

The above object interaction diagram captures the key interactions between various components during subsequent requests from the user:

- The RetrievePlanetWeatherServlet.doGet() method checks to see if a session object exists by calling getSession(false). Since a session object exists for this user, this method returns an HttpSession object.

- Since the session object exists, the servlet retrieves only the “date” from the `HttpServletRequest` object using the `getParameter()` method.
- Then it retrieves the `measurementSystem` from the session object by calling `getValue(“preferredMeasurementSystem”)`.
- Subsequent interactions follow the same execution path as the Command bean example. The servlet instantiates the Command bean, sets its parameters, calls the `execute()` method on the Command bean. Then the servlet retrieves all the properties (temperature, humidity, and pressure) from the Command bean using `get` methods. The servlet constructs the View bean and passes the control to the JSP that represents the display page.

The important classes in this example are `RetrievePlanetWeatherServlet`, `weatherInfo.jsp`, `WeatherViewBean`, and `RetrievePlanetWeatherCommand`. The `WeatherViewBean` and `RetrievePlanetWeatherCommand` do not change from 7.6.2, “Command bean example” on page 119. Hence we don't duplicate this code below. Figure 36 on page 113 shows the `WeatherViewBean` source code. Figure 42 on page 125 shows the `RetrievePlanetWeatherCommand` source code.

```

public void doGet(HttpServletRequest request,
    HttpServletResponse response)
    throws javax.servlet.ServletException, java.io.IOException {
    //Insert Code to getParameters from HttpServletRequest

    //See if session exists
    HttpSession session = request.getSession(false);

    if (session == null) {
        //SESSION DOES NOT EXIST - FIRST INTERACTION
        session = request.getSession(true);
        measurementSystem = request.getParameter("measurementSystem");
        session.putValue("myMeasurementSystem", measurementSystem);
    }
    else {
        //SESSION EXISTS - SUBSEQUENT INTERACTIONS
        measurementSystem = (String) session.getValue("mydMeasurementSystem");
        if (measurementSystem == null) {
            //INSERT CODE TO HANDLE ERROR
            // e.g. forwar request to DisplayError.jsp
            return;
        }
    }
    //The rest of the code looks similar to thr Command bean example
    //Set command parameters using values from request and session
    //execute() command.
    //Construct View bean
    //Forward request to weatherInfo.jsp.
}
}

```

Figure 47. Modified RetrievePlanetWeatherServlet.doGet() - session management source code

We would like to store the preferred measurement system in the session object and use it for subsequent requests. To handle this, the weatherInfo.jsp needs to be modified so that it can display the results and also accept the next date. Using this date and the measurement system entered earlier the system should be able to generate the next response page. This is done by appending an HTML FORM tag to the end of the JSP. The ACTION attribute of the FORM tag points back to the RetrievePlanetWeatherServlet. The weatherInfo.jsp Figure 48 shows how the JSP from the Command bean example can be modified to handle this scenario.

```

<HTML>
<HEAD><TITLE>Solar System Web application</TITLE></HEAD>

<BODY BGCOLOR="#FFFFFF">
<jsp:useBean id="viewBean" scope="request"
class="itso.solarsystem.weatherinformation.WeatherViewBean"/>

<H1>Solar System</H1>
Planet: <jsp:getProperty name="viewBean" property="planetName"/><BR>
Date: <jsp:getProperty name="viewBean" property="date"/><BR>
Measurement system:
<jsp:getProperty name="viewBean" property="measurementSystem"/><BR>
Temperature: <jsp:getProperty name="viewBean" property="temperature"/>
<BR>
Pressure: <jsp:getProperty name="viewBean" property="pressure"/><BR>
Humidity: <jsp:getProperty name="viewBean" property="humidity"/><BR>

<FORM ACTION ="RetrievePlanetWeatherServlet">
<B>Query planet's weather information</B><BR>
Enter Next Date (mm/dd/yyyy) :
<INPUT size="10" type="text" maxlength="10" name="date"><BR>
<INPUT type="submit" name="submit" value="Submit">
</FORM>
</BODY>
</HTML>

```

Figure 48. Modified weatherInfo.JSP - session management source code

The above example demonstrates how HttpSession objects can be used to carry application-specific information across multiple requests. With this example, the user preference is stored in the system only until any one of the following conditions is met: a session times out, user terminates the connection by closing the browser or the server is restarted.

This example can be extended so that the user registers preferences during the first visit to the site. The system can store such user preferences in a user profile database. During subsequent visits, the user can be identified using mechanisms such as user ID. Based on this ID, the system can retrieve the user profile information from the database and store that information on the session object to be used during that particular session. At the end of the session the Web application should remove the user profile information from the session pool so as to free up the resources on the Web application server.

7.7.2 Session management design considerations

Now that we understand the mechanics of session management we will focus on implementing applications that leverage this feature of WebSphere Application Server.

7.7.2.1 Cookies and URL rewriting

The WebSphere Application Server can be configured to use cookies, URL rewriting, or both to maintain sessions across multiple HTTP requests. With both mechanisms WebSphere creates a Session ID to identify a session uniquely. With cookies, this Session ID is sent back to the browser as a field inside the cookie. With URL rewriting the Session ID is appended to the URL and sent back to the browser. During subsequent requests the browser sends the Session ID back to the server as part of the cookie or the URL. The server is responsible for retrieving this Session ID from the HTTP request and using it to obtain the proper HttpSession for this user.

Early in the high-level design phase it is important to decide whether your application should be developed to support cookies, URL rewriting, or both. Such a decision should be made based on the demographics of the end users of the system. Some users configure their browsers to not accept cookies. If you suspect this may be the case, consider supporting the URL rewriting option. For the majority of applications we recommend using only cookies.

The WebSphere Application Server administration console provides a simple way to configure your applications to support either or both of these session management techniques. However, in order to maintain session state using URL rewriting the underlying servlets and JSPs must be coded so that every URL you send back to the browser is encoded with the session ID. This can be achieved by using the following techniques:

- Encode all URLs in servlets and JSPs

This can be done using the `HttpServletResponse.encodeURL(String url)` method. This method appends the Session ID to the URL that is passed as an input parameter. For example, if you have a servlet that does not support URL rewriting and has the following code:

```
out.println("<a href=\"exampleLink.html\"> Example Link <a>");
```

Then, in order to support URL writing replace all such references to URL links as shown below:

```
out.println("<a href=\"");  
out.println(response.encodeURL("exampleLink.html");  
out.println("\"> Example Link <a>");
```

- Use `HttpServletResponse.encodeRedirectURL(String url)` to encode all redirects.
- Do not include links to parts of your Web applications in plain HTML files. In essence, to maintain session state using URL rewriting, every page that the user requests during the session must be converted to JSPs or servlets. And all links inside such servlets and JSPs must be encoded using `encodeURL()`.

These requirements are necessary for URL rewriting to maintain session state, since all HTTP requests made by the user must have the Session ID appended to the requesting URL.

From this discussion, it is clear that the decision to support URL rewriting impacts the code development significantly. This decision also has certain performance implications, since all display pages including static pages have to be converted to JSPs or servlets. This adds unnecessary runtime processing. This also means that all static pages that could have been hosted by an information Web server now need to be moved to the Web application server node since all pages have to be converted to JSPs or servlets. Therefore it is important to decide early in the high-level design whether you plan to support URL rewriting.

7.7.2.2 Session persistence and clustering

Chapter 3 “Choosing the runtime topology” on page 19 discussed how Web application availability and performance can be increased by adding duplicate Web application server nodes to the runtime topology. This is achieved by using a load balancer to distribute Web requests across multiple Web application servers. Under such a scenario, if one Web application server fails, the load balancer would recognize this event and forward all the subsequent requests to the remaining Web application servers, which increases the overall availability of Web applications. Performance should be improved by distributing the load across multiple machines.

The above scenario can be extended to provide *failover* support. This is achieved by enabling WebSphere Application Server session persistence and session clustering.

When session persistence is enabled WebSphere Application Server stores all session data in a JDBC-compliant relational database such as DB2 or Oracle. This is achieved by inserting the session data (name-value pair) into the database as a result of an `HttpSession.putValue()` method and retrieving the same from the database as a result of `HttpSession.getValue()` method. The session persistence is automatically managed by WebSphere Application

Server. Application programmers need not write any special code for this session persistence to occur. However all objects that are being inserted into the session pool must implement the *serializable* interface.

Session clustering is a mechanism where more than one instance of the application servers share a common session pool. Essentially, a cluster is the binding of two or more application servers that reside on separate nodes. This allows servlets to execute on any one of these nodes and have access to session data that was created by another node. WebSphere Application Server exploits its session persistence feature to implement session clustering. Therefore, in order to enable session clustering session persistence must be turned on. Under this configuration, multiple application server nodes would share the common session database. This allows for session data created by one application server node to be accessed by another application server node during subsequent interactions. All changes to session data are committed to a common session database upon the completion of servlet execution. Hence, once the transaction is completed and the changes are committed. The session data is still accessible regardless of the failure of an individual node. This allows for complete failover support.

In designing systems that exploit session persistence and clustering features, we provide the following guidelines:

- Session persistence is implemented using a generalized persistence mechanism in order to allow for various types of information to persist in the session pool. Storing large amounts of data in such a generalized session pool could result in performance degradation. Hence the session pool must be used only to store data that is essential during subsequent transactions.
- All objects that must be propagated across the cluster along with the session must be serializable. We recommend implementing the *serializable* interface for all objects that you anticipate being stored in the session pool. This allows for an easy transition of your applications to a clustered environment.

Further details on session management issues can be found at:

http://www.ibm.com/software/webservers/appserv/doc/v30/se/web/doc/begin_here/index.html

7.8 Application Security

The WebSphere Application Server provides a robust security model for securing Web application resources. Key components of the security architecture include the security plug-in, security collaborator, security application and security server. The security model supports various authentication and authorization techniques. The authentication policy for performing authentication can be specified in terms of:

- User registry: where the user and group information is stored. WebSphere provides the following options:
 - LDAP directory
 - Native operating system user directory
- Authentication Mechanism: validates the authentication data against an associated user registry. WebSphere supports the following two options:
 - Lightweight Third Party Authentication (LTPA): This implies the use of an LDAP user directory.
 - Native operating system user authentication: This implies the use of NT, AIX, or Solaris user directory.
- Challenge Mechanism: specifies how a server will challenge and retrieve authentication data from the user. WebSphere supports the following options for challenge mechanisms:
 - None: Security runtime does not challenge the user for authentication data.
 - Basic: A user is challenged for user ID and password using the 401 error code. This results in the browser displaying the user ID and password dialog box.
 - Certificate: Mutual authentication over SSL using client and server-side certificates.
 - Custom: Works similar to basic challenge mechanism. However, instead of sending a 401 error code, WebSphere redirects the user to a logon HTML page. The URL for this logon HTML page can be set by the system administrator using the WebSphere security configuration options.
- Secure Channel Constraint: specifies if an SSL session is required while passing the authentication data from the browser to the server.

9.2.4, “Web application security” on page 211 describes the overall WebSphere security architecture, explaining how various security

components interact with one another, and discussing configuration options for securing Web resources.

Enabling basic and certificate challenge mechanisms described above primarily involves configuring WebSphere security options. Such a setup does not involve developing specialized login programs for the Web application. Custom challenge mechanism on the other hand requires a specialized login program in addition to security configuration. This section explains best practices for designing custom login servlets.

Custom challenge type can be characterized by the following flow:

- A user who is yet to be authenticated issues a request to access a secure Web resource that is configured to use custom login challenge type.
- The user is redirected to a URL configured as LoginURL. To perform a form-based login, this URL points to an HTML file containing an HTML form requesting a user ID and password. The system administrator is responsible for specifying the LoginURL during security configuration.

A sample login.html page is listed below:

```
<FORM METHOD=POST ACTION="/servlet/CustomLoginServlet">
Please Enter:<br>
UserName: <input type="text" size="20" name="user ID"><br>
Password: <input type="password" size="20" name="password"><br>
<input type="submit" name="login" value="Login">
</FORM>
```

- Submitting the form triggers a servlet that performs a customized login, including authentication. To do this, the servlet can use the `ServerSideAuthenticator` helper class provided by the WebSphere framework. This class provides a login method that takes the user ID and password as parameters and performs the authentication using the specified authentication mechanism. On successful authentication, the user can be redirected to a Web page, such as a welcome page.

A sample `CustomerLogicServlet.doPost()` method that implements the custom authentication is listed below:

```

public void doPost (HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    String userID = null; // user id found on form
    String userPw = null; // password
    userID = req.getParameter("user ID"); // obtain user ID data form
    userPw = req.getParameter("password"); // obtain password from form

    ServerSideAuthenticator authenticator;
    authenticator = new ServerSideAuthenticator();
    org.omg.SecurityLevel2.Credentials retCreds = null;

    //Perform Login
    try {
        retCreds = authenticator.login(userID, userPw, true);
    } catch(Exception e) {
        throw new ServletException();
    }

    //Setup Single-Sign On Cookie
    SSOAuthenticator ssoAuth = new SSOAuthenticator();
    ssoAuth.login(userID, userPw, req, res);

    try {
        res.sendRedirect(redirectURL);
    } catch(Exception e) {
        throw new ServletException("Error redirecting to URL" +redirectURL);
    }
}

```

Figure 49. CustomLoginServlet.doPost() method - source code

Usually the authentication state must be maintained across multiple Web requests. To do so, one can exploit the WebSphere Single Sign-On (SSO) framework. As shown above, a custom login servlet can be programmed to use the `com.ibm.websphere.security.SSOAuthenticator` class in the SSO framework. The `SSOAuthenticator.login()` method from this framework inserts an SSO cookie. Note, the SSO infrastructure requires LTPA to be the authentication mechanism. If a custom login is required, but if LTPA cannot be the authentication mechanism, then it is left to the application developer to write a servlet to maintain a cookie that contains the user ID and password.

On subsequent interactions, one can use the `HttpServletRequest` object passed into the service method of the servlet to obtain information about the user invoking the method. Invoking the `getRemoteUser` method on the

HttpServletRequest returns the name of the user. If the user is authenticated, the method will return the user name, whereas if the user is not authenticated, it will return the key word “anonymous.” The getRemoteUser method can be used as shown below:

```
public void doGet(HttpServletRequest req, HttpServletResponse res) {  
    // obtain the user name  
    String userName = req.getRemoteUser();  
}
```

7.8.1 Other design considerations

7.8.1.1 Prevent caching dynamic content

By default, browsers and proxy servers are configured to cache HTML pages in order to minimize the network traffic and improve performance by avoiding downloading recently retrieved pages. Such configurations work best with static content. Dynamic content on the other hand, would need to be downloaded every time the user requests such information. In order to prevent dynamic content caching, the best practice is to insert the following lines of code into the servlet doGet() or doPost() methods. Doing so informs the browsers and proxy servers to prevent caching the HTTP response under consideration:

```
public void doPerform(HttpServletRequest req, HttpServletResponse res)  
throws ServletException, IOException {  
    //Set Content Type  
    res.setContentType("TEXT/HTML");  
  
    //Prevent Caching  
    res.setHeader("Pragma", "no-cache");  
    res.setHeader("Cache-Control", "no-cache");  
    res.setDateHeader("Expires", 0);  
  
    //Perform the required action  
    .....  
}
```

7.8.1.2 Avoiding duplicate updates - handling the reload problem

When a user presses the reload button or resizes the page, the browser issues an HTTP request to the URL that generated the current response page. This results in the re-execution of the interaction controller logic and the regeneration of the display page. If the interaction controller under consideration in turn invokes a business logic method that performs an update, this results in the duplicate update of the model. For example, let us consider a discount brokerage application that provides a trade entry form. Let's also assume that the user enters the trade information and clicks on the

Submit button. As a result, a confirmation page is displayed. At this time, the user resizes the browser or presses the Reload button. As a result, the browser sends the trade entry request for the second time. This results in the re-execution of the trade. Clicking on the Back button could also result in such re-execution of the business logic. To overcome this business logic re-execution problem, we recommend using the following design pattern for all interactions that result in an update:

- Interaction controller (update HttpServlet) retrieves the input parameters and determines the type of update command to be executed.
- The interaction controller (update HttpServlet) then executes the selected update command.
- Based on the results, the interaction controller (update HttpServlet) creates a Result bean and inserts it onto the HttpSession pool.
- Finally, the interaction controller (update HttpServlet) invokes the sendRedirect(URL) on the HttpServletResponse object. This sendRedirect sends a redirect request to the browser. The browser in turn sends an HTTP request to the new URL sent by the sendRedirect command.
- The URL points to an interaction controller (display HttpServlet) that is capable of creating the final display page. As a result of the sendRedirect the display interaction controller (display HttpServlet) is invoked.
- The display interaction controller (display HttpServlet), retrieves the necessary parameters and selects the display page (JSP) to be invoked.
- Subsequently, the display interaction controller (display HttpServlet) retrieves the Result bean from the HttpSession pool. Using this Result bean it initializes the View bean associated with the display page (JSP). Using the HttpServletRequest.setAttribute() method, it sets the View bean into the request header.
- Then the display interaction controller (display HttpServlet) uses the RequestDispatcher to forward the request to the display page (JSP).
- The display page (JSP) retrieves the View bean stored in the HttpServletRequest object using the useBean tags.
- Finally, the JSP generates the HTML response page.

This approach to handling updates ensures that the update request is never inadvertently re-executed as a result of reloads which could happen as a result of pressing the Resize, Back, Forward, or Reload buttons on the browser. This is achieved by using the sendRedirect mechanism. As a result of the sendRedirect, the location on the browser is changed to the display

interaction controller URL. Hence on subsequent reloads the display interaction controller is executed.

7.9 Conclusion

In summary, this chapter introduced a number of user-to-business Web application design challenges, recommended design pattern solutions, provided examples that apply the recommended design techniques, and documented the advantages and disadvantages of the proposed solution. The design techniques presented are derived from the experience of many developers who have built high-quality, high-performance e-business applications. The techniques are presented as design patterns so that you can easily adapt them to your application requirements.

7.10 Where to find more information

IBM Publications:

- For information on the IBM Application Framework for e-business:
<http://www.ibm.com/software/ebusiness/>
- IBM Application Framework for e-business: Web Application Programming Model:
<http://www.ibm.com/developer/features/framework/framework.html>
- WebSphere Application Server Library:
<http://www.ibm.com/software/webservers/appserv/library.html>
- *Developing Dynamic Web Sites Using the WebSphere Application Server*, by Shane Claussen and Mike Conner:
<http://service2.boulder.ibm.com/devcon/news0399/artpage2.htm>
- AlphaBeans - JSP Format Bean Library Project:
<http://oss.software.ibm.com/developerworks/opensource/jsp/index.html>

Other Publications:

- Gamma, Erich et al. 1994. *Design Patterns Elements of Reusable Object-oriented* (Addison-Wesley Professional computing series), Addison-Wesley Publishing Company; ISBN 0201633612
- For information about the ECMAScript Language Specification:
<http://www.ecma.ch/stand/ECMA-262.htm>
- HTML:
<http://www.w3.org/MarkUp/>

- To learn more about Java technology:

<http://www.javasoft.com/products>

Chapter 8. Application development guidelines

The development of an e-business application does not differ very much from the development of any object-oriented, client/server application. However, there are some special considerations, which we will outline in this chapter. We will describe the development process used to build an e-business application from the start of the development project until its deployment in a production system. We will also show the usage and handling of the tools used to produce the development artifacts in the different development phases.

8.1 The development process

Today it is quite common in the industry to develop object-oriented software via an iterative and incremental process. This approach has different roots. For more information refer to the work of Grady Booch, *Object-Oriented Analysis and Design with Applications*; Ivar Jacobson, *Object-Oriented Software Engineering*; and James Rumbaugh, *Object-Oriented Modeling and Design*.

There is no defined standard process for development that everyone uses. Different teams typically adopt a recognized process using a vendor methodology or using their services team methodology. IBM Global Services has its own methodology used in customer engagements that covers the development process. The development process we follow throughout this chapter is simplified and more generic than the IBM Global Services methodology but is similar to it.

The process we will discuss is divided into different phases. Each phase is done in a sequential manner and is subdivided into further smaller phases. Some phases are only run through once. Others are done over and over again, forming the iterative and incremental part of the development process. The actual process and which phases you use might differ slightly depending on the development team or organization that uses the process.

We can divide the whole process into the following phases:

- Solution outline
- Macro design
- Micro design
- Build cycle
- Deployment

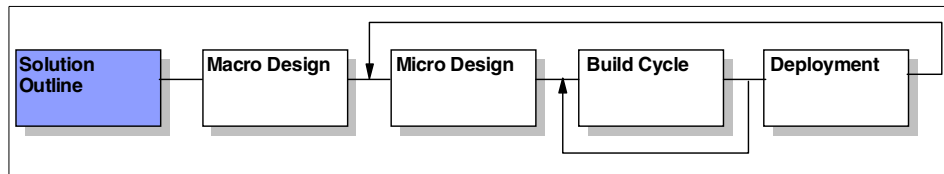


Figure 50. Development process overview

In the solution outline phase you decide the scope of the project, explore what the essential business needs are, come up with an idea of the base architecture, and get the commitment from the project sponsor to start.

Then you start with the macro design which concentrates on the detailed requirements gathering, business process modelling, high level analysis and design, the base architecture, and a plan for the following development phases, including a development release plan. These two phases are usually done once in a project.

Now the iterative and incremental part of the development starts. For each release of the developed e-business application, the micro design, build cycle, and deployment phases are completed. Usually, a certain set of use cases that have to be developed to meet a part of the system requirements make up a release. The releases are defined in the project plan produced in the previous phase. A release can be an internal one that is not deployed to any users. This is quite common for early stages of big projects. Others, like alpha or beta releases, might be deployed to a certain number of test users. It might take several iterations until a first official release of the application is deployed to the users. In turn, there are often several releases to the users until all requirements are met, plus maintenance releases to fix errors and other defects.

8.2 The scope of this chapter

In this chapter we will explain the development process used for e-business applications and will focus on:

- The results
- The process to get those results
- The tools to produce those results

We chose this structure since some process models, like the IBM Global Services methodology, suggest this approach. The idea is to drive the whole

process from the results, called work products. The process is divided into phases as explained before. Each phase is divided into activities. In turn each activity might include several tasks. Each phase, activity or even task produces particular work products as output and needs others as input. The IBM Global Services methodology also provides technical papers (techniques) that describe how to produce the different work products.

This chapter will give practical advice on how to use certain tools, like Rational Rose, IBM WebSphere Studio, and IBM VisualAge for Java, to produce a work product. But it is also possible to use the described development process and produce the required work products with other tools.

In Chapter 7, “Application design guidelines” on page 81, we introduced some examples taken from a problem domain where weather information can be obtained for the solar system. The same problem domain is used in the Pattern Development Kit (see Chapter 10, “The Pattern Development Kit and an example topology” on page 231 for more information). We will use those examples or at least use the same problem domain, as the Solar System Web application, in this chapter.

8.3 The application and architecture domains

As we mentioned before, we do not explain a specific methodology in this chapter. We will describe the process to create the different work products in a general way, but not go into the work breakdown structure that the IBM Global Services methodology provides. However, the IBM Global Services methodology provides a domain concept that will help to position the contents of this chapter.

A domain is a logical grouping of related work products. Different roles and skills fit into a domain through enabling specialization. Domains build the basis for method tailoring which is an important part of the IBM Global Services methodology.

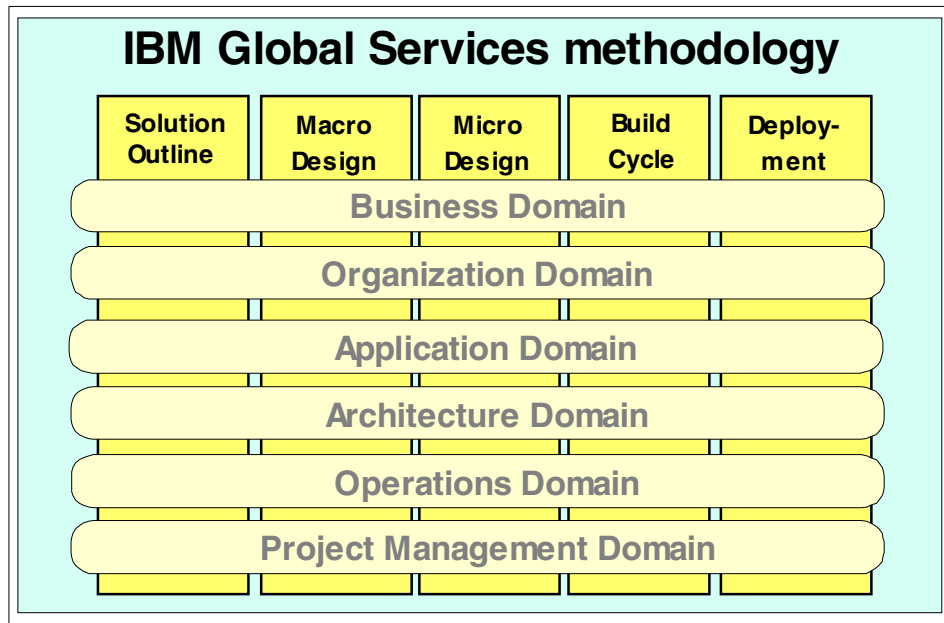


Figure 51. Domain concept in IBM Global Services methodology

Domains span the whole development process. There are six top level domains. We will concentrate on the application and architecture domains in this chapter. This does not mean that you should not be concerned with the other domains; for example the use case model work product in the application domain is dependent on the business process model work product from the business domain.

8.4 Solution outline

The first phase of a development project is the startup. This phase normally begins with a small team of domain experts, analysts and IT architects exploring the requirements for the new solution. Beyond the pure business requirements, it is important to explore the existing environment to find out how the new application can fit. The target audience for the solution has to be named and their experience has to be determined.

Based on all this initial information an architecture for the application has to be determined. The team has to decide about the overall strategy of the solution, that will drive the whole project, based on the business impact the solution has on the organization.

In the process of making these base architectural decisions the team can use the assets of the Patterns for e-business:

- First choose a business pattern that best fits your business problem. The Patterns for e-business home page under <http://www.ibm.com/software/developer/web/patterns/> will help you in making this decision. For purposes of this discussion, the chosen business pattern is user-to-business.
- Next review the application topologies for user-to-business and choose the appropriate one.

There are two different base strategies, depending on the organization, business domain, and planned solution:

- Web-up, where the premise is to build a Web application from scratch.
- Enterprise-out, where the idea is to Web-enable an enterprise application, meaning to build an additional Web channel to access an enterprise application.

Our example is a Web-up strategy because we want to build a new Web application with new business logic, leading us to the two application topologies described in Chapter 2, “Choosing the application topology” on page 11.

The architectural decisions made are a separate work product and should be well documented. They are used as input for the macro design where the architecture is driven from the chosen logical application topology.

8.5 Macro design

In the macro design phase the project team is usually extended from the few domain experts, analysts and IT architects working in the solution outline phase to a broader skill set. This team works on the refinement of the results from the solution outline phase. Their task is to do the following:

- Refine the requirements to come up with the business process model by identifying and describing key business use cases.
- Evaluate various technology options available for the implementation. At this point you need to choose a set of technologies for application development. Chapter 6, “Technology options” on page 65 helps you make this decision.
- At this stage it is also important to plan the deployment model. For that reason it is important to finalize the operational model. Chapter 3, “Choosing the runtime topology” on page 19 helps you choose a logical

operational architecture. Since such decisions have an implication on the long-term system management of the deployed application, we urge you to consider the guidelines provided by Chapter 9, “System management products and guidelines” on page 197.

- Subsequently the logical operational architecture needs to be mapped to a physical instantiation. Chapter 4, “Product mapping” on page 43 provides guidance in achieving this.

As in the solution outline phase, you should document your architectural decisions as work products because they will serve as input for the following release cycles.

Other tasks that have to be done in this phase include:

- Design and plan the tests.
- Set up the development environment.
- Create a development plan for the following release cycles.

Based on a chosen business pattern and application topology you can start with media design, GUI prototyping and the information architecture. These activities help you understand what the exact business requirements are.

There are four different approaches for these tasks:

- For Web-up:
 - Starting from an existing Web site as your prototype
 - Starting from an abstract application flow model using techniques like storyboards
- For enterprise-out:
 - Starting from an existing third-tier system
 - Starting from a static object model, usually taken from a third-tier system

Traditional applications usually present only structured data. Web applications on the other hand have to combine structured data and unstructured information on the same page. For example, an online brokerage application not only needs to show the stock price, account balance, etc., but also has to show research information about these stocks. This need to combine the structured and unstructured information on the same application presents some unique challenges. For this reason, early in the macro design phase it is important to develop an Information Architecture for the Web application. Creating an information architecture helps in understanding:

- What information has to be published.
- Which users are allowed to access that data.
- What experiences the potential users have, and the different roles they can play when accessing the information.
- How the information is accessed. For example, is the client a browser, a stand-alone application, a Personal Digital Assistant (PDA) or a cell-phone?

Based on the information architecture, the business scenarios for the Web applications can be modelled using storyboards. These scenarios can then be implemented in GUI prototypes to verify if they meet the business requirements. Furthermore, the GUI prototypes and media design are used to find the right look-and-feel for the Web application.

The release cycles start after all architectural decisions have been made and documented.

8.6 Micro design

The development plan outlines several release cycles to implement the requirements iteratively and incrementally. After the macro design phase the requirements of a project are captured in different work products, like the business process model, domain use cases, domain class models, and interaction diagrams for those use cases.

Each release cycle starts with the micro design that focuses on transforming the business model into a design model by taking the selected use cases and running them through a typical object-oriented development phase. Transforming means that we use the business model to bring it to such a technically detailed level that it can be implemented. This is done by adding all the architecture and implementation-specific classes and components to the existing business model.

The design patterns explained in Chapter 7, “Application design guidelines” on page 81 are used for the transition of the business model to the actual design model. These design recommendations should lead you from your high-level application design model to a micro design model that is ready to implement.

The work products we produce in the micro design phase of the development process are:

- Use cases

- Class models
- Interaction diagrams
- State diagrams
- Component model
- Deployment model

Most of the work products are Unified Model Language (UML) artifacts. The UML specifications are defined by the Object Management Group (OMG, <http://www.omg.org>). Refer to the UML specification under <http://www.omg.org/uml>, and to *UML Distilled: Applying the Standard Object Modeling Language*, by Martin Folwer for further information on UML. We use UML to express the work products and show the examples in UML notation. We work with Rational Rose 2000 Professional J Edition as our modelling tool. Use the documentation and tutorials that come along with the Rational Rose product to find more details on how to use it. Refer to the Rational Rose home page under <http://www.rational.com/products/rose/>, to get the latest information on the product.

8.6.1 Use case

A use case describes a function that the developed system has to implement to meet a specific requirement. A function can be as simple as querying the current weather data of a planet in the solar system or it can be very complex, for example, journalling the weather information of the planets the user queries in a database to be able to show the user at logout a summary of all recorded data.

8.6.1.1 Use case: work products

We create a use case to describe a particular required user function (taken from the requirements) in a textual manner. The interaction of the user with the system is modelled in a use case diagram showing actors, systems, system boundaries, use cases, and the relationships between these elements.

In our solar system Web application there is a use case that covers the retrieval of historical weather information.

Use case: Retrieve historical weather information

The user specifies a date to retrieve historical weather information for planet Mars. The user also selects a measurement system, metric or imperial, for use in displaying the retrieved results. The user starts the request to get the weather information.

The system fetches the weather information for the given date. The system converts the fetched data according to the given measurement system. The system displays the converted temperature, humidity, and pressure along with the chosen date and measurement system to the user.

The description is kept very simple and is written in the active voice. A rule of thumb is to concentrate on what the actor does, his/her interaction with the system, and how the system reacts (the system's reaction to the user's requests).

Since this is a very simple example of a use case, the matching use case diagram is not very complex.

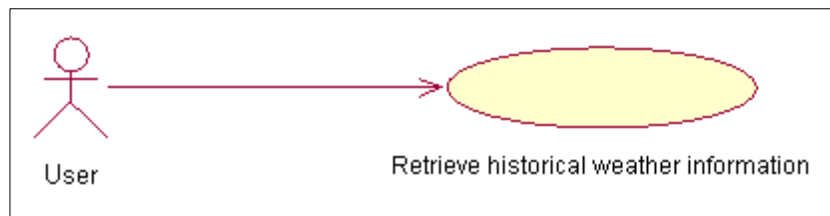


Figure 52. Retrieve historical weather information use case diagram

The actor, called User, interacts with the use case called Retrieve historical weather information.

8.6.1.2 Use case: process

Creating use cases for particular functions is a process that starts in the macro design phase of a project. It is often the case that there is already a domain use case that has to be divided into more detailed use cases to be able to capture all the required functionality. Not all requirements are found in the macro design so you may have to start from a requirement or a desired function and create a new use case from scratch.

At the creation of a use case a domain expert has to be part of the team working on the initial description to come up with a meaningful textual

description of the problem to be solved. It is also a good idea to consult a domain expert from time to time when a developer is working on the details of a use case to ensure that the actual user requirements are met.

8.6.1.3 Use case: tools

Rational Rose has a special container called “Use Case View”, to organize the use case model. All elements of a use case can be created in this view (actors, use cases, and use case diagrams) and these elements can be grouped together in logical containers that are sub-views of the Use Case View.

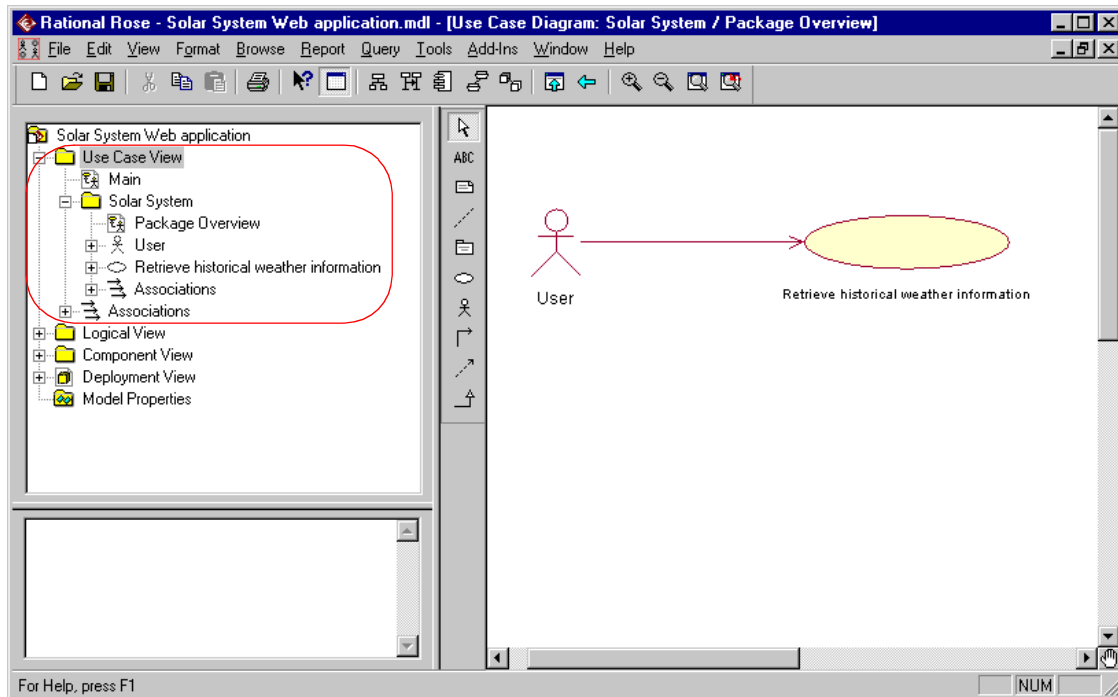


Figure 53. Rational Rose - Use Case View

With our simple solar system problem domain we have only created one sub-view under the Use Case View in Rose, called Solar System that holds all the use cases, actors, relationships, and use case diagrams.

Figure 54 shows the use case description of Retrieve historical weather information. It is found in the Use Case Specification window under the General tab in the Documentation text field.

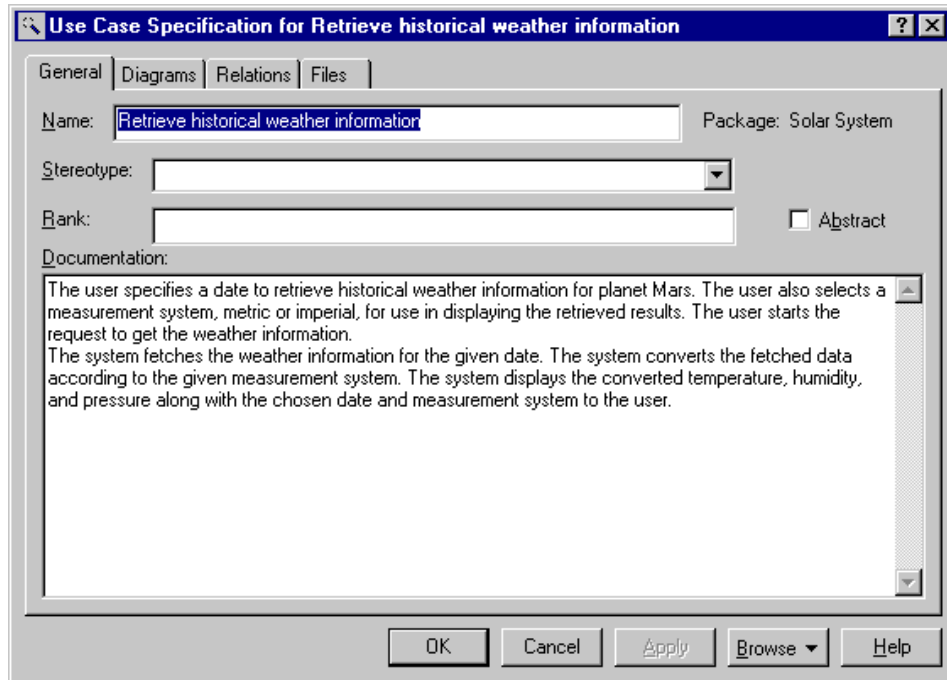


Figure 54. Rational Rose - Use Case Specification window

Some development teams prefer to have a template for use case documentation. This is not well supported by Rational Rose. If you decide to use a more powerful tool for use case documentation like a word processor, a link to that file can be kept in the use case specification. Use the Files tab in the Use Case Specification Window to create that link. Keep in mind that the file is not stored inside Rose and you have to maintain it manually.

8.6.2 Class model and class diagram

A class model contains all the classes found in the problem domain. The kind of classes that are represented depend on the development phase. In the analysis phase there are usually only the business model classes. In the design phase there is a more detailed model with not only domain classes, but also more technical classes like Java beans or servlets and implementation-specific objects like interfaces.

8.6.2.1 Class model and class diagram: work products

Class diagrams are used to capture the relationships between the different classes.

All the classes found in the micro design phase make up the class model. Each class is modelled with its attributes, methods, and relationships to other classes. We also document implementation language specifics. For example, class and method visibility in classes and role names and multiplicity of relationship in the relationship details.

The class relationships are modelled in one or more class diagrams. If there are many classes, it is a good idea to group the related classes in separate class diagrams. Classes that form a logical or functional unit should be shown together. It is often a good idea to show a class in different class diagrams explaining different relationships for specific functionality.

In a class diagram we express the static relationships of the classes, for example, inheritance, association, dependency, and implementation relationships. The most important attributes and methods may also be shown in the diagram if it helps express the described function.

For our solar system Web application example, we will show all the classes that are needed to implement the Retrieve historical information use case with their static relationships in Figure 55.

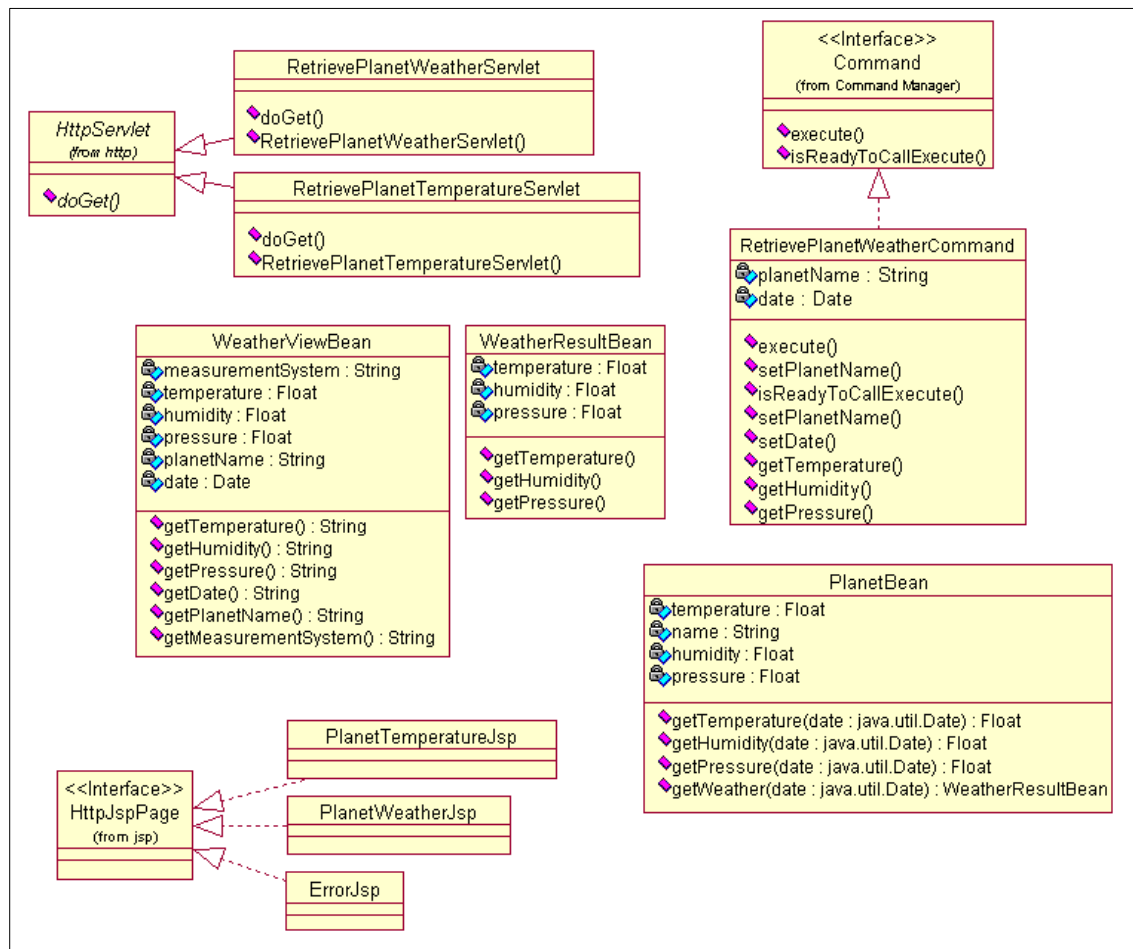


Figure 55. Solar system weather information - overview class diagram

Since we are in the micro design phase the diagram shows all the technical details that we will use to implement the use case. In 7.1, “Application elements” on page 82, the Model-View-Controller design pattern is discussed. The elements of our application relate to the MVC pattern in the following way:

- **Controller**

The RetrievePlanetWeatherServlet and RetrievePlanetTemperatureServlet are responsible for controlling the interaction of the user with the system.

- **Model**

- PlanetBean and RetrievePlanetWeatherCommand implement the business logic and data access.
- WeatherResultBean wraps the results of a weather information query on one object, by implementing the contract between the controller and the business logic.

- **View**

- PlanetWeatherJsp, PlanetTemperatureJsp, and ErrorJsp are the page constructors responsible for displaying the appropriate result information to the user.
- WeatherViewBean wraps the results of the weather information query and provides a certain view of this data (for example, a user-preferred measurement system) by implementing the contract between the controller and the view.

Figure 56 shows the dependencies between the different classes that implement the Retrieve historical weather information use case.

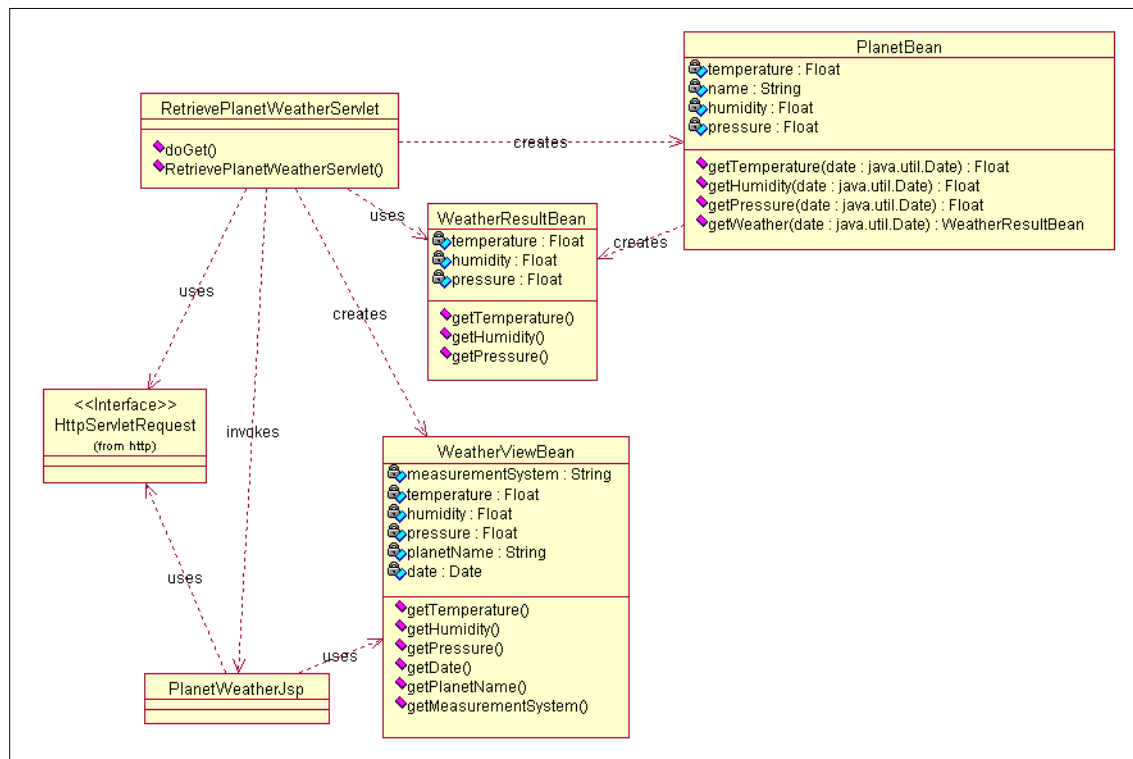


Figure 56. Solar System Weather Information - dependencies class diagram

The dependency relationships shown result from the interaction between the different classes.

The interaction controller, RetrieveWeatherInformationServlet, creates the model PlanetBean which in turn creates and returns the Result bean, WeatherResultBean. The servlet then creates the WeatherViewBean with the properties of the Result bean and with the information about the date and measurement system taken from the HttpServletRequest helper class, storing it in the HttpServletRequest for the view. Finally it invokes the PlanetWeatherJsp page constructor that uses the View bean taken from the HttpServletRequest to display the weather information in the appropriate measurement system to the user.

For more information on Result and View beans, see 7.4.2, “Result bean and View bean design pattern example” on page 107.

8.6.2.2 Class model and class diagram: process

After the macro design phase there is a business model containing all the model classes which is used as an immediate input work product. When the business model use cases are further detailed and new use cases are created in the micro design you will find many new classes in the use case descriptions. New technical or implementation-specific classes will appear in the design since we are transforming a business model into a design model, introducing a new level of implementation details.

Starting from the design use cases, using methods like CRC (Class Responsibility Collaboration) cards to find the class's responsibilities (methods, relationships, and attributes) is a good way to come up with an initial model. Later, in the ongoing process of the micro design we will refine the class model, especially when modelling the interaction and state of the classes.

8.6.2.3 Class model and class diagram: tools

A special view, called Logical View, is used in Rational Rose to hold all class models.

We organize related classes in groups that are packages inside the Logical View. These logical packages should contain classes that, when grouped together, form a functional unit or are used to implement a use case. A good organization of the classes already in the Logical View is especially important when the project is big, or else you can lose the overview. Base classes, such as the Java class libraries, that your classes depend on, should also be incorporated into the model.

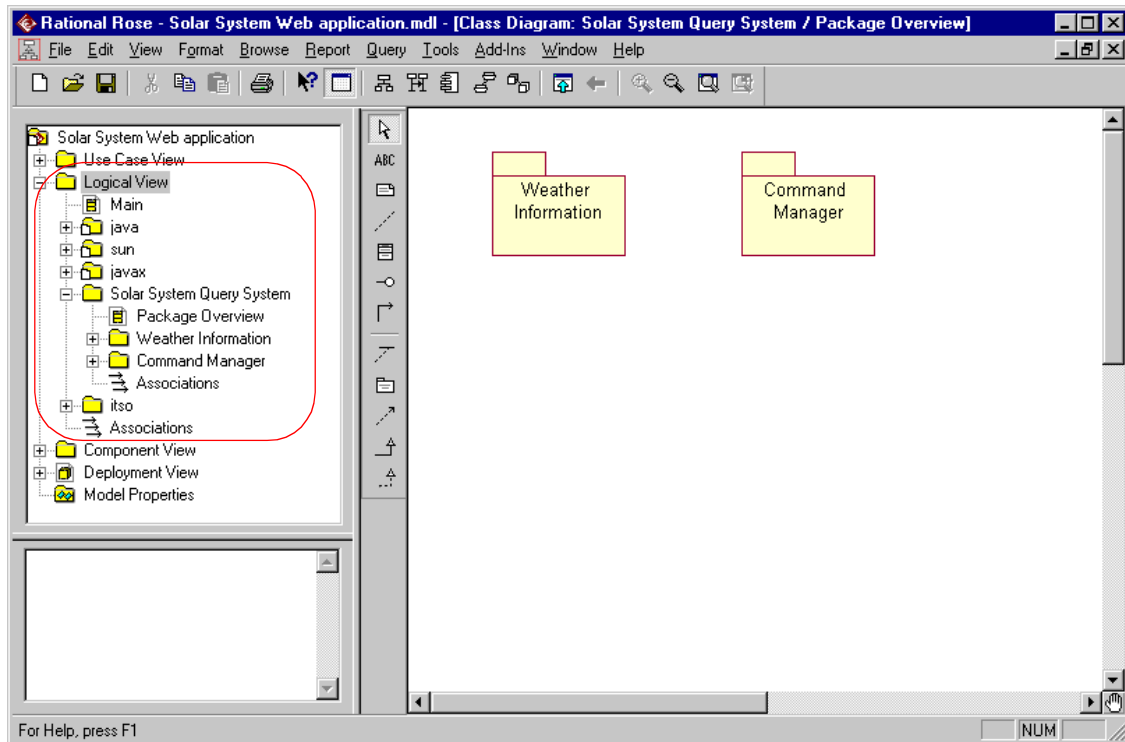


Figure 57. Rational Rose - Logical View

In Figure 57, we have created a group called Solar System Query System that holds all the classes we have identified throughout the design process. This group, in turn, contains other logical packages like the Weather Information package. The Weather Information package holds all the classes and class diagrams we described above.

8.6.3 Interaction diagram

For each use case there is usually at least one interaction diagram that outlines the main interaction flow for the classes involved. If a use case covers several scenarios it is likely that there are several interaction diagrams. All the important dynamic issues of the classes should be captured in interaction diagrams.

8.6.3.1 Interaction diagram: work products

An interaction diagram shows the end-to-end flow of messages between different objects that collaborate to fulfill some function, for example, displaying a planet's weather information for a certain date. End-to-end flow

means that the message flow is shown from its trigger until its completion. The action is usually triggered by an actor, system or system boundary. The involved objects are shown with the message they send.

Depending on the stage of the design phase and the level of detail you want to show, the accuracy of the diagrams will differ. In early stages of the design we only use the most important objects to show their interaction. The messages we show are not language-dependent and may not even be methods implemented by the receiver class, meaning that we use normal textual descriptions on the message arrows. To be able to use interaction diagrams as documentation for the build cycle, the diagrams should be made more accurate. It is acceptable to leave details out, but we use only the classes which are really involved and the messages sent which are implemented by their receivers. To be able to generate code from the class model at the start of the build cycle it is important to have good and detailed interaction diagrams with all the important methods identified and documented.

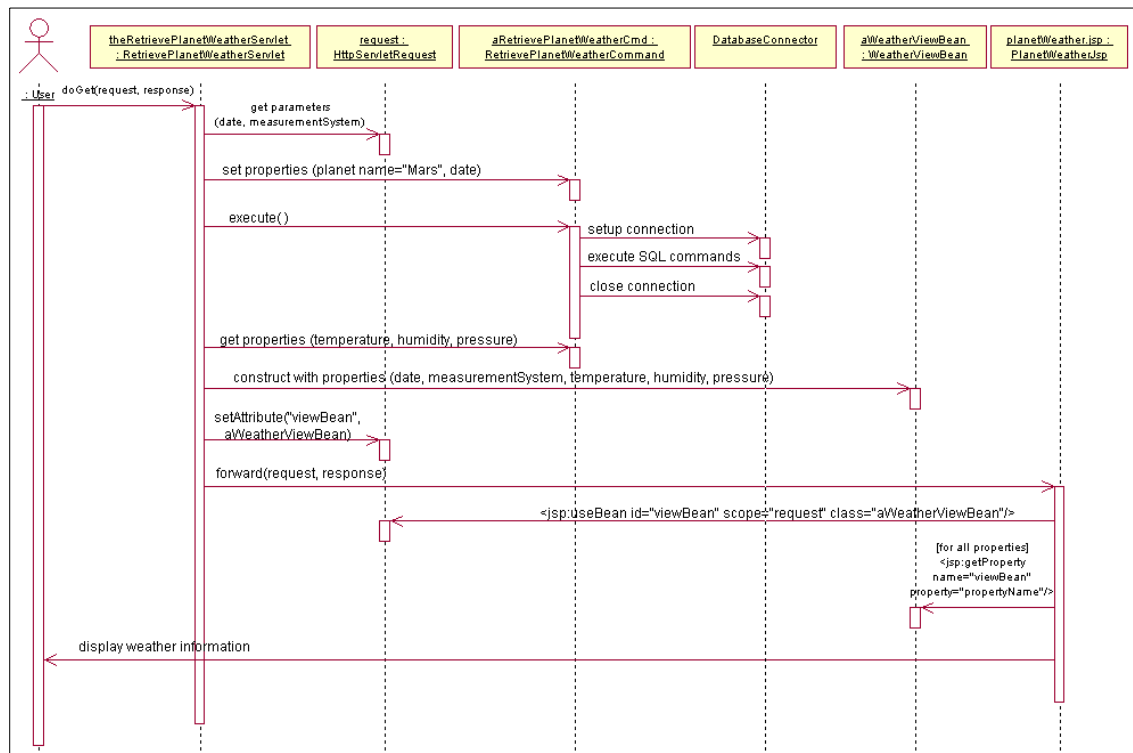


Figure 58. Solar system weather information - interaction diagram

Figure 58 shows the interaction between the classes of our Retrieve planet weather information use case with the technical details on the data retrieval:

- The RetrievePlanetWeatherServlet controls the whole message flow.
- The RetrievePlanetWeatherCommand encapsulates the business logic and data access. (More information on Command beans can be found in 7.6.1, “Command beans” on page 118.)
- The DatabaseConnector represents the connector technology used to access data in a database.
- The PlanetWeatherJsp is the page constructor invoked by the servlet using the View bean PlanetWeatherViewBean to present the results in the preferred format to the user.
- The HttpServletRequest is used for data exchange between the servlet and the JSP.

When you create an interaction diagram, you will use the classes from the existing class model as input. But, since modelling the interaction between these classes often assists in determining their responsibilities, the interaction diagrams are also used as input work products to update the classes and diagrams of the class model.

8.6.3.2 Interaction diagram: process

A use case has one or more scenarios, or flows, that a user can follow through the functionality described by that use case. It is a good starting point to capture these scenarios in interaction diagrams. At the very least, all important interactions that show the overall picture of a use case should be created at the beginning of the design. The refinement of existing diagrams and creation of new ones should go on until you feel comfortable with the understanding of the system modelled in the interaction diagrams. In other words, stop modelling when you think you are able to start with the build cycle.

8.6.3.3 Interaction diagram: tools

We use interaction diagrams as part of the Logical View in Rational Rose to describe the interaction of the classes. If the diagram is specific to a certain logical package you should place it inside that package.

When working on an interaction diagram and finding new messages sent between the participating classes, put those messages in as methods in the receiver class. Also, try to capture the dependencies between the classes that result from the different message invocations in the class diagrams (see Figure 56 on page 161 on how we did this for the Solar System example).

This will help create a good class model that includes all the necessary responsibilities to implement the required functionality.

8.6.4 State diagram

Some classes have complex internal state behavior. For those classes, state diagrams are a useful representation to capture that complexity.

8.6.4.1 State diagram: work products

A state diagram describes the internal states and the state behavior of a class. The input for this work product includes the class model, the use case, and the matching interaction diagrams that describe the state behavior of the classes. This work product might in turn influence the responsibilities of the class whose internal state it describes. For example, methods and attributes or the class' interaction with other classes (modelled in interaction diagrams) will change depending on the findings of the created state diagram.

8.6.4.2 State diagram: process

Some classes that are found and modelled in the design phase have very complex internal state behavior that must be described in order to understand the class. Use a state diagram to capture these states. State diagrams are also very helpful while actually implementing a class.

8.6.4.3 State diagram: tools

In Rational Rose, state diagrams can be created for any class. They are held in the Logical View under the class they belong to.

8.6.5 Component model

A component model contains the software modules that, when put together, make up the developed application. A component maps one or more classes or even a whole package from the class model into the appropriate language-specific component. For example, with Java as the implementation language, you have to map the classes of the design model to class packages in the component view.

In the build cycle, the component model is used to map the logical design model into the actual implementation model (writing source code). A class can be assigned to various components but those components must have the same implementation language.

8.6.5.1 Component model: work products

Create components for all classes to map the design model to the implementation model.

Depending on the complexity of the application and the number of involved classes and packages in the logical class model, use one or more component diagrams to show the mapping of classes to components and the dependency between those components.

Since the implementation language for the e-business applications described in this book is Java, HTML, and some other scripting languages, like JavaScript or Java Server Pages (JSP), our example will map the class to those implementation languages.

The component model is very important for the build cycle, since the initial source code, usually Java, is created from components. For Java this means you have to create packages in the component model that correspond to the actual Java package you want to create and map the model classes to components that will eventually be used to generate Java classes.

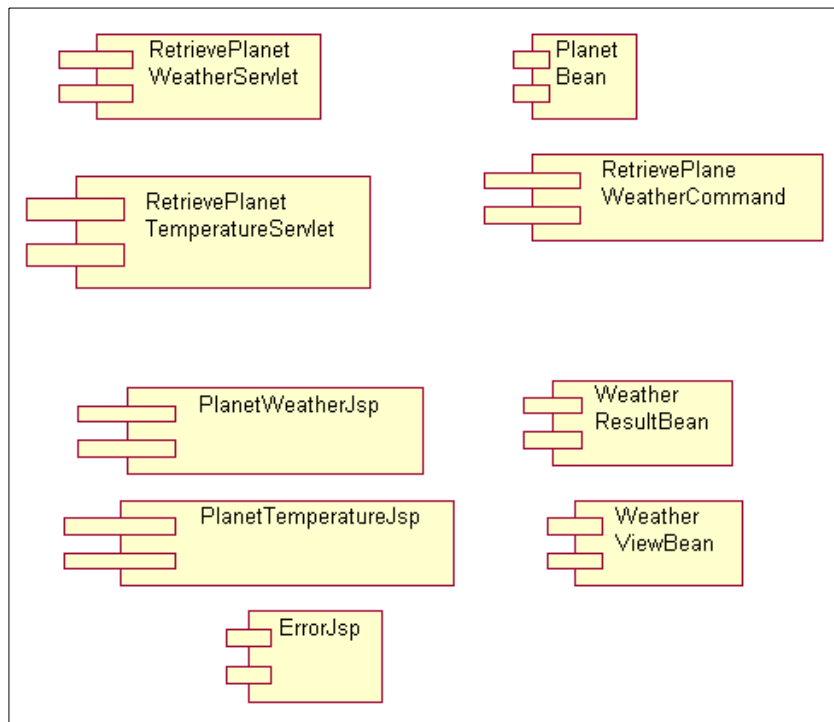


Figure 59. Solar system weather information - component diagram

Figure 59 contains all the components we have created for the solar system weather information retrieval example in the physical package called weatherinformation. We made a one-to-one mapping between classes and

components. In the package structure we chose, the weatherinformation package is placed under the solarsystem package which is the topmost package of our application and holds all the application-specific packages. This package in turn is placed under the itso package, which we used to contain all the applications developed within the ITSO organization. This structure can be seen in Figure 60 on page 169.

8.6.5.2 Component model: process

The component model should not be created before the class model reaches a rather stable state; otherwise, changes to the class model often lead to much rework in the component model.

The component model is usually only needed at the end of the design stage when you start the implementation by generating the initial source code from the component model.

8.6.5.3 Component model: tools

The Component View is another special container in Rational Rose. It holds all components and packages (in this case, Java class packages) that are used to map the class model to the implementation model.

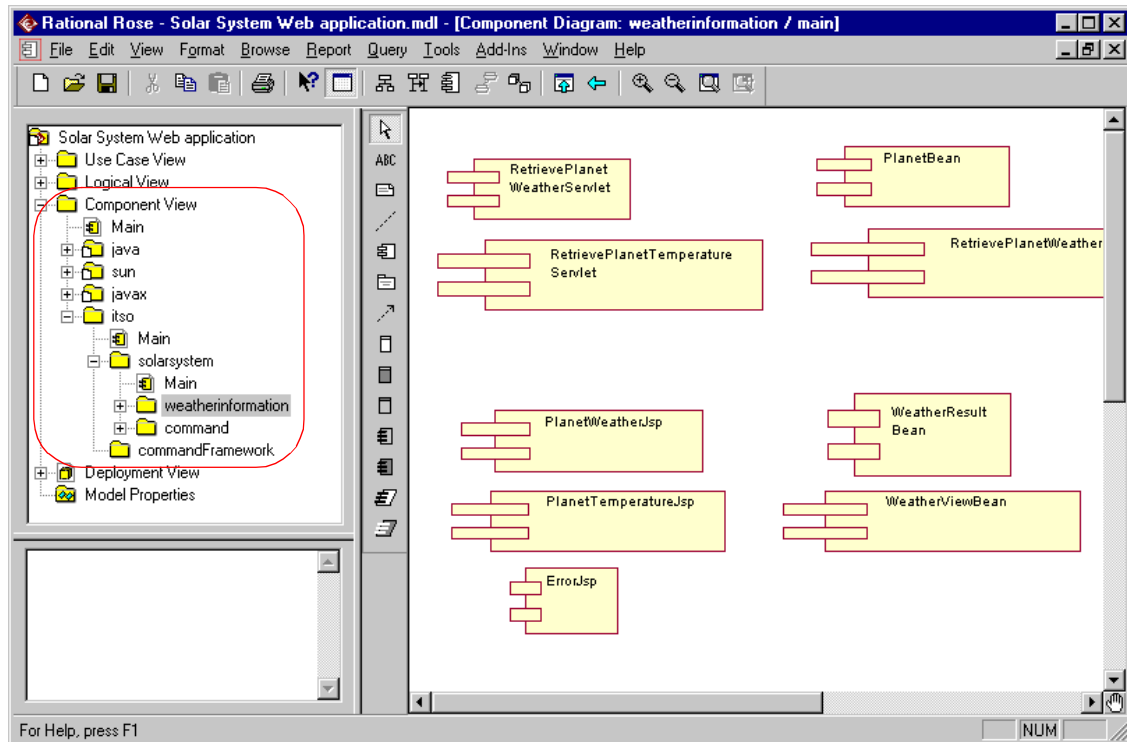


Figure 60. Rational Rose - Component View

As discussed earlier, we use the Component View to organize the physical package structure of the solar system Web application as shown in Figure 60.

The components and packages in this view are used for Java code generation. Code generation will be discussed later in 8.7.1.3, “Develop source code: tools” on page 179.

8.6.6 Deployment model

A deployment model contains the runtime characteristics of the system. The runtime application is deployed on various processes and devices. These processes and devices together with their relationships are captured in one deployment diagram.

8.6.6.1 Deployment model: work products

A deployment diagram is used to explain the deployment scheme of the application at runtime. In addition to this information, it can be used to show the mapping of the components to the processes.

We use this kind of diagram, shown in Figure 61, to show the deployed runtime topology of the developed e-business application. For the user-to-business pattern topology 1 this is quite simple since there are only two logical layers, presentation and business logic. Application topology 2 can be more complex, because it contains a third tier that can include multiple back-end systems or third party applications.

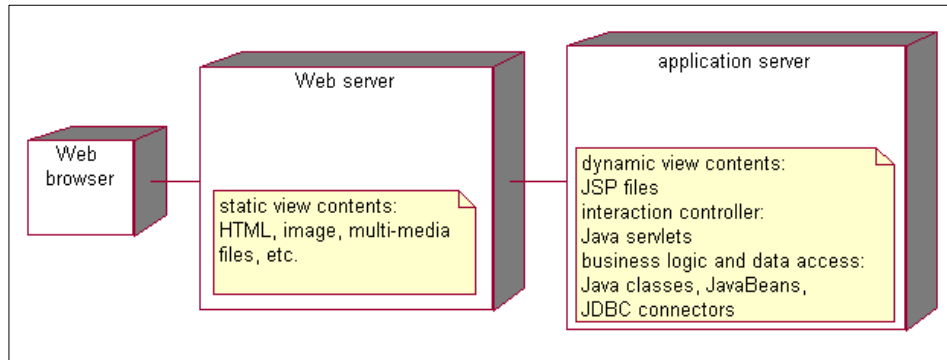


Figure 61. Solar system weather information - deployment diagram

For the solar system Web application we have a quite simple deployment diagram showing three processors and the components running on those processors:

- Web browser:
The user works with a Web browser to access the solar system Web application.
- Web server:
The Web server is accessed by the client Web browser and serves static view contents, such as HTML, image and multimedia files. It passes any client access requests for dynamic contents to the application server.
- Application Server:
The application server serves dynamic contents:
 - View: the page constructor implemented with JSP technology
 - Controller: the interaction controller as Java servlets
 - Business logic: using JavaBeans and Java classes
 - Data access: using the JDBC connectivity built into WebSphere Application Server using the appropriate Java connector

8.6.6.2 Deployment model: process

The deployment model is not very important before you plan for the deployment phase. But it can be useful to start to model the deployment plan even before you start the implementation to ensure that the developed components can actually be deployed with the planned runtime topology.

8.6.6.3 Deployment model: tools

Rational Rose provides a special view, called Deployment View, that holds all deployment objects, processes and devices. There is one deployment diagram that shows all the devices and the different processes of the running system and how they are related. The deployment model is not used for Java source code generation.

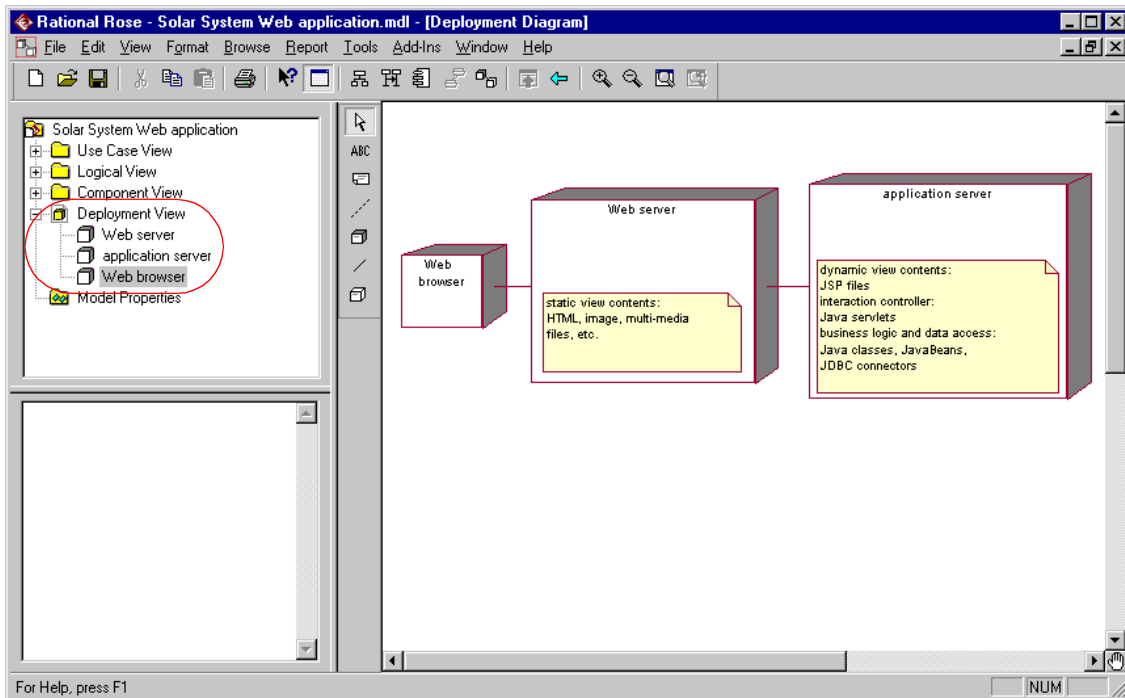


Figure 62. Rational Rose - Deployment View

The Deployment View for the solar system Web application is relatively simple, because we only have three nodes that we show, the Web browser, Web server, and application server.

The UML specification suggests an approach to map the components to the deployment nodes. In the future we will see tool integration for this option. For

now, we decided to use our own format to explain our runtime topologies (see Chapter 3, “Choosing the runtime topology” on page 19).

8.7 Build cycle

In each release cycle the micro design is followed by the build cycle. The designed system is actually coded and tested in several build cycles. As in the micro design, each build cycle focuses only on the requirements valid for that particular release. So with every build cycle the developed system is growing in the functionality implemented.

In the build cycle the results of the micro design are turned into code:

- Write and unit test the source code.
- Build the executable code if necessary, for example, all Java code.
- Perform various tests on the executable code.
- Test the application in a runtime environment.
- Prepare for deployment.

The incremental approach used to run the release cycles is also used for the different activities of the build cycle. It is run in several iterations for one release, with each iteration transforming more of the design into tested executable code that is ready to be deployed.

In this section we will focus on the process of building and testing. Be sure to refer to Chapter 5, “Performance guidelines” on page 55 while implementing the source code and when executing your stress and performance tests.

For more detailed information on the following topics including using WebSphere Studio, VisualAge for Java, and source code management, refer to *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00.

8.7.1 Develop source code

We use the work products of the previous micro design phase as input for the coding phase. For Java, follow the guidelines outlined in Chapter 7, “Application design guidelines” on page 81 and in 5.3, “Java and Java Virtual Machines” on page 60.

8.7.1.1 Develop source code: work products

Both user-to-business application topologies described in this book suggest a layered approach, where the presentation logic is divided from the business

logic. The application design detailed in Chapter 7, “Application design guidelines” on page 81, suggests using the Model-View-Controller (MVC) pattern for the separation of concerns. We divided the description of the required work products into three sub-sections that reflect the use of this pattern.

Controller source code

Interaction controllers coordinate the application flow by accepting requests from view elements, like HTML pages, invoking the required business logic from a model element, for example, a Java bean, and invoking a dynamic view element to show the results to the user, for example, a JSP page.

To implement a controller, a Java servlet or a JSP can be used, depending on the complexity of the interaction that is controlled. Refer to 7.3.1, “Model-View-Controller (MVC) design pattern” on page 94 for more details on when to use what.

View source code

View elements are implemented using different techniques. They reside on the Web server or Web application server, and are invoked from thin browser-clients.

For static contents, Web pages are built using hypertext markup language (HTML) along with multimedia contents, like images, audio, and video files. Client-side scripting, for example, using JavaScript, can be incorporated into static pages to perform active tasks on the browser client. Examples of such a task would be to check the input values of an HTML form or to pop up a help information window when a help button is pressed.

When the contents of a page change often, possibly determined by user input, dynamic page construction techniques are used. We call these view elements page constructors. Java Server Pages (JSP) are used to construct dynamic contents on the server. For example, in our solar system application, a JSP is used to construct the page showing the results of a weather information query on a planet or the journalling results of a specific user collecting weather information on different planets.

Model source code

The business logic is implemented using Java classes, Java beans, or enterprise Java beans (EJBs), depending on the complexity and purpose of the particular model element.

Connectors are used to access data, local or remote databases, or other applications residing on the third tier. Connectors are Java class libraries that

provide an application programming interface (API) to allow easy access to databases, middleware, or back-end systems like JDBC-accessible databases, MQSeries, IMS or CICS.

Third tier source code

As explained in 2.2, “Application topology 2” on page 14 there might be a need to add new components to the third tier to be able to access legacy systems or third-party applications. It is also possible that these back-end applications will have to be changed to integrate them with the newly developed Web application on the second tier.

The work products needed to accomplish this integration task may differ very much depending on the actual system to be integrated. These tasks may include things like having simple batch programs run on a back-end host, enabling database access to a host database, MQSeries integration of back-end systems, or invocation of new IMS or CICS transactions.

8.7.1.2 Develop source code: process

An e-business application is developed by a multi-disciplinary team. The skills include graphic artists, Web page designers, client and server-side script writers, Java programmers, and traditionally skilled programmers. Depending on the size of the e-business application there might be multiple team members with each skill or multiple roles performed by one team member.

Whether there is only one person on the team or one hundred, the concept of the separation of roles and responsibilities is key to the successful development and maintenance of an e-business application.

First let’s have a look at the overall process of source code development and the involved roles (skills). We outline the application flow here for both topology 1 and topology 2 applications, and show the different kinds of source code work products described earlier.

In Figure 63 on page 175 and in Figure 64 on page 176, the dashed arrows show what components are produced by each role, the dotted arrows show what components are consumed by each role, and the straight arrows show the application’s control flow between components.

Application topology 1

Figure 63 illustrates the basic structure of application topology 1 and how the components created by each role interact.

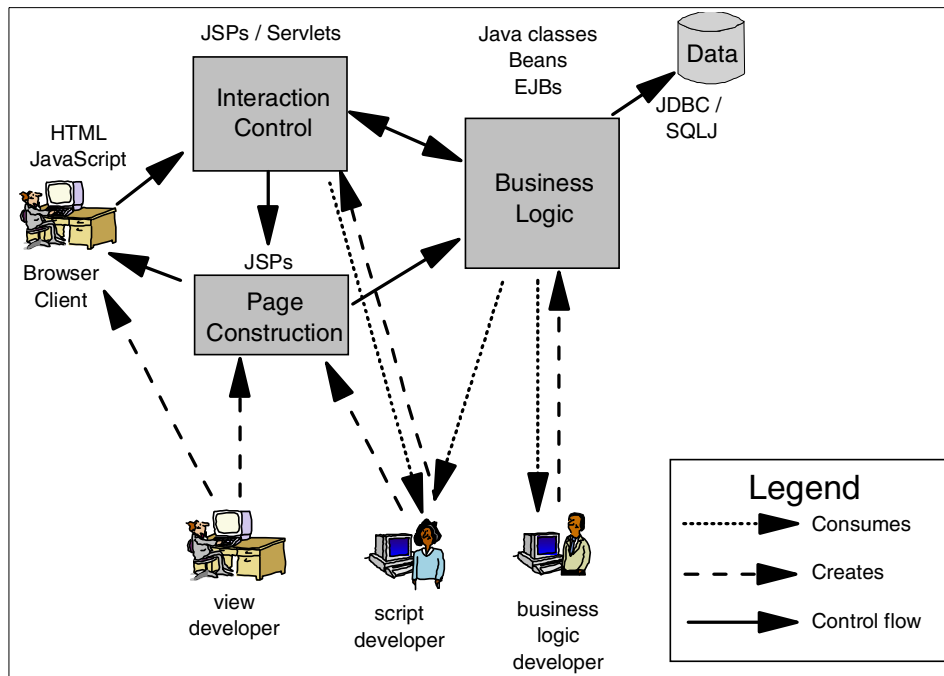


Figure 63. Application topology 1: process of source code development

HTTP clients communicate with an interaction controller. The interaction controller invokes some business logic depending on the user's request, for example, a request to obtain a query result or to perform an update operation. When the business logic has executed, the response page is generated using the dynamic page construction feature of the Web application server.

Variation on application topology 1

For simple e-business applications that do not have much business logic, the process shown in Figure 63 can be modified. The role of the business logic developer is not needed because there is no complex business logic to build. This task can be performed by the script developer, who uses tools to generate the model beans that implement the business logic. This developer also has enough skill to write simple Java code inside the model beans.

Application topology 2

Figure 64 illustrates the basic structure of application topology 2 and how the components created by each role interact.

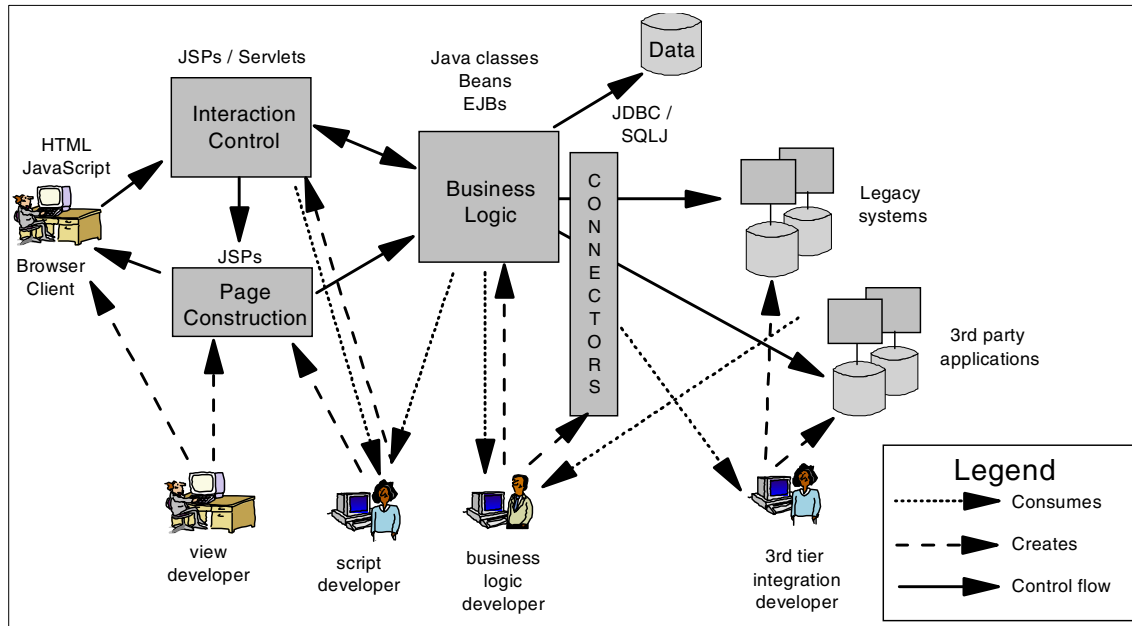


Figure 64. Application topology 2: process of source code development

The same basic application flow described for topology 1 applies to this topology. The difference is that connectors are used to access business logic or data on the third tier residing in legacy systems or third-party applications.

Roles

The roles used in Figure 63 and Figure 64, view developer, script developer, business logic developer, and third-tier integration developer, reflect the skills that a developer needs to create the respective component. In fact, a team member can perform more than one role, for example, a developer might produce view elements as well as writing scripts.

- **View developer:**

The view developer role is responsible for the view part of the presentation logic, usually consisting of display pages written in HTML with JavaScript, images and other multimedia elements included. A developer performing this role also generates simple page constructors (JSPs) using appropriate tools, like WebSphere Page Designer.

The view developer might also be involved in GUI prototyping.

The input for the view developer's work is the Result beans and View beans that have been created by the business logic and script developer respectively. These beans are used to construct the resultant pages.

- Script developer:

Complex dynamic pages that have to be coded fall into the responsibility of the script developer. This role also includes the creation of the interaction controllers. These can be implemented using JSPs or servlets. If servlets are used the developer has to have Java programming skills.

The script developer depends on both the view and the business logic developer. The script developer writes the code that controls the interaction between user request, business logic execution, and invocation of the result display page.

- Business logic developer:

The business logic developer is responsible for implementing the business logic, including data and third-tier access using skills in Java, Java beans, enterprise Java beans, and connector programming.

The business logic developer has to pay attention to the interface specifications of the code to ensure the script developer can safely call the business logic.

- Third-tier integration developer

If there are components to change or to add on the third tier, the third-tier integration developer is responsible for developing the required "glue" code to allow the integration of the new Web application with existing back-end systems or third-party applications. The skills needed can range from batch-job programming and database skills, to transaction application programming.

The business logic and third-tier integration developers have to stick to the defined interfaces and protocols in order to be able to develop their code independently.

The process of source code development encompasses the following activities:

- Writing the source code
- Testing the written code
- Transforming the source code to executable code
- Managing the source and the runtime code

These different activities are closely coupled and even though we explain them each separately and sequentially, they usually are executed in parallel.

Writing the source code

We typically have different input work products for the different types of artifacts that have to be developed in this phase:

- For the view source code:

As discussed in 8.5, “Macro design” on page 151, GUI prototypes may have been produced to better understand the requirements and their possible solution. If so, when writing the view portion of the source code, there will be something to start with. Typically, there will be a set of static view contents (for example, HTML pages) that make up the user interface with no business logic behind it, and an interaction controller built into the HTML code that provides basic navigational functions to fulfill the tasks of a GUI prototype.

Now, this built-in interaction controller logic needs to be removed from the existing prototype code. Then the static content (HTML pages) that serves as input pages needs to be separated from the dynamic content. The dynamic content, for example, the result of a complex query, needs to be written as JSP code.

- For the controller source code:

The interaction controllers of the different use cases were modelled in the micro design phase. Therefore, there should be good documentation on these controllers in the design model tool. You can generate the initial Java source code from this model. The functionality of generated servlet classes is then programmed by using the design class model and the interaction diagrams as input work products.

- For the model source code:

The same is true for the business logic. The initial Java beans are generated. The business logic is added with the documentation of the design class model and the interaction diagrams.

- For the third-tier source code:

The design of the “glue” code for the third-tier components should produce the needed input for the coding phase. Use the appropriate process, which very much depends on the technology used, to implement the relevant back-end system source code. The process can include tasks like writing code for the back-end system, enabling access to a host database, or adding new programs to a transaction system.

Testing the source code

Each development element should be unit tested by the developer who created and therefore owns the source code. Since the MVC pattern is used as a base design architecture, each component is unit-tested on its own. Since some components depend on input from others (for example, a JSP needs the bean that it is displaying to work properly) it is very important to write test code that simulates the missing components. Writing test code during the source code development phase for the dynamic contents and the Java code helps with this dilemma in the early stages, and allows almost instant verification of the developed code. You should encourage your developers to do this along with source code development.

Creating runtime code

The developed work products differ in the way they are deployed. All view elements, like HTML, images, and JSPs are deployed as is. There is no difference between source code and runtime code, because they are interpreted or compiled at runtime. All Java source code has to be compiled into bytecode before deployment. Therefore it is important that the script and business logic developer plan for the final build of the runtime code.

For the new or changed third-tier components there may be a different kind of runtime code build process. We do not go into details of the third-tier components in this redbook but we mention it for planning purposes.

Managing the source code

When developing as a team it is always crucial to have good source code management. This issue becomes even more important when developing an e-business application with all the many different kinds of assets we described in this section.

Source code management depends very much on the tools used. The integration between the tools is important to be able to easily share the assets from one tool to another. The team support and the support of version control systems are further significant characteristics of development tools.

8.7.1.3 Develop source code: tools

The tool suite for developing e-business applications for WebSphere Application Server along with its usage and its life cycle is well documented in *Developing an e-business Application for IBM WebSphere Application Server*, SG24-5423-00. We will show an updated version of the tool usage here.

There are two starting points for the source code development phase:

- The design model, captured in Rational Rose

- The GUI prototypes, created with an HTML authoring tool, like NetObjects Fusion

The script and business logic developers generate the initial source code for the servlets, Java beans, and other Java classes using Rational Rose, and import it into VisualAge for Java. When they have coded all the logic using VisualAge for Java, they can create the class files in WebSphere Studio using a special built-in tool to interface with VisualAge for Java. From then on, the developers can use the same built-in tool to keep the Java classes in both tools synchronous.

The view developer imports the static GUI prototypes into WebSphere Studio and from then on manages the code in the workbench of WebSphere Studio. The view and script developers use the WebSphere Studio Page Designer to import code and to create new view code.

WebSphere Studio is also used to publish the developed code to the Web and application server, for testing purposes, and finally into production. The Web server may have been used to test and evaluate the GUI prototypes in the early stages of the project.

Figure 65 shows an overview of the development tools and what they are used for.

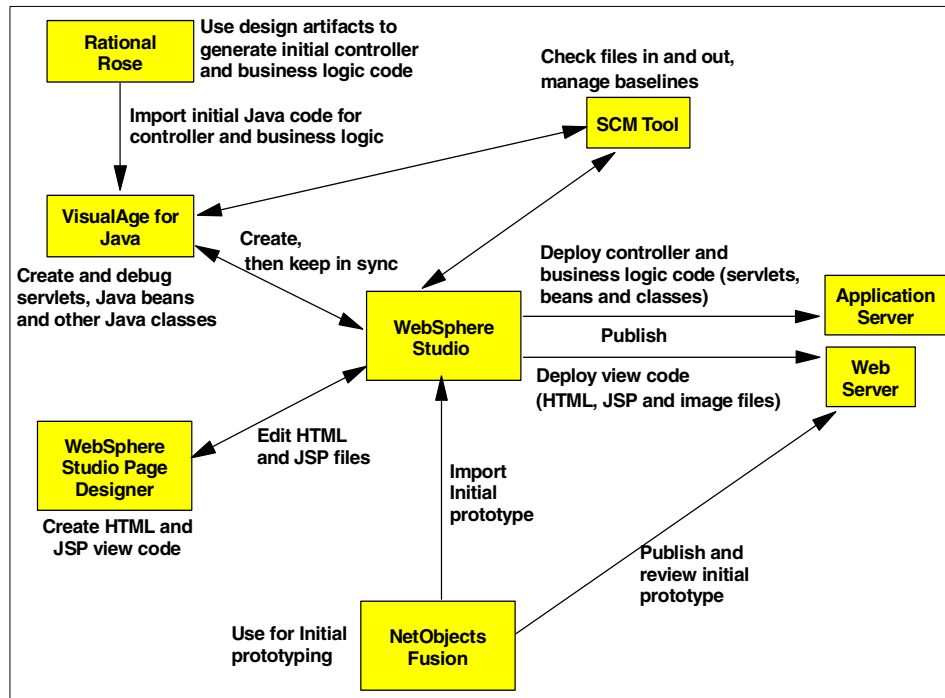


Figure 65. Tool usage in the source code build cycle

If a source code management (SCM) tool is used, the development artifacts from VisualAge for Java and WebSphere Studio have to be checked in and out using the appropriate SCM client. The SCM tool should be used to manage baselines of all source code.

Rational Rose

For the controller and business logic source code we used the code generation feature of Rational Rose. As described in 8.6.5, “Component model” on page 166, we mapped our model classes to components in the Component View of Rose. There is some initial work to be done before you can start generating the Java source code for those components:

- First you have to define your general Java settings, by selecting **Java -> Project Specification...** from the Tools menu.

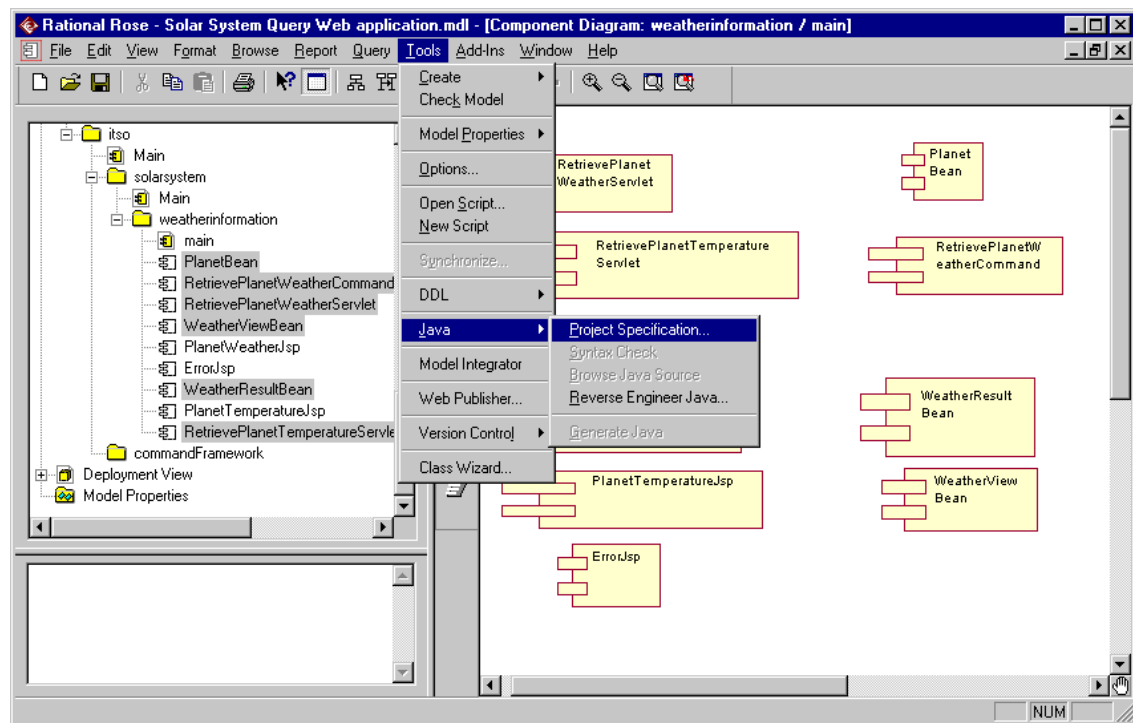


Figure 66. Rational Rose - Java code generation options

- In the Project Specification window, under the Class Path tab, add a directory, (for our example, c:\itso\codegen\solarSystem), to the Directories list. This will be the target directory for the code generation.

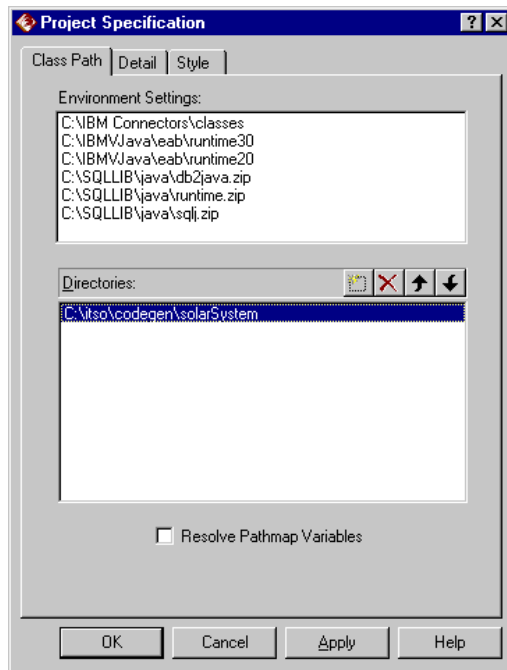


Figure 67. Rational Rose - Java Project Specification

Further general options for the code generation can be set on the other two tabs of the Project Specification window.

- Start the code generation for selected components (servlets, beans, and commands) from the Component View, as shown in Figure 68.

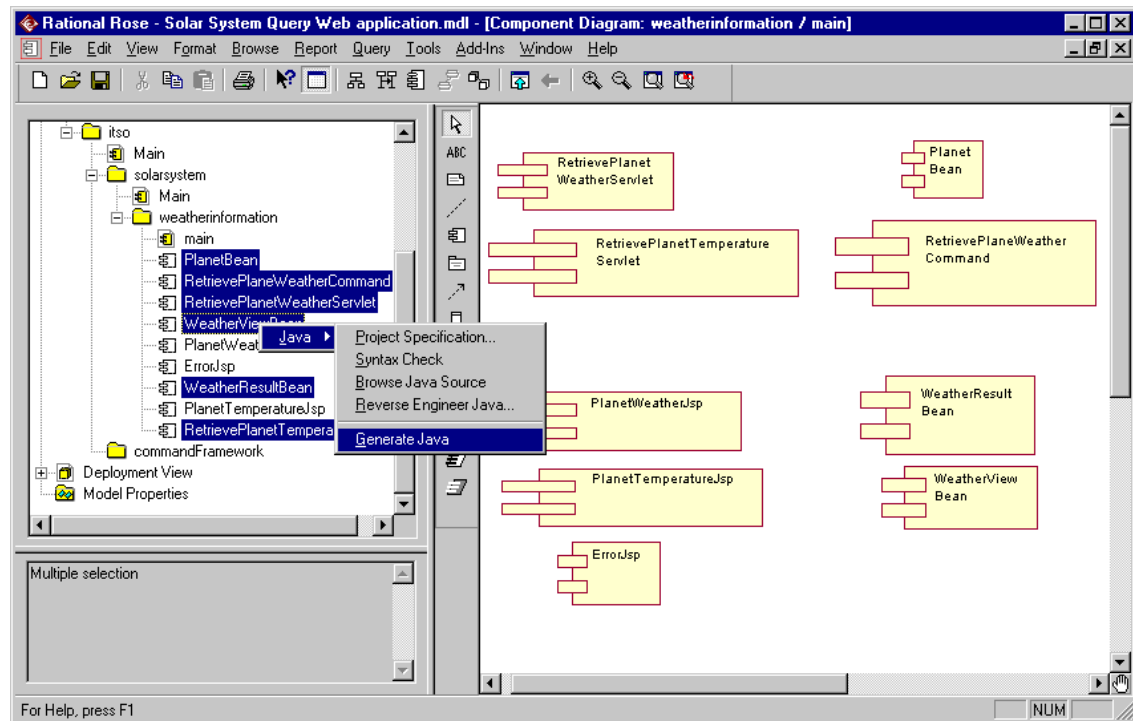


Figure 68. Rational Rose - Java code generation

- You have to map the components and/or packages to a directory, as shown in Figure 69.

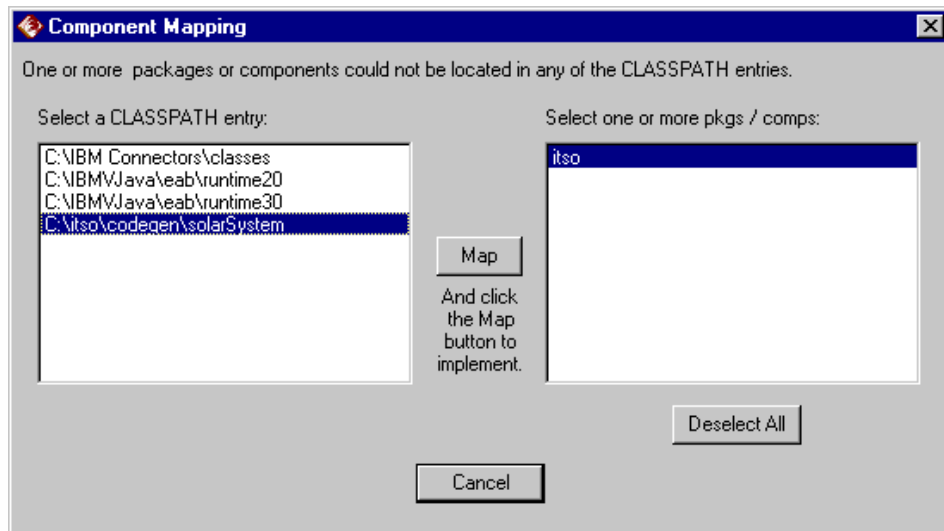


Figure 69. Rational Rose - component mapping for Java code generation

For more information on Java code generation with Rational Rose refer to *Rational Rose 2000 - Using Rose J*, which comes with the product's user documentation.

The approach we used for integrating the work products from the design phase into the build cycle is rather manual. There is a tool under development that should ease this task, called *XMI Toolkit*. It is currently available as beta Version 1.2. With this tool you can move a Rational Rose model into VisualAge for Java, modify the source in the IDE, and update the model with your changes. For more information on this tool refer to:

- <http://www.ibm.com/software/vadd>
- <http://www7.software.ibm.com/vad.nsf/data/document1924>

WebSphere Studio

WebSphere Studio is a team-enabled environment, where the other development tools are integrated. It is used to organize the source code and runtime code of the entire development project and can be used to launch a specific tool for any kind of file type. WebSphere Studio has very good publishing capabilities, making it easy to publish to different destinations, like a test environment or the production system. Furthermore, it can be used to automatically publish changed files, or selectively publish parts of the files.

Developers use the WebSphere Studio workbench to create and modify all their assets, except for the work products created for the third tier. The assets

are stored in a shared file system or a version control system (VCS) sitting underneath WebSphere Studio. Independent of which technology is being used for sharing the files, WebSphere Studio provides a check out/check in paradigm, where checked out assets are locked to avoid two users from simultaneously (unknowingly) updating the same asset.

While the shared file system is the simplest to set up, it doesn't provide the level of capabilities provided when using one of the VCSs. VCSs provide file versioning and are easier to manage with finer grained access control. WebSphere Studio supports Microsoft SourceSafe, Rational ClearCase, Intersolv PVCS, IBM VisualAge TeamConnection, and Lotus Domino. When used with Domino's Common Source Code Control Interface (SCC) support, WebSphere Studio integrates with the provided workflow templates to provide added value.

Typically assets are created and modified through the WebSphere Studio workbench, stored in the shared file system or VCS, and are available to all team members. There are two important cases where this isn't true:

- First, there is the case where you already have GUI prototypes, typically implemented in static HTML files. These files have been produced in an earlier phase of the project, using an HTML authoring tool, like NetObjects Fusion (not using the WebSphere Studio workbench).

To make them available, the view developer either places them into a folder within the shared file system, or if a VCS is being used, uses the VCS client to add the new assets to the VCS project. After the assets are placed into the shared file system or VCS project, the developer can use the workbench to do an **Insert -> File... -> Use Existing** or **Insert -> File... -> From External Source** and add the assets to the e-business application so they are available to the entire team.

- The second important case is when the script and business logic developers are taking advantage of the advanced version and component management within VisualAge for Java (with its own team support). In this case, the Java source is maintained in VisualAge for Java's source repository where it has advanced capabilities such as method versioning and incremental compiles.

These assets can also be made available in WebSphere Studio. After the developers use VisualAge for Java to change one or more Java classes, beans, or enterprise beans, they use the VCS client within VisualAge for Java to replace the updated class files within the WebSphere Studio project. If the script or business logic developer creates a new class in VisualAge for Java, the asset must be added to the VCS project and then

added to the e-business application as described above for a view developer.

With Version 3 of WebSphere Studio and VisualAge for Java there is a new built-in function that allows the developer to directly create Java assets in WebSphere Studio using the source code repository of VisualAge for Java. Such assets can then be updated in both directions using WebSphere Studio.

WebSphere Studio and VisualAge for Java Servlet and JSP Programming, SG24-5755-00, has a chapter devoted to software configuration management and how it relates to WebSphere Studio and VisualAge for Java.

WebSphere Studio has the capability to integrate any kind of development tool, so it is possible to have a special tool for each work product. Furthermore, WebSphere Studio comes with the WebSphere Page Designer and three integrated wizards:

- The JavaBean Wizard
- The Database Wizard
- The SQL Wizard

With these wizards it is possible to build very simple applications without using a Java development environment like VisualAge for Java. You start using the SQL Wizard to create SQL queries, inserts, updates, and deletes. With the Database Wizard you can create the input pages, servlets, beans, and JSP output pages for the database application. Or, you can even drag and drop the generated input/output form onto an existing page within Page Designer in cases where the content for the page has already been created.

The JavaBean Wizard is used to create Web pages based on Java beans, that for example have been provided by a script or business logic developer. This wizard eases the work of the view developer when having to integrate any normal beans and the command, navigator and access beans that are special artifacts produced with VisualAge for Java by the business logic developer.

For a detailed description of the product refer to *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00.

WebSphere Studio Page Designer

The WebSphere Page Designer is a WYSIWYG (What you see is what you get) tool for page construction. It supports static (HTML) and dynamic (JSP) contents. There are also two tools for creating images and animations included in the product.

The view developer generates the input pages with Page Designer. The WYSIWYG view of the Page Designer is used to fine tune the input pages and the generated JSP templates. The script dialog or source view is used to add any client-side JavaScript that is needed.

The script developer writes the more complex JSPs using Page Designer's source view.

VisualAge for Java

VisualAge for Java is an integrated development environment (IDE) for developing all kinds of Java code, including normal Java classes, Java beans, servlets, and enterprise beans. It is repository-based, uses incremental compiles, and has its own built-in version control management. VisualAge for Java is perfectly suited for team development, because it has a highly sophisticated management concept for code developed by a number of different users. It has a built-in WebSphere application server test environment which allows the user to test the whole Web application inside the development environment. The product comes with various Java connectors for all kinds of systems, ranging from JDBC for database access, over middleware like CORBA or MQSeries, to back-end systems like IMS and CICS.

The script developer uses VisualAge for Java to edit, test, and debug the servlets and JSPs, and may also use the Java programming environment to monitor the HTML-generated output as the JSPs execute.

The business logic developer creates and edits classes, servlets, and beans inside the IDE, using smart guides or manually. The developer uses builders such as the Enterprise Access and Data Access Builder within VisualAge for Java to specify the details of the beans generated to access the legacy applications and data (exploiting the services provided by the WebSphere Application Server Advanced Edition connectors). With VisualAge for Java the business logic developer can edit, test, and debug all classes, servlets, and beans inside the development environment.

8.7.2 Testing

An important part of the build cycle is the testing. We talked about unit tests in the source code development section since unit testing really belongs in the coding phase. The test methods described here are produced in a separate process, possibly by different team members in parallel or after the coding phase.

There are four kinds of tests in this phase that we will elaborate on:

- Integration test
- Usability test
- System test
- Stress and performance test

8.7.2.1 Testing: work products

It is important to plan for testing as early as in the macro design phase. Because an e-business application is built for a heterogeneous runtime environment, including various technologies and platforms, often with no control over the users (for example, the number of users, education level, and behavior) the testing must be planned with much caution.

We suggest that the test plans are really work products and that you should handle them as such.

Test plans

Plan for integration tests whenever a portion of the source code is ready for deployment. Do not wait until all code is written before starting integration of the different development assets. It is a good idea to start integrating small amounts of code very early instead of doing it all at the end. There is always the danger of a “big bang” where you realize that the components might not work together, but since everything is already finished the change effort is huge.

Plan to write test code to simulate missing components. Test the presentation and business logic code with simulated back-end system test code, and vice-versa.

If the developed application is for the Internet and the target audience is virtually any user, it is not easy to run a usability test where the final users can be easily involved. In intranet environments, you know the users. For Internet applications, plan for internal usability tests where you use internal staff to test the look and feel of the application, preferably people who do not know much about it.

Plan to test all the browser products you want to support. Test to see what happens if a feature your application requires is turned off, for example, cookies, JavaScript or Java.

Plan for an official or unofficial beta Web site where any, or chosen, users can test the user interface and give you their feedback. For intranet applications work together with the actual users to explore their impressions about the usability of your application. Plan to do this task as soon as you have

prototypes, even if there is no real functionality behind them. Rerun these tests after incorporating the users' feedback.

The system test needs to be performed in a runtime environment that is as close as possible to the final production environment. It is important to plan the setup of such an environment early in order to be able to start system tests as soon as the first of the developed code is ready to run. As for the integration test, plan to test as early as possible. A good indicator of when to start is when there is at least one developed component that encompasses all application elements (view, controller, model and optionally connector), so it spans over all runtime platforms. For our solar system example, this would be when we have one scenario where the user enters the name of a planet, the controller retrieves the model for that input and the page constructor builds an output page with the weather information of the chosen planet.

The challenge in planning for stress and performance testing is to build an accurate test environment. You have to plan for many hardware resources, that must be set up, installed and configured. Access by a large number of clients can be simulated, but the servers must have the real hardware configuration you plan to use. Another thing to keep in mind is that for an overall performance test of the system, the development of all the different components has to be synchronized, especially the third-tier activities, to be able to start these kinds of tests.

Plan to create test data for all the different tests.

Test drivers

The methods of running a test, including the setup, the steps, and how to perform them, has to be well documented. This ensures that you can run the same test over and over again. It is important to compare the different test results in order to tell if something has changed. The change could be due to an error that has been fixed or a new feature that has been added, but it could also indicate a new bug or some other malfunction that has not been there before.

Whether you run your test manually or you use a test tool, we recommend building test drivers that can be rerun over and over again. As stated before these test drivers have to encompass an exact description of the test system setup, the steps performed, the expected outcome, the test data and optionally the test code that simulates missing system components. For manual tests it is sufficient to have this information in a text format. Keep any scripts created for test tools.

Test data

An important work product for any test is meaningful test data. Generate test data from the given use cases, because the use case will describe the intended scenarios and data for the application. Also add unexpected data values and values that are not in the correct range. With this data you can test if the components are handling exceptions correctly.

Test results

The results from the various tests should be recorded and kept so you can use them as a comparison for later test runs in the build phase or for a later development cycle.

The test results tell you a lot about the quality of the developed code. As the project proceeds, comparing test results from different test runs in the same or different build cycles, will give you a good understanding of how the code quality has changed.

8.7.2.2 Testing: process

We have talked a lot about the test planning and when it should take place, so we won't elaborate on this issue again. The process of test planning should start early in the project, usually in the macro design when you create your first project plans. In the build cycle the plans for the various tests should be reworked to match the scope of the actual developed components for each cycle. The plan can then be fine-tuned in each test phase.

The unit test of source code has to be done in the development phase.

An integration test on a component basis can take place as soon as all necessary code elements for such a component are ready. Simulate missing components. It is often a good idea to test the code of a specific component while simulating the code of the other components involved. With this approach you can minimize the complexity of the test and focus on problems with that component without having to worry about the proper function of the other components involved. This makes it closely related to the code development. A final integration test for all components is only needed once components are interconnected. Otherwise, it is sufficient to have all related components tested together.

The system test is used to test the application as a whole. It is run in a production-like environment. You can start with system testing as soon as there are components available that have to be deployed on all the different system platforms. After all components are integration tested there should be a final system test of the whole application.

The usability test can start as soon as there are prototypes and should be rerun after the incorporation of user feedback and whenever new features are introduced to the user interface. These tests are pretty independent of the business logic development, since it is okay to have dummy code behind the GUI that does not do the real work, as long as the user gets a good impression of the look and feel of the application.

Stress and performance testing should be performed as the development progresses to eliminate any surprises before it is too late. The final testing should be done when all components of the development cycle are completed. These tests have to be done in an environment that is as similar as possible to the real production environment. This might be difficult for Internet applications, where you have a long network link from your Web application server to the client browser. This link can include many unknown factors, like the network link of your company to the outside world, the bandwidth of the network, the infrastructure of the Internet in the user's home country, the user's Internet access provider, and even the technique the user uses to access the Internet access provider. So do not spend too much time trying to figure out where to fine tune your code before you have some results of the overall performance test. These results might show that the bottlenecks are hidden in places or components that you would never have guessed. Perform stress tests to avoid pitfalls in production when the number of users increases to an amount you have never even tried to simulate. Considerations for performance are found in Chapter 5, "Performance guidelines" on page 55.

8.7.2.3 Testing: tools

Unit tests are done in the development environment if possible, for example, Java code can be tested directly in VisualAge for Java. Use test code, like special packages with test classes, to unit test Java code. This approach is very handy since you can organize the test code together with the source code, but the deployment is only done for the source code.

VisualAge for Java can also be used very effectively for integration testing because it contains an application server (the WebSphere Test Environment) where you can test the interaction controller together with the business logic and page construction (servlets with beans and JSPs). You can even have your static contents, like HTML, served by the application server built inside the development environment. As for unit testing, you can write Java code to perform integration tests.

For black-box tests used in integration and system testing, the usage of recording/playback test tools is very powerful. You perform a test scenario

using the application's user interface, and record all the steps you perform. Later you can rerun the recorded script as often as required automatically.

Usability tests are, as the name suggests, done by humans who test the functionality and the look-and-feel of the application. An error reporting and tracking tool can help support the manual testing. It should have the capability to record all relevant information about the error and allow the tester to enter this information together with the error report. The developer can then use the error report together with the exact error information stored in the tool to fix the problem.

Try all possible client technology and configurations in usability testing when developing an Internet application because you cannot control:

- What browser product and version are used
- What features are turned on or off, for example, cookies, JavaScript, Java

For stress and performance testing, tools that can simulate large numbers of clients, fast and multiple server access, and heavy network load are indispensable.

8.8 Deployment

Depending on the size of the developed components in the build cycle, the development team might decide to build it in several iterations. Even though each build cycle produces executable code, only the final result is usually deployed. The whole project is also run iteratively. It is up to the development team to decide which release, built in a development cycle, is going to be deployed. There might be alpha or beta releases that are deployed only to a certain number of test users.

Deployment: Work Products

A deployment plan has to be created that encompasses, not only when and how to install and set up the newly developed application, but it must include all hardware and prerequisite software requirements.

You also have to plan for system management, taking into consideration what has to be managed and how, how to establish the required security, and what has to be done for availability and recovery, before you can deploy the application into a production environment. More information on systems management can be found in Chapter 9, "System management products and guidelines" on page 197.

Deployment: Process

When preparing for deployment of a Web application, you have to plan and execute the hardware setup. In e-business applications with one of the architectures described in this book, this means server and network hardware, database and Web application server machines, network routers and firewall machines. For an intranet application this might also include client hardware, for example, thin network computers. The software for all this hardware has to be installed and configured, including any databases used, Web and application servers, firewall and security software. If the system uses application topology 2, it needs access to other production systems. In some cases this means adding or changing components of a running system. This task must be addressed in the deployment plan.

You also have to plan how to hand over the operations of the production system to the staff who will be responsible for it.

Deployment: Tools

In WebSphere Application Server, the Web content (HTML, JSPs, GIFs and other Web assets) must be published to one directory or set of directories while servlets, beans and EJBs are published to a different set of directories. The structure for publishing the Web content might be dictated by local site rules. For example, there may be rules that say all content must be published to a single directory, all content for each subsite must be published to a single directory, or that content must be published to directories by type (HTML, images, etc.). This structure is not always the best way to organize the content for authoring (inside WebSphere Studio, for example). If the Web server is separated from the application server, static and dynamic contents are published to different servers. Finally, for scalability, it is often desirable to publish some static content, such as images, to a separate server from the HTML and logic.

There are many options in setting up and configuring the Web application server, as explained in Chapter 3, “Choosing the runtime topology” on page 19. Deploying the dynamic contents to the WebSphere Application Server and configuring the deployed contents is a task that has to be planned thoroughly and executed very carefully.

The WebSphere Studio workbench provides the ability for you to define an arbitrary directory structure to organize your e-business application’s assets, view the relationships between the assets, and define how the assets will be published. Each asset is assigned a publishing target by default, mirroring the author time directory structure, but you have the ability to override this behavior and publish individual assets or entire folders to different directories and even split the content between multiple servers. The workbench adjusts

the links to ensure that no matter how you publish your site, the links don't break.

After the deployment of the files you have to configure your Web application to run under the IBM HTTP Server and the WebSphere Application Server. Use the IBM HTTP Server Configure Server service and the WebSphere administrator's console for the configuration.

8.9 Where to find more information

IBM Publications:

- *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00
- *Developing an e-business Application for IBM WebSphere Application Server*, SG24-5423-00
- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265-00.
- *Connecting e-business to the Enterprise by Example*, SG24-5514-00

Other publications:

- Booch, Grady. 1994. *Object-Oriented Analysis and Design with Applications* (Addison-Wesley Object Technology Series), Reading, MA; Addison-Wesley Publishing Company; ISBN 0805353402
- Jacobson, Ivar. 1992. *Object-Oriented Software Engineering; A Use Case Driven Approach*, Reading, MA; Addison-Wesley Publishing Company; ISBN 0201544350
- Rumbaugh, James et al. 1991. *Object-Oriented Modeling and Design*, Englewood Cliffs, NJ; Prentice Hall; ISBN 0136298419
- Fowler, Martin, Kendall Scott (Contributor) and Ivar Jacobson. 1997. *Uml Distilled; Applying the Standard Object Modeling Language*. Reading, MA; Addison-Wesley Publishing Company; ISBN 0-201-32563-2
- John Barry, Tom Bridge, Paul Fertig, Tom Guinane, Geoff Hambrick, Daniel Hu, Tom Kristek, Dave Livesey, Guillermo Lois, Mike Page, Branko Petch, Frank Seliger, Thomas Wappler, Brian Watt, Martin West, Goerge Yuan: *Developing Object-oriented Software - An Experienced-Based Approach*, Prentice Hall, 1997, ISBN 0137372485

Chapter 9. System management products and guidelines

Previous chapters described how you could begin designing an e-business application by selecting the appropriate application and runtime topology, and then discussed the application design and development issues typically faced in e-business implementation.

Once the application development is completed and your users have successfully tested and accepted the system, it is time to start implementation of the systems management phase. In this phase, you will have to deploy and manage the application in a production environment, which will require early planning. Specifically, we categorize the set of post-implementation activities that have to be performed on a routine basis under system management.

What exactly is involved in system management? It typically involves:

- Application management
- Performance monitoring
- Availability management
- Security management
- Disaster recovery
- Operating system and network administration
- Asset management
- Software distribution
- Problem reporting
- Change management

Looking at this list of activities, system management is certainly not trivial. In fact, each of these activities requires highly specific skills and professional experience to perform competently. Besides the skills factor, you will also have to decide on a set of tools to manage the system management activities. What tools should you select so that they can work in an integrated and coherent manner?

Beyond the technical challenge that system management poses, there is also the added pressure from management. In many situations, you will be bounded by service level agreements (SLAs) to your users. Such agreements typically cover system availability hours, system utilization and problem resolution response time. These measurements will be collected, tabulated and reviewed on a regular basis by management, to ensure accountability and a well maintained system. Thus, you will also require reporting tools to facilitate the SLA review.

With all this management focus and these targets to meet, it makes systems management more daunting and challenging. Since it is important, it is our recommendation that you start planning early. Incorporate system management requirements in the early phases of your design since what you design will affect how you eventually manage it. Conversely, what tools are available to manage your system also affects your application design.

To explain each system management activity in detail is beyond the focus of this redbook. What we will focus on are the key system management activities related to examples in the user-to-business topologies using WebSphere Application Server Advanced Edition V3.0.2. These include:

- **Managing your WebSphere application**, which describes how day-to-day administration can be performed in WebSphere and also the tools available for generating reports.
- **WebSphere end-to-end security**, which describes security issues from the physical, systems, network and application perspectives.
- **Backup and recovery**, which presents ideas on disaster recovery using Tivoli Storage Manager.

The following table maps the set of activities we will describe against the list of system management activities.

Table 3. System Management Activities described

System management activity	Described in:
Application Management	9.1, "Managing your WebSphere application" on page 199
Performance Management	
Availability Management	
Security Management	9.2, "User-to-business WebSphere end-to-end security" on page 208
Disaster Recovery	9.3, "Backup and recovery of your systems" on page 220

9.1 Managing your WebSphere application

For this discussion we will look at the runtime topology shown in Figure 70. In this example, the network is divided into three segments, an internal (or secure) network, a DMZ, and the external network. All external user access will have to flow through the protocol firewall to the Web server in the DMZ. Depending on the type of user request, the Web server redirector may forward the request through the domain firewall to the application server in the internal network. The application server will process the request and send any response back to the external customer through the two firewalls.

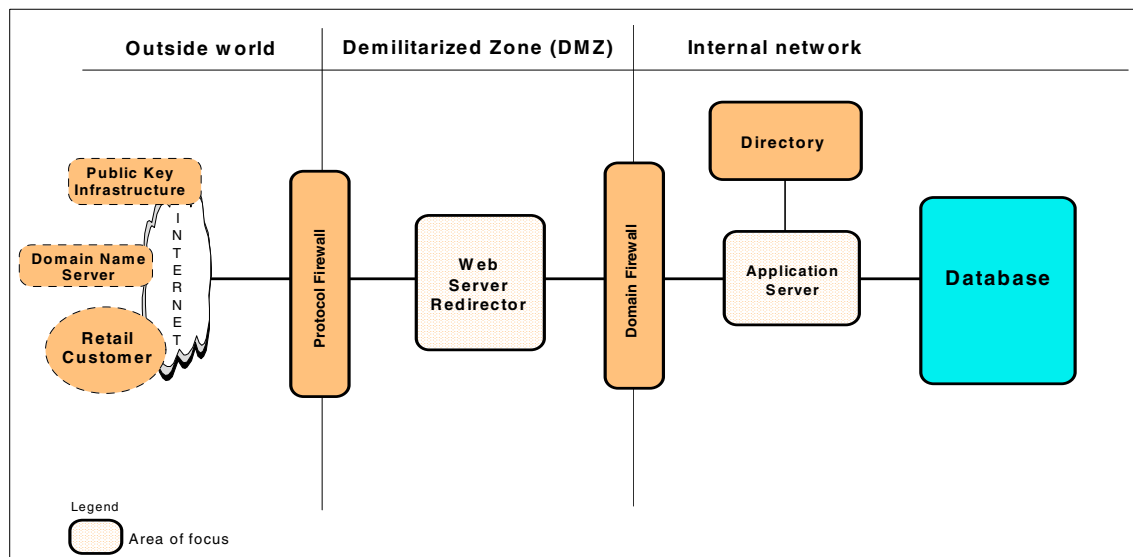


Figure 70. Managing WebSphere resources

A WebSphere application is a combination of HTML pages, JSPs, servlets and EJB resources. These resources will be deployed and managed under the WebSphere Application Server environment. In our scenario shown above, the Web server redirector will host static HTML pages while the application server will host JSPs, servlets and EJBs.

In the rest of this discussion, we will refer to the WebSphere specific terms described in the following table.

Table 4. Description of WebSphere terms

WebSphere Terms	Description
Web resources	Refers to servlets, JSPs and HTML pages.

WebSphere Terms	Description
Web application	Application consisting of Web resources.
Enterprise application	Application consisting of Web applications and EJBs.
Application server	A JVM runtime service that handles user requests to enterprise and Web applications.

9.1.1 WebSphere resource management

WebSphere resources use core underlying services like the servlet engines, EJB engines, security application and cluster models residing in the application server. These services manage the Web resources and applications that are hosted by the Web server and the application server. Depending on application requirements, you may be managing a single stand-alone application server or multiple application servers which support failover capabilities.

The WebSphere administrative console is used to manage, deploy and configure the WebSphere resources. This console manages each application server through the administrative server running in each WebSphere Application Server.

Figure 71 shows a logical view of the WebSphere administrative components. It consists of an administrative console, the WebSphere Application Server components (administrative server, application server) and the WebSphere repository.

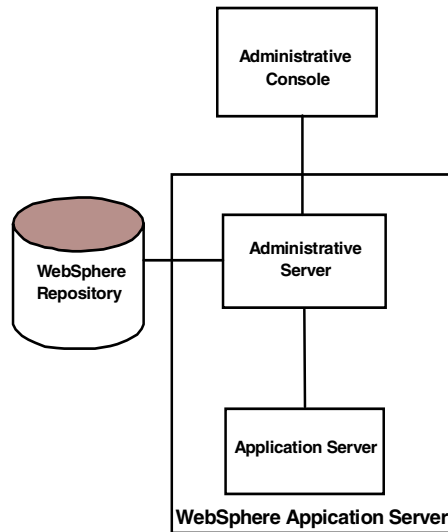


Figure 71. Managing stand-alone WebSphere Application Server

Note: The administrative server is typically running in each WebSphere Application Server. The only exceptions would be nodes running as “thin” servlet redirectors or under administration agent mode. The thin servlet redirector is discussed in 15.1, “Creating a standalone redirector” on page 297.

9.1.1.1 WebSphere administrative domain

The management challenge grows when you have more than one WebSphere Application Server. Can you still effectively manage these distributed servers from a central location? Fortunately, you will still be able to do so from one WebSphere administrative console. This is possible through a configuration known as the WebSphere Administrative Domain. When you have more than one related WebSphere Application Server, they can be configured to be part of the same WebSphere domain.

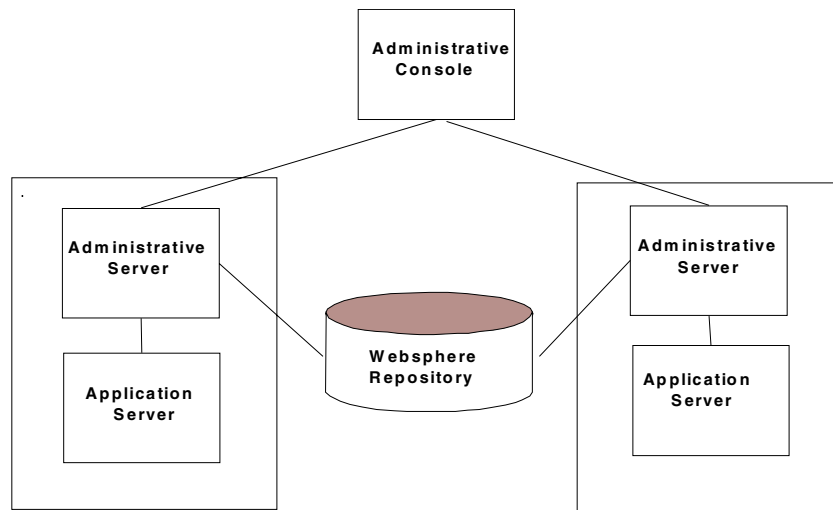


Figure 72. Managing multiple WebSphere Application Servers

The characteristic of this domain is the sharing of an administrative repository by all WebSphere Application Servers. The repository is a DB2 UDB database and it contains the following types of information:

- Static information including configuration information, node attributes, JNDI names, and model clone configuration
- Dynamic information including the state of resources while the system is running

This repository is located on the first WebSphere Application Server during initial setup. For subsequent WebSphere Application Server installations, you will have to create a remote database connection to the repository of the first WebSphere Application Server.

9.1.2 Using the WebSphere administrative console

With an understanding of the various components in WebSphere administration, we will describe the functions that the administrative console provides. These functions are categorized under three tabs in the application:

- Tasks
- Types
- Topology

9.1.2.1 Tasks



Figure 73. Functions under the Tasks tab

There are three main functions you can perform under the Tasks tab.

1. **Configuration tasks** - This is where you will do your initial configuration of new application servers. This will include configuring servlet engines, virtual host definitions, enterprise and Web applications. For a detailed step-by-step description of the configuration, please refer to the online administrative console tutorial located at <http://www.ibm.com/software/webservers/appserv/doc/v30/ae/tutorial/guitut.htm>
2. **Performance tasks** - You will utilize the WebSphere Resource Analyzer to measure performance statistics of your application server. It can be configured to provide a runtime performance view of a variety of resources, including servlets, enterprise beans, sessions, database pools, JVM memory and thread pools. When collected over a period of time, these statistics can help you create a baseline for the performance of your application. The data is also useful in management SLA reviews.
3. **Security tasks** - The last task you can perform is configuring security for an enterprise application. This includes selecting an authentication mechanism and the type of user registry. The components of application security are discussed in 9.2.4, “Web application security” on page 211.

9.1.2.2 Types

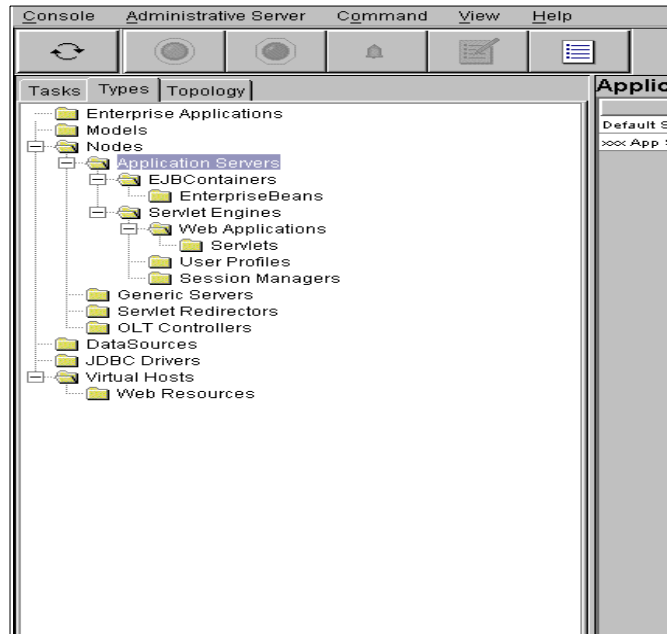


Figure 74. Functions under the Types tab

Under the Types tab you can see the resources that can be configured in the WebSphere Application Server environment. In addition, it also lists all existing configured resources.

The Types view also allows you to see the relationships among resources in the administrative domain. The resources are displayed according to their hierarchical, parent-child relationships. One useful thing about this hierarchy is that it shows you the order in which you must configure resource instances.

The Types tab also allows you to set default properties for each resource type. This is like creating a template with default values. When these resources are instantiated, the newly created resources will inherit the default values. This can greatly simplify the configuration process.

9.1.2.3 Topology

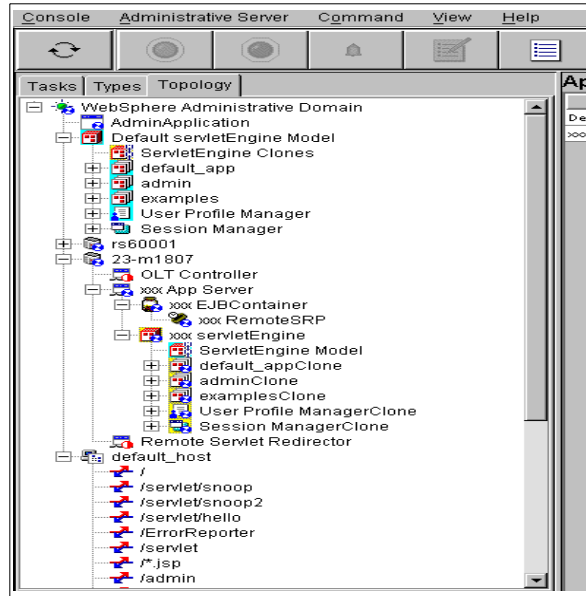


Figure 75. Functions under the Topology tab

Under the Topology tab, you are provided with a physical view of your WebSphere Application Server deployment. For each application server, it will list the resources associated with it.

The functions provided here are where most of the day-to-day administrative operations take place. For example, you can manage at the administrative domain level by stopping or starting different application servers within your domain. You could also manage at the application server level, by monitoring the status of your servlet and EJB engines.

9.1.3 WebSphere Site Analyzer

Beyond managing the WebSphere resources that keep your application up and running, you will also have to manage another important facet of the system. This relates to the Web content that you put up on your site. There are two areas of focus here.

First, remember that the first impression customers have of your company will be your Web site and you certainly want to leave them with a positive impression. Thus, you will need to ensure that they do not encounter errors or unnecessary delays when navigating your Web site.

Second, your Web site becomes a new communication channel which will allow you to deliver messages to your customers. As more and more customers visit your Web site, how do you know whether these messages are received and perceived effectively?

To help you answer the above, WebSphere provides an integrated tool called the WebSphere Site Analyzer. This tool allows you to analyze your Web site in two areas, content analysis and usage analysis. These analyses can be generated and presented in the form of charts for easy understanding.

The truth is that there is a wealth of information hidden in each of the “hits” and visits to your Web site and this information could:

- Help you understand your customers better by unlocking the secrets that are in the trails they have left behind.
- Assist you in enhancing your Web site, so that it will help you retain and attract new customers in your target market.
- Understand the peak periods of customer access, to help you justify system upgrades or plan for system management activities.

9.1.3.1 Create a WebSphere Site Analyzer project

To utilize WebSphere Site Analyzer to explore the site contents, you first create a Site Analyzer project. This project will help you organize related tasks into a single folder. The tasks include:

Content analysis

Content analysis provides structural information about your Web site by crawling through your site. This structural information includes:

- Broken links
- Unavailable resources
- Inactive files
- Duplicate pages

The crawler collects data and places it in the Site Analyzer database. This is useful in ensuring structural and policy conformance of your Web site and hopefully reduces the possibility of visitors encountering errors.

Usage analysis

Usage analysis provides detailed visitor access information about your Web site. The information is gathered from the logs and helps answer the following questions:

- Who is using your site?

- Where are they entering your Web site from?
- What are the top referring sites?
- What are the commonly used browsers?
- What are the commonly used platforms?
- What is the typical customer behavior and usage patterns on your Web site?
- What individual pages are accessed including time spent?

9.1.3.2 Create an analysis report

After you have created a content or usage analysis project, you will proceed to perform an analysis. This is done by selecting and creating elements to be part of the report. The analysis report can be generated either in HTML or XML format for viewing.

9.1.3.3 Guidelines for using WebSphere Site Analyzer

To optimize this tool, we recommend the following:

1. Each time you have placed new materials on the Web site, perform a content analysis. This may reveal broken links that you may have inadvertently introduced.
2. Routinely run the content analysis to check for structural correctness. Sometimes, certain Web resources become unavailable and you may not be aware of this.
3. Generate a usage analysis report at routine intervals as part of the management review of service level agreements. The Web access information in this report will help management gauge overall access patterns to your Web site.

9.2 User-to-business WebSphere end-to-end security

Once your application is running in production mode, you can expect a lot of user access to the system. If we lived in a perfect world where all users were law abiding, then we would not have to worry about security. Unfortunately, this is not the case and your system will constantly face security threats, both external and internal.

The WebSphere application you have developed is only one component of the total system configuration. The golden rule is that the security strength of your system is only as strong as its weakest link. Thus, it is necessary to ensure that the other components in the system are configured securely.

With this in mind, end-to-end security will consist of physical, operating system, network and application security.

Table 5. Components of end-to-end security

Security Type	Description
Physical	Control access to the hardware equipment hosting your application.
OS	Security at the operating system level.
Network	Secure connectivity flow between external, DMZ and internal networks.
Application	Configure WebSphere security.

9.2.1 Physical systems security

Physical systems security is the foundation of the end-to-end security building blocks. Access to the hardware has to be controlled and monitored proactively. Anyone gaining unauthorized physical access to your servers could halt your server, steal valuable information from your storage, plant viruses, install harmful software, etc. All of these activities are disruptive to your operations and will cause damage to your system.

If the hardware is not secured properly it will void the other security measures you take.

9.2.2 Operating systems security

After securing your physical systems, you will have to work on securing the operating system. As the OS grows richer in function and features, new bugs are discovered or are waiting to be exploited. An example would be a bug that allows a user with non-privileged access to perform privileged operations.

At the operating system level, we recommend the following practices for administering your system:

1. Keep yourself updated on new security glitches.

Unfortunately, there are public Web sites that provide detailed information about newly discovered glitches. Fortunately, there is also a wealth of public information available which provides temporary or permanent remedies for these glitches. As an administrator, you need to be warned quickly about these loopholes and to take immediate actions to rectify the problem.

As a service to their customers, most OS vendors provide updated information related to security hacks. A good source of information is the Web site by CERT (<http://www.cert.org>). You can subscribe to their mailing lists to receive regular updates and news flashes by e-mail.

2. Access privilege account needs.

Your OS security policy will have to consider who has access to the privilege accounts. You will have to determine the roles and level of responsibility each person has. For example, you may want to separate the role of an OS administrator from the application administrator.

3. Enforce good password policies.

Many security hacks are the result of simple passwords. You will have to enforce good password policies and practices for all accounts in your system, whether they are privileged or non-privileged. Besides having this policy, educate your users on their role in the overall system security.

4. Enable logging and auditing.

Remember to turn on OS system logging and auditing. In the event of a system break-in, hopefully you will have some trails to start off your investigation.

9.2.3 Network security

Once you have the physical hardware and the operating system secured, you need to turn your attention to security between interconnected systems.

Network security is the act of protecting resources residing in your internal network and DMZ from the external network. You want to restrict and prevent unauthorized user access to your internal systems. At the same time, you do not want to make it difficult for legitimate users to access your systems.

The key technologies available to achieve this network protection include firewalls, intrusion detection monitors and anti-virus detection. The SecureWay suite of products provide you with a comprehensive security solution that will meet most requirements. Information about SecureWay products can be found at <http://www.ibm.com/software/secureway/>.

9.2.3.1 WebSphere in a firewall environment

How do you restrict and control network access? You could use a firewall in between two networks. When properly configured, the firewall acts as a choke point and will force all traffic to and from the Internet to flow through it. By doing so, it can then scan the traffic and determine whether to allow or disallow the packets based on a set of rules.

When designing your firewalls take the following into consideration:

1. Make sure that there is no direct communication channel between the applications on the intranet and the external Internet. In our example in Figure 70 on page 199, all external user requests and application responses flow through the Web server residing in the DMZ. If necessary, the Web server will forward the request to the server in the internal network.
2. KISS: Keep it short and simple. Reuse pre-configured rules that exist. Define new rules if necessary. Always remember to include a rule that excludes everything else.
3. You should not allow information pertaining to the internal network to reach the Internet. For example, you would not want the IP addresses of your internal systems to be made available to external users. Hiding this information will reduce the risks of external security hacks.
4. At a minimum, you will need a firewall between the external network and your DMZ, and a firewall between the DMZ and the intranet. Introducing a DMZ configuration creates an additional security barrier which a network infiltrator would have to overcome.

Note: When you implement this DMZ configuration, it is possible to implement all three firewalls in the same physical machine with three network adapter cards.

In our example, the WebSphere Application Server is separated from the Web server. In this case, you need a servlet redirector to facilitate communication between the Web server and the WAS. This servlet redirector communicates with the WebSphere Application Server via IIOP over RMI. Additional ports in the firewall have to be opened for proper communication to take place with

the WebSphere bootstrap, the WebSphere Location Service Daemon (LSD), system management, application server and redirector components. In our scenario, we have configured the following firewall ports. For a more detailed description see Chapter 16, “Setting up firewalls” on page 305.

Table 6. Firewall ports usage in a servlet redirector configuration

Firewall Service	Port	How to configure
Bootstrap	900	Default fixed value.
LSD	9000	Default value. To change, edit the port parameter of the LSD command.
EJS	12101	No default value. Edit admin.config file in WebSphere Application Server.
Application Server	12201	No default value. Use the administrative console to edit the respective application server configuration.
Redirector	12301	No default value. Edit the redirector batch file on the redirector machine.

9.2.3.2 Internet and intranet security considerations

The intranet environment may be a LAN-based departmental network or could even span geographic regions via a WAN-based Virtual Private Network (VPN). With this distinction in mind, we can see that the WAN-based intranet environment has similar characteristics to the Internet based systems. The physical network used in the VPN network is typically outside your management control. Thus, you should continue to focus on network security.

For a LAN-based intranet that is segmented along departmental domains, you could still implement a firewall between the various departments. A good example would be to separate the production network environment from the development network environment.

9.2.4 Web application security

The user requests will flow through your firewall to your application. The final security checkpoint would be application security which will decide who can invoke specific application function.

WebSphere provides an integrated security model to configure security on your Web resources. This can be centrally configured through the WebSphere administrative console.

Let us begin by describing the various WebSphere security components listed in the following table.

Table 7. WebSphere security components

Security Component	Location
Security plug-in	Supported Web server
Security collaborator	WebSphere Application Server
Security server	Security application

9.2.4.1 Security plug-in

When Web clients try to access Web resources deployed by the WebSphere Application Server, the security plug-in component will be invoked. The security plug-in will then send the client requests to the security collaborator for security decisions. This security plug-in is attached to the Web server during software installation.

The following table lists the respective Web servers that WebSphere Application Server currently supports.

Table 8. Supported WebSphere Web servers

	AIX	Solaris	NT
Apache Server V1.3.6	x	x	x
Netscape Enterprise Server V3.51 and V3.61	x	x	x
Microsoft Information Server V4.0			x
Lotus Domino Application Server R5	x	x	x
Domino Go Webserver R4.6.2.5 and R4.6.2.6	x	x	x
IBM HTTP Server V1.3.6	x	x	x

9.2.4.2 Security collaborator

The security collaborator is attached to the WebSphere Application Server. It makes security decisions on remote method calls on servlets or EJBs by performing

- Authorization checks
- Pre and post security trace logging
- Delegation policy enforcement

9.2.4.3 Security application

The security application resides in every WebSphere administrative server. It provides the means to configure security policies for both Web resources and EJBs.

In a particular WebSphere Domain, there is a shared repository (db2) which contains all the security configuration and policy information. Any WebSphere Application Server in the same WebSphere domain can access this shared repository.

9.2.4.4 Security server

The security server is found in the security application. Both the security plug-in and security collaborator will call the security server for authentication/authorization services. It provides:

- Centralized control over security policies (permissions, delegation)
- Central security services (authentication, authorization)

The security server is a trusted third party for security policy and control. Web servers and WebSphere Application Servers call on the security server to provide authentication, authorization and delegation services.

9.2.5 WebSphere Application Server security model and policy

We have described the key infrastructure components in WebSphere security. Next, we would like to describe the WebSphere security model and policy. Understanding this allows you to make informed decisions in configuring and managing WebSphere security. For example, you can decide which of the authentication methods are supported given a particular user registry.

Specifically, we will describe the authentication, authorization and delegation models and policies within WebSphere.

9.2.5.1 Authentication

Authentication is the process of proving who the user says he really is. In WebSphere, authentication between a user and the WAS can be specified in terms of:

- **User registry** - This is where the user and group information will be stored. In our scenario, we used IBM LDAP as the user registry. See Chapter 13, “Step 3: Securing the PDK application” on page 275 for details on configuring WebSphere using the LDAP user registry. Note that each WebSphere administrative domain can have only one user registry.

- **Authentication mechanism** - After the user has provided the required data, the authentication mechanism will validate it against an associated user registry. Two types of authentication mechanism are supported:
 - a. Lightweight Third Party Authentication (LTPA)
 - b. Native operating system
- **Challenge Mechanisms** - The challenge mechanism specifies how a server will challenge and retrieve authentication data from the user. It can be of the form:
 - a. **None** - Security runtime does not challenge user for authentication data.
 - b. **Basic** - A user is challenged for ID and password.
 - c. **Certificate** - Mutual authentication over SSL.
 - d. **Custom** - The ability to specify a custom HTML page to retrieve a user's ID and password.

Note: The components of the authentication are dependent on one another. For example, the authentication mechanism is defined based on the user registry and this choice drives the challenge mechanism. This is illustrated in the following diagram.

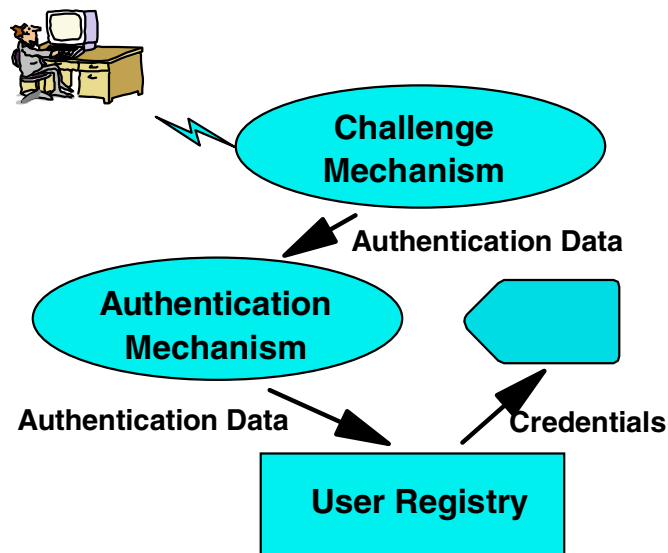


Figure 76. Relationship between authentication components

In the following table, we illustrate the relationship between the user registry and the authentication mechanism. If user ID and password are supplied, then authentication is delegated to a user registry. If digital certificates are used, then the certificate credentials are mapped to an associated user registry entry.

Table 9. Mapping between authentication mechanism and user registry

	UNIX	Windows NT	LDAP
Native OS (user ID, password)	The supplied password is encrypted using the OS's crypt facility. This is then compared against the system's password repository.	Authentication is delegated to the NT Security Access Manager via systems call.	N/A
LTPA (user ID, password)	N/A	N/A	An LDAP bind is performed using the DN (Distinguished Name) and the password.
LTPA (digital certificate)	N/A	N/A	Based on the trust in the Web server, certificates are validated through successful establishment of a mutual SSL connection. A credential mapping is then performed based on the information contained in the certificate.

9.2.5.2 Authorization

The authorization model in WebSphere 3.0 is based on the classic capability model. In this model, the permissions required to perform a particular operation are associated with a principal. This is different from the security model in WebSphere V2, which is based on the Access Control Model (ACL).

To see the difference, let's take an authorization policy that specifies a protection matrix as depicted in Table 10.

Table 10. Mapping between principal and resources

User	/sectionaForm.jsp	/topologyOne/histData	/topologyTwo/histData
Alice	HTTP_GET		
Bob	HTTP_GET HTTP_PUT	HTTP_GET	HTTP_GET
Charlie		HTTP_GET	

The ACL model is a column-based view of the protection matrix. It specifies that the /sectionaForm.jsp resource can be accessed by Alice and Bob. On the other hand, a capability model is a row-based view of the matrix. The capability model specifies that user Alice has permission to perform an HTTP_GET on the /sectionaForm.jsp resource.

When a principal requests to perform an operation on a resource, the security runtime considers a set of permissions to be the "required" permissions for performing the operation on the resources. If the requesting principal has been granted at least one of the required permissions, then the security subsystem authorizes the request to be processed.

Take for example the following configuration:

Table 11. Mapping between enterprise application and resource

Enterprise Application	Resource
TopologyOneWebApp	/sectionaForm.jsp
	/topologyOne/histData
TopologyTwoWebApp	/topologyTwo/histData

Table 12. Mapping between method group and resource

Method Group	Resource
ReadMethod	HTTP_GET
WriteMethod	HTTP_PUT

The required permissions that will be stored in WebSphere would be:

Table 13. How WebSphere permissions are stored

Principal	Set of Permissions
Alice	{ (TopologyOneWebApp, ReadMethods) }
Bob	{ (TopologyOneWebApp, ReadMethods), (TopologyOneWebApp, WriteMethods), (TopologyTwoWebApp, ReadMethods) }
Charlie	{ (TopologyOneWebApp, ReadMethods)}

9.2.5.3 Delegation

Delegation is the process of forwarding a principal's credentials along with the cascaded downstream requests that occur within the context of work that the principal originated or is having performed on its behalf.

When a client uses an intermediary to invoke a method on a target resource, the intermediary invokes the method assuming a certain identity. The identity could be:

- **Client** - identity of client requesting the method invocation
- **System** - identity of server hosting the intermediary resource which will eventually invoke the method
- **Specified** - a different identity

Delegation policy is specified in terms of the RunAs policy. The RunAs policy consists of two parts:

- RunAsMode
- RunAsIdentity

The **RunAsMode** specifies whether a method should execute with the identity of

- The principal of the caller (client identity)
- The principal of the system (system identity)
- A specified principal (specified identity)

If the RunAsMode is set to specified identity, then the **RunAsIdentity** specifies the principal identity used.

Let us take an example of how the various WebSphere components work together to provide security.

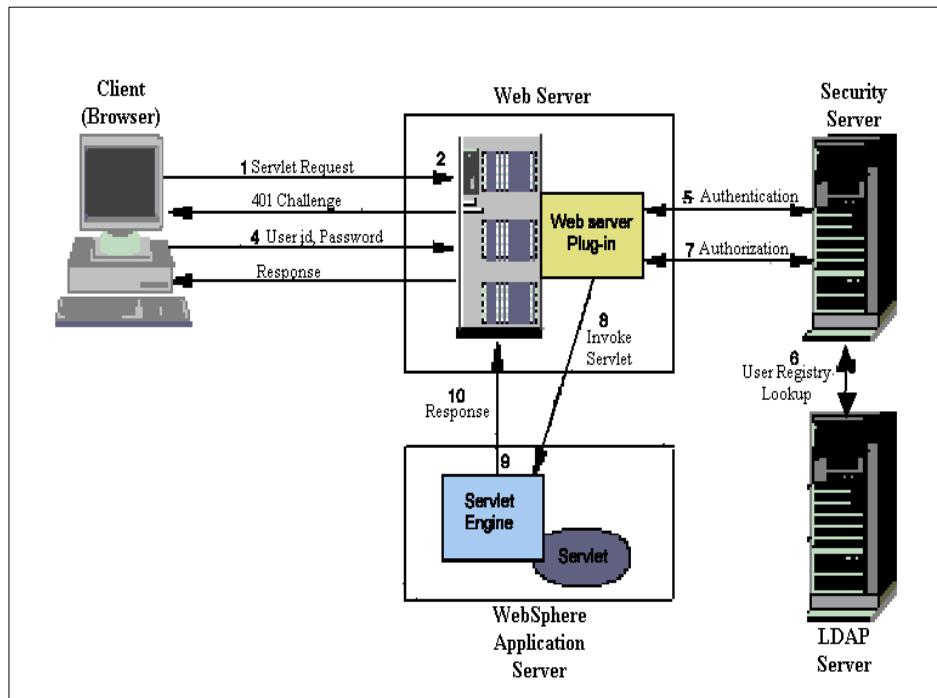


Figure 77. Security interaction flow

1. A user, Alice, requests /webapp/topologyone/histData through a Web browser.
2. The Web server will check if this resource is protected.
3. The Web server issues a 401 challenge to Alice asking her to prove who she is.
4. Alice responds with her user ID and password (authentication mechanism).
5. The Web server plug-in performs authentication by delegating the task to the security server using the given ID and password.
6. Alice's ID and password are matched against the entries in the LDAP server.

7. Once Alice has been authenticated successfully, the Web server plug-in consults the security application to determine whether the user has the required permission to access the requested resource.
8. Upon successful authorization, the Web server plug-in creates a security context with the user's credential information and passes the request to the servlet engine to service the request.
9. Before invoking a method on the servlet, the security collaborator performs a check by extracting the user's credential information from the security context and verifies that the user has the authority to invoke the method on the servlet.
10. Finally, the method is executed and the results are sent back to Alice's Web browser.

9.2.6 HTTP Single Sign-On (SSO)

When you enable HTTP single sign-on, user authentication credentials are preserved across multiple applications in the same domain, for example:

- Cooperating but disparate Web servers
- Cooperating applications like the IBM OnDemand Server and Windows NT Suites.

With SSO, your application will then avoid repeated requests of user security credentials. However, if your application wants to use the SSO feature, then it must use an LTPA registry, (LDAP for example).

9.2.7 WebSphere V3 security differences with V2

There are some security differences and improvements in WebSphere V3 over WebSphere V2. As a quick reference, Table 14 shows a summary of the differences.

Table 14. Comparison between security in V3 and V2

WebSphere V3	WebSphere V2
Users and groups must exist in either a directory server or OS user registry.	Users and groups can be created in WAS, independent of directory server or OS user registry.
Security is applied at the application level.	Individual resources are secured. No protection at application level.
EJB methods are protected.	Only servlets and Web resources are protected.

WebSphere V3	WebSphere V2
Authorization policies are comprised of application level security and method groups.	Authorization policies are comprised of realms and ACLs.
Single realm concept.	Multiple realm concept.
Centralized security services for multiple application servers.	Security services for each application server runtime.
Wider portfolio of security policies like SSO, delegation, LTPA and digital certificates.	Basic security policy and services are provided.

9.3 Backup and recovery of your systems

Backup may seem a mundane and repetitive task you perform routinely, but it is absolutely necessary. Its importance to you is never emphasized enough and typically you will only realize it during a disaster. Imagine losing valuable transactional data due to a hard disk failure and you do not have a backup.

We suggest you consider the following factors when considering a backup solution:

1. Data to backup

You should consider the solution's support for the various data you need to backup. The data includes operating systems, application data, transaction logs and configuration files.

2. Available backup window time

In most situations, there is a limited window of time to complete the backup. Thus, you will have to consider the performance of the backup solution. Consider the performance of the solution as a whole, not the individual pieces.

3. Required system recovery time

Not only should the backup be fast, but the recovery process should be equally fast. Consider how the backup solution is able to provide fast recovery.

4. Support for enterprise backup

The solution should be scalable to perform backup of new systems that you may install, as a result of growth and upgrades. You may also want to use the backup solution for other existing applications.

5. Integration with system management tools

It is very useful if the backup solution can be integrated with existing system management tools and thus provides a central administration capability.

6. Support for emerging technology

The software should be able to support emerging storage area network (SAN) based storage solutions. As your informational needs grow, these storage solutions will provide large capacity and high performance data access.

9.3.1 Using Tivoli Storage Manager

Based on the above factors in selecting a comprehensive backup solution, we recommend the IBM Tivoli Storage Manager (TSM). It is an integrated storage management solution that will meet the needs of any company, from small Internet startups to large enterprises. TSM provides the following features:

- Full support of client platforms
- High performance backup and recovery process
- Wide support for IBM and non-IBM tape/optical technology
- Scalable solution to meet growing storage demands
- Support for SAN-based solution
- Integrated with the Tivoli suite of systems management products

Let's begin by looking at how you can configure a Tivoli Storage Manager solution. We recommend that each and every system in your configuration be backed up. The frequency of backup will vary, depending on the type of information and the frequency of changes.

Figure 78 shows the recommended Tivoli components for deployment in the internal network nodes.

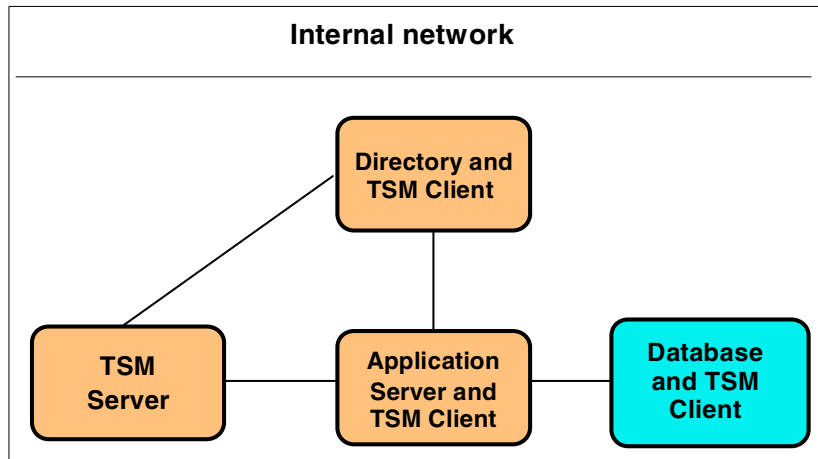


Figure 78. TSM server and client setup in the internal network

Referring to Figure 78, let's assume you install one TSM server in the scenario. This server should be located in the internal network with no external Internet access to it.

Once the TSM server setup is completed successfully, install TSM clients at every system that you would like to back up. For example, we have installed TSM clients at the directory server, application server and the shared file system server. Test the connectivity between the TSM clients and the TSM server by doing a user-initiated backup.

For the servers in the DMZ (see Figure 79), you can also install a TSM client. To facilitate communication between the TSM client in the DMZ and the TSM server in the intranet, you will have to open up one port in the firewall. This port can be pre-configured in both the TSM client and server.

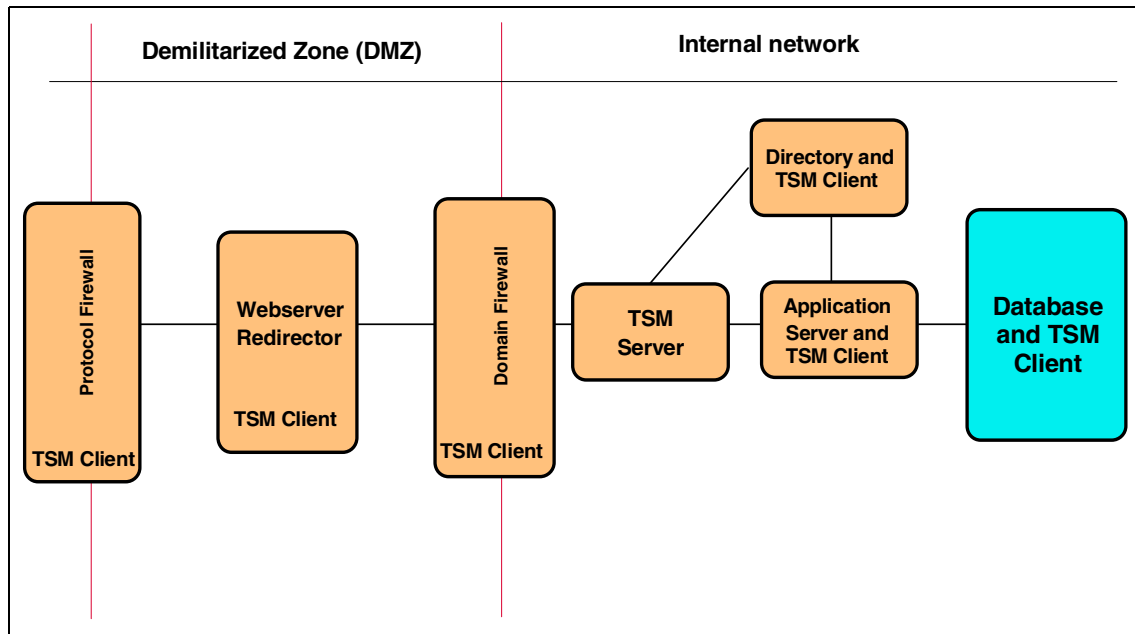


Figure 79. TSM server and client setup in the DMZ and internal network

If this additional firewall port is a security issue in your environment, then there are two viable options to consider:

- Install Tivoli Data Protection for Workgroups (TDPfW) for Windows NT systems.

TDPfW provides a stand-alone disaster recovery for Windows NT machines. It can back up entire Windows NT machines or volumes, and if a disaster occurs, TDPfW can restore the complete machine, including the boot volume, disk partitions, security, operating systems, and user files from a locally attached SCSI tape drive.

This is very useful for small LAN environments (for example, the DMZ) where you have to manage only a few servers. However, this product currently is supported on Windows NT only.

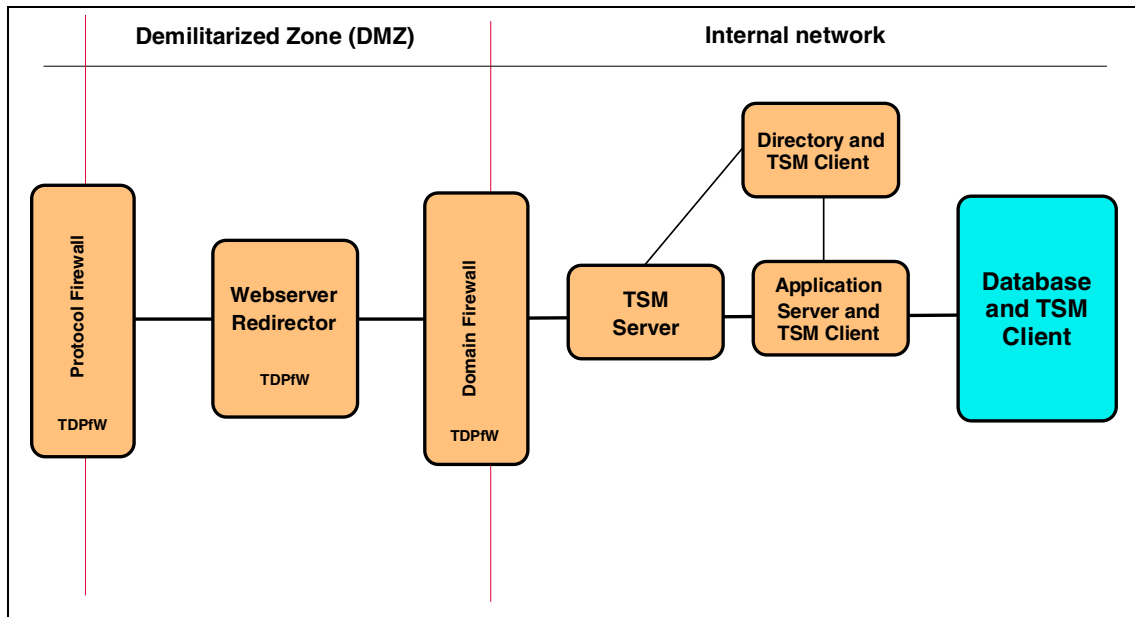


Figure 80. TSM server in the internal network with TDPfW in DMZ

- Installing a TSM server in your DMZ

To facilitate a more automated solution, you could install another TSM server in the DMZ. Keep in mind, that as more servers are added to the DMZ, the overhead involved in administering each TDPfW server increases.

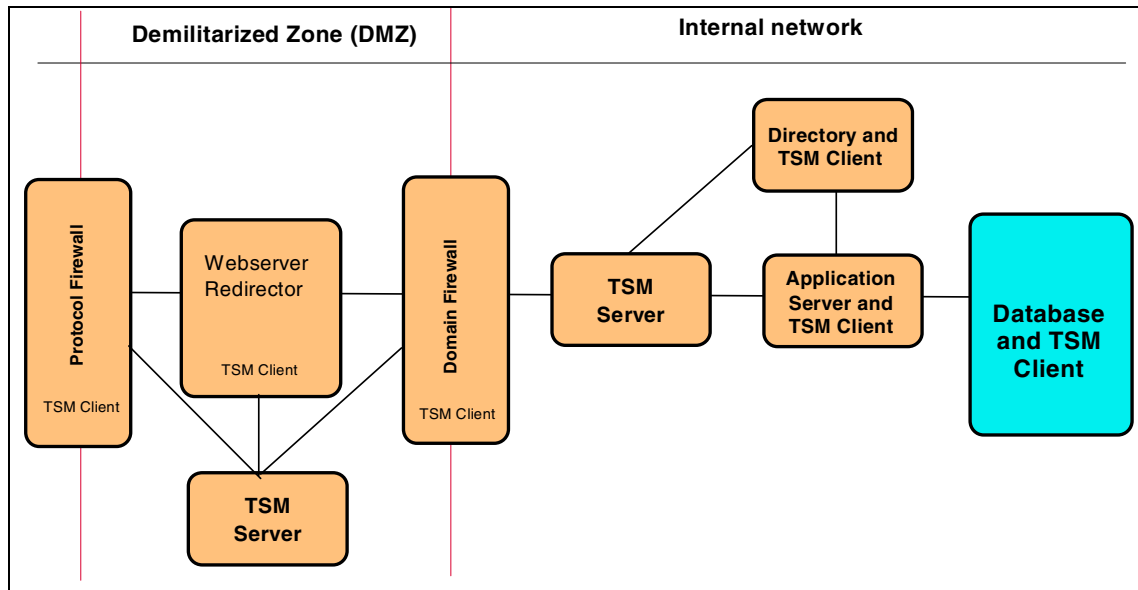


Figure 81. One TSM server in the DMZ and each internal network

9.3.2 Application backup and recovery

We have discussed general architecture for developing a backup solution. In this section, we will highlight product specific areas to look out for.

9.3.2.1 Operating system

In a total system recovery scenario, the operating system (OS) will be the first software to be reinstalled. You should refer to your OS instruction manual for re-install steps. Our checklist below highlights practical tips for OS backup management:

- Do you have and know where the OS media is located?
- Do you have new updates to the OS?
- Do you have an OS backup after initial system setup?
- Do you do regular OS backup?
- Do you do ad-hoc OS backup before major installation and after major configuration changes?

9.3.2.2 WebSphere database

When you install and configure the WebSphere Application Server, SecureWay Directory and WebSphere Site Analyzer, you will notice that they use DB2 as their database. In this section, we will only briefly illustrate how a

DB2 backup can be performed. For a comprehensive review of DB2 backup and recovery, please refer to the DB2 administrative guides and references.

In DB2, you can perform either an online or offline backup. When you do an offline backup, the database needs to be shut down. The command for an offline backup is:

```
db2> backup db <instance name> offline=yes
```

If shutting down the database is not an option, then you will have to perform an online database backup:

```
db2> backup db <instance name> online=yes
```

When performing an online database backup, you will have to take into consideration how the database logs will be managed, as they need to be used in a recovery process. Losing the logs will complicate the recovery process. When you use TSM, you will enjoy an automated database log backup solution. This can be achieved by:

- Configuring the DB2 user exit program to utilize TSM.
- Turning on the `userexit` and `logretain` parameters in DB2 database manager.

9.3.2.3 LDAP database backup

With the IBM SecureWay Directory, you can back up the LDAP database from the GUI interface, the command line, or via native DB2 commands.

Using the GUI (see Figure 82 on page 227):

1. Click **Database** in the Navigation frame.
2. Click **Backup** in the Working with back-end database in the work area.
3. Type the fully qualified name of the file to be created in the text entry field. This file will be in LDIF format.
4. Click the **Backup database** button to start backing up the database.

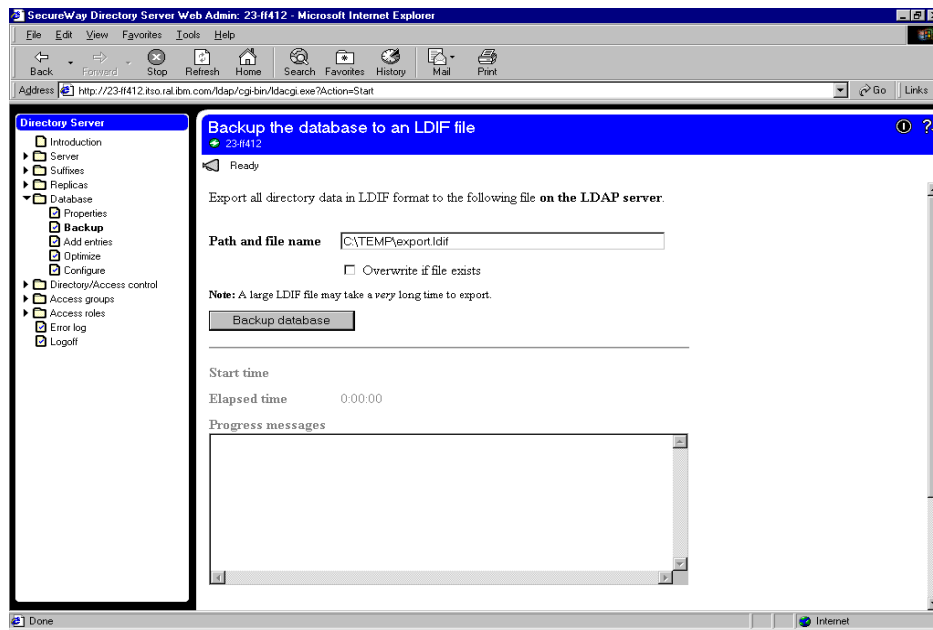


Figure 82. LDAP database backup GUI

Using the command line, the db2ldif tool can be used to dump entries from a directory to a text file in LDIF format. The syntax is:

```
c:\> db2ldif -o <output file> [-s <subtree>]
```

Note: LDIF format does not honor any ACL settings and all data is available in readable text.

To use the DB2 database backup, see 9.3.2.2, “WebSphere database” on page 225.

9.3.3 Guidelines for backup and recovery

Independent of your backup software choice, there are some guidelines you can follow:

1. Monitor the backup process.

Ensure that the backup process is successful. If the backup process fails, understand why it fails and take necessary actions to remedy. For example, is the backup failing due to dirty drives or faulty media? Perhaps, there is a problem with the hardware? Could it also be a loss of connectivity between your backup server and the client machines?

2. Properly manage the backup media.

After taking a backup of the system, ensure you have a proper and systematic tracking system for your backup media. In the event of a disaster, can you easily identify and retrieve these backup media for recovery?

3. Test your backup.

Do not assume that if the backup process is successful, you have a valid backup. Test your backup! This will give you additional confidence that your backup can really help you recover from a disaster.

4. Document, test and maintain a recovery plan.

No matter what size your system configuration may be, always have a documented recovery plan. This will prove useful during a disaster situation. The plan should describe step-by-step the process you will take to recover. Test the plan to ensure that it works! For the document to be useful at all times, you will have to maintain and update it when configuration changes occur.

9.4 Where to find more information

- Nagaratnam, Nataraj et al., *Security Overview of IBM WebSphere Standard/Advance 3.02*, IBM white paper 2000

Part 3. Application topology 1: a working example

Chapter 10. The Pattern Development Kit and an example topology

The next chapters will describe how to take a runtime topology and modify it to suit your needs. As our example, we will use an extension of the runtime topology found in 3.2.3, “Emerging variation 2” on page 27. We use that basic idea and extend it by adding application server clones distributed over multiple platforms.

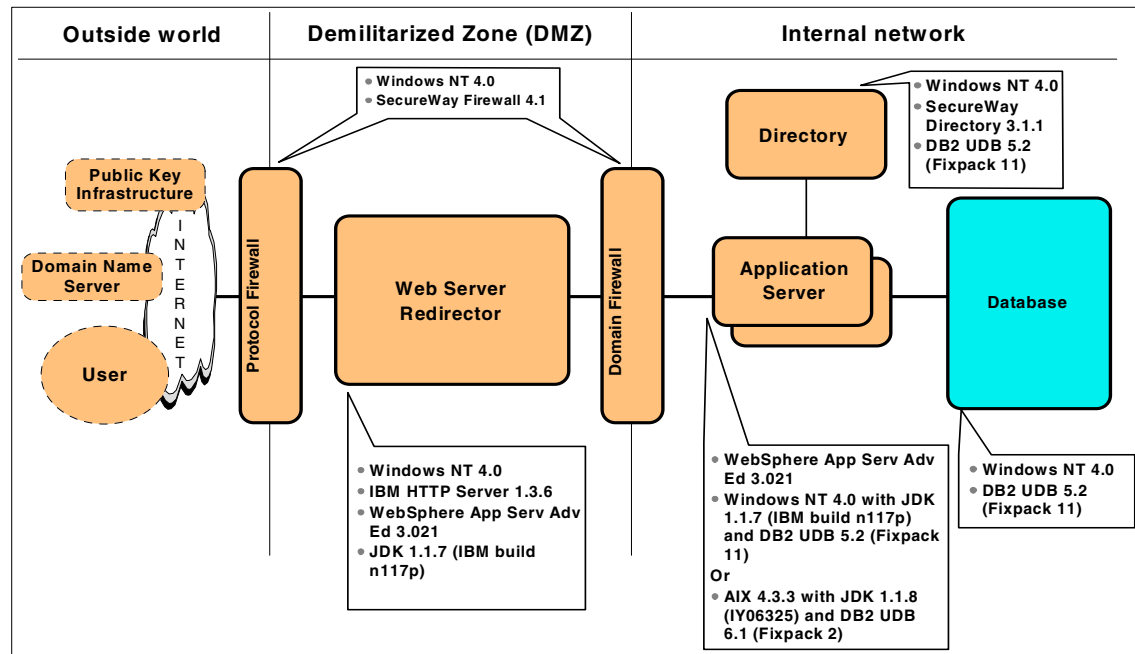


Figure 83. Product mapping, extended emerging variation 2 of runtime topology A

In this scenario, a standalone (“thin”) servlet redirector is serving several clones of an application server on both AIX and Windows NT servers.

- The protocol firewall was configured to be open on port 80 only. Thus, only HTTP traffic could flow from the Web browsers to the Web server.
- The domain firewall was configured to be open on the ports needed to allow IIOP traffic to flow between the redirector and the application servers.
- The application servers were hosting the business logic that accessed data stored in the database within the internal network.
- User authentication was implemented using WebSphere’s security features and users had to authenticate against the LDAP server.

- Each of the application servers had the same data and code installed, eliminating the need to install a shared file system.

Our example will show the following:

- Modifying the Pattern Development Kit application code using VisualAge for Java and WebSphere Studio
- Setting up the Web server and servlet redirector
- Setting up the WebSphere Application Server
- Cloning the application server
- Security, including setting up firewalls and configuring WebSphere to use LDAP

10.1 The Pattern Development Kit

For our sample code we took an application from the IBM Pattern Development Kit (PDK) and modified it. The Pattern Development Kit is designed to provide an overview of WebSphere Advanced Edition in a working environment. The PDK provides a single-machine setup with applications and product code. The kit is divided into sections, each showing a different aspect of the Web application.

- Section A shows an example of Web access to a read-only DB2 database and a writeable DB2 database.
- Section B shows an example of custom authentication using an LDAP directory.
- Section C shows an example of Web access to a read-only DB2 database. This section generates a dynamic menu based on a user profile stored in the LDAP directory.
- Section D shows an example of Web interactivity with MQ and CICS back-end applications.
- Section E shows an example of Web access to an IMS and DB2 back-end.
- Section F shows the EJB implementation of the command manager pattern.

The PDK is scheduled to be available in July 2000. When it becomes available, information about it can be found at:

<http://www.ibm.com/software/developer/web/patterns/>

10.2 PDK section A

Our sample application will be based on the application from Section A of the Pattern Development Kit, which illustrates application topology 1. We will show you how we modified it to fit our needs. This included reworking the application slightly and separating the implementation among several machines instead of using the single-machine setup used by the PDK.

The original application used by the PDK displays inter-planetary weather information to the user. Section A is structured in the following way:

1. A splash screen is initially displayed in the user's browser.

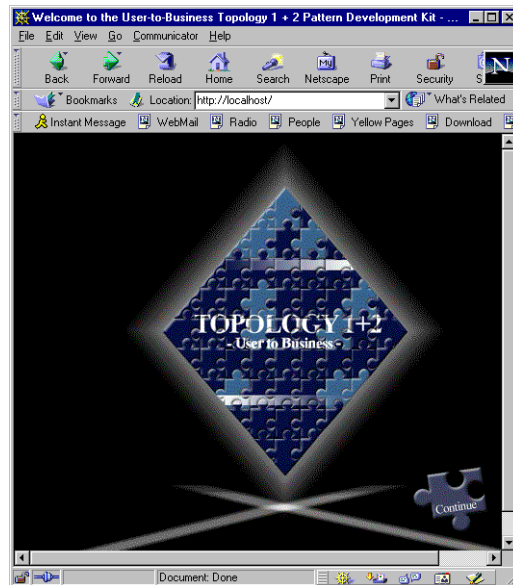


Figure 84. PDK initial screen

2. A link on the splash screen leads the user to a frameset (two frames) that contains a menu on the left and a contents page on the right of the browser window.



Figure 85. PDK menu

The contents page displays basic information about the sample site. In a production site this page could be replaced with company information, site information, etc.

The frame on the left of the browser window contains the menu page. By clicking on a menu item the user can navigate to other areas of the site.

3. The Topology 1 menu item leads the user to a JSP data entry page. From this point on, you are in WebSphere Web application called "topologyone".

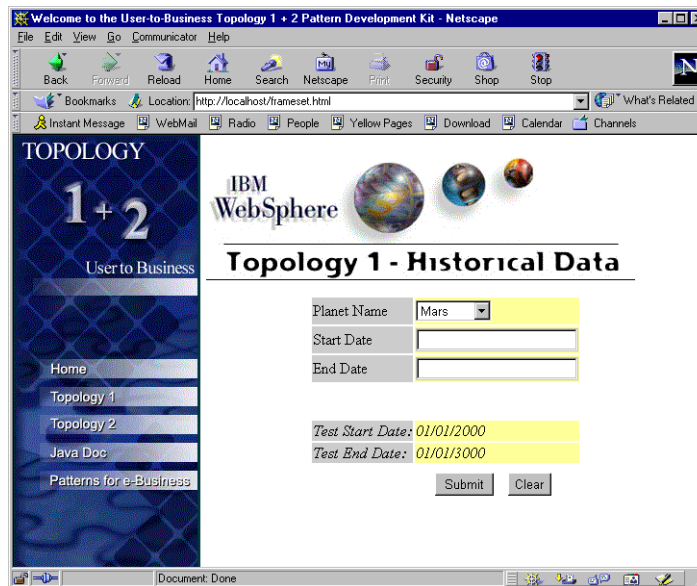


Figure 86. PDK topology 1 application

4. The user enters information in the input fields. The application uses this input to retrieve the correct data from a DB2 database.
5. For XML-enabled browsers, the weather data will be returned in XML format. For browsers that are not XML-enabled, the data will be returned in a simple table.

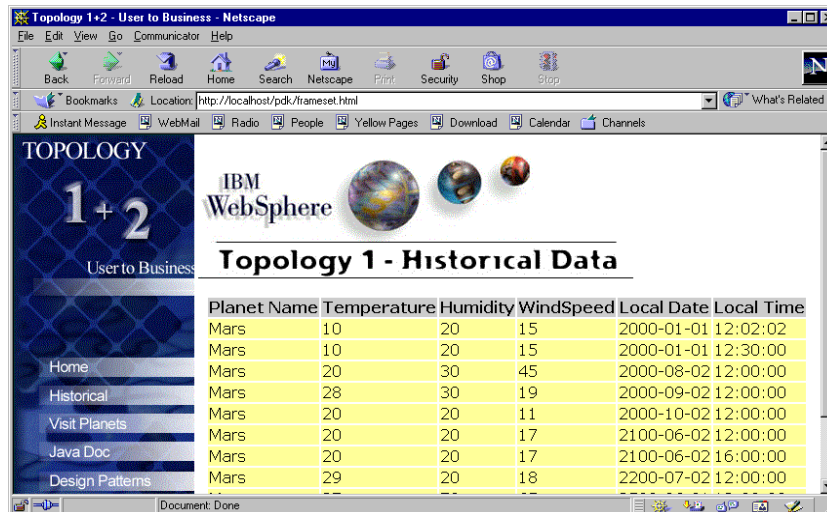


Figure 87. Topology 1 application output - table format

10.2.1 PDK application interaction

The following figure illustrates the internal workings of the topologyone application in the PDK's section A.

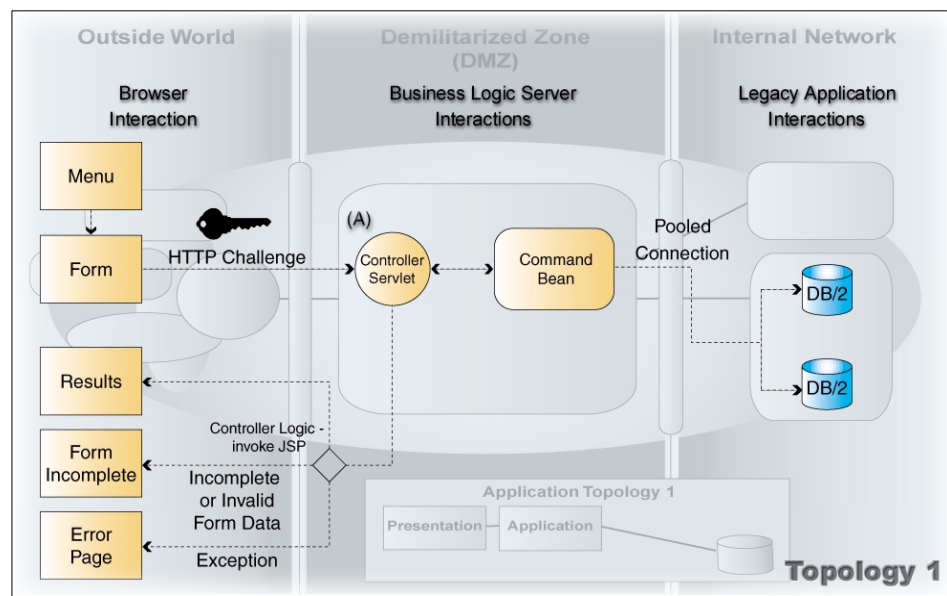


Figure 88. PDK section A application component interaction

1. A Web user clicks Topology 1 on the vertical menu that is displayed on the left hand frame of the frameset.
2. After clicking the menu item the user is presented with a JSP form. Once the input fields have been entered the user clicks the **Submit** button on the JSP form.
3. The parameters are passed to the controller servlet via a query string. The controller servlet is a secured resource; therefore the Web browser challenges the user to enter a valid user ID and password (basic authentication).
4. Once the user has been successfully authenticated, the controller servlet creates two command bean instances.
5. One instance is delegated the responsibility of reading data from a read-only DB2 database (HISTDATA) and the other is required to write a journal record to a second DB2 database (JOURNAL).
6. The controller servlet retrieves the required data from the commands and passes it into the HTTPRequest object.
7. The controller servlet then invokes a JSP to display the retrieved data. If no data is returned from the appropriate command the controller servlet will display a no data page. If an error occurs the controller servlet will also inform the Web user of the error. The Web user can then try the action again.
8. The Command beans instances use pooled database connections. Their lifetime is bound to that of the request.

We will change this slightly to fit our runtime topology layout for the emerging variation 2 discussed in 3.2.3, “Emerging variation 2” on page 27.

Chapter 11. Step 1: Modifying the PDK application

To be able to better understand the Pattern Development Kit, you should use the appropriate tools to look inside the source code for all the Pattern Development Kit's artifacts. This will be a high-level look at working with the PDK using VisualAge for Java and WebSphere Studio. Detailed information on using these tools can be found in *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755.

11.1 Using the Pattern Development Kit in VisualAge for Java

If you are interested in the Java source code (servlets, beans, and other classes) of the PDK, use VisualAge for Java to view and modify the code.

The following steps are necessary to get the Pattern Development Kit's Java source code inside the IDE of VisualAge for Java:

1. Add the following features to the workspace:
 - CICS Connector 3.0.4
 - Data Access Beans 3.0
 - IBM Common Connector Framework 3.0
 - IBM EJB Development Environment 3.0
 - IBM Enterprise Access Builder Library 3.0
 - IBM Enterprise Data Access Libraries 3.0
 - IBM IDE Utility class libraries 3.0
 - IBM Java Record Library 3.0
 - IBM WebSphere Test Environment 3.0
 - IMS TOC Connector 1.1
 - MQSeries Connector 1.1
 - Sun Servlet API 2.1
2. Get the VisualAge for Java repository file, called U2BTOP.dat, from the Pattern Development Kit's CD, found in
`Sdk\U2BTop\TestDrive\coding\artifacts\repository`
3. Import the Pattern Development Kit project from the U2BTOP.dat repository file into the VisualAge for Java repository
4. Add the Pattern Development Kit project to the workspace

11.1.1 Changing the PDK application in VisualAge for Java

In this example, we have used VisualAge for Java to create a simpler version of PDK Section A. The new version fetches historical data but does not update the journal. We also had to modify the database call in the Command

bean to use a user ID and password since the database will eventually be moved to a separate machine.

The portion of the RetrieveHistoricalDataServlet that updated the journal has been removed. This was for simplicity in the lab environment. The new servlet looks like the following:

```

package com.ibm.hursley.asg.ws.skeleton.topologyone.sectiona;
/**
 * RetrieveHistoricalDataServlet
 * * @versionPatterns Solution Kit 1.0.1
 * * @authorASG IBM Hursley, Florian Hilgenberg IBM GS Germnay
 * */
/*
 * Packages required by all servlets
 */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.hursley.asg.ws.skeleton.view.*;
import com.ibm.hursley.asg.ws.skeleton.command.*;

public class RetrieveHistoricalDataServlet extends HttpServlet{

/**
 * RetrieveHistoricalDataServlet constructor comment.
 */
public RetrieveHistoricalDataServlet() {
super();
}
/*
 */

public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException{
performTask(req, res);
}
public void doPost(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
performTask(req, res);
}
public void init(ServletConfig config) throws ServletException {
super.init(config);
}
}

```

Figure 89. *RetrieveHistoricalDataServlet* - part 1

```

public void performTask(HttpServletRequest req, HttpServletResponse
res) {

RetrieveHistoricalDataCommand dataCommand = null;

try{
try{

dataCommand = (RetrieveHistoricalDataCommand) this.getClass()

.getClassLoader().loadClass("com.ibm.hursley.asg.ws.skeleton.topologyone
e.sectiona.RetrieveHistoricalDataCommand").newInstance();

} catch(Exception e) {

/*
* Uses this call in VAJ Environment
*/
dataCommand = (RetrieveHistoricalDataCommand) java.beans.Beans
.instantiate(this.getClass().getClassLoader(),

"com.ibm.hursley.asg.ws.skeleton.topologyone.sectiona.RetrieveHistorica
lDataCommand");
}

/*
* Retrieve the form fields
*/
dataCommand.setPlanetName(req.getParameter("planetName"));
dataCommand.setStartDate(req.getParameter("startDate"));
dataCommand.setEndDate(req.getParameter("endDate"));

/*
* Retrieve the data
*/
try {

dataCommand.execute();

} catch (CommandException e) {

/*

```

Figure 90. RetrieveHistoricalDataServlet - part 2


```

    * Display a no data returned page
    */

    getServletConfig().getServletContext().getRequestDispatcher("sectionA/n
    oreturndata.jsp").forward(req, res);
    return;
}

/*
    * Set the Request object attribute
    */
    req.setAttribute("results", dataCommand.getData());

    /* Forward to HTML View JSP */

    getServletConfig().getServletContext().getRequestDispatcher("sectionA/s
    ectiona.jsp").forward(req, res);

    } catch(Throwable t) {
    try {

        getServletConfig().getServletContext().getRequestDispatcher("sectionA/e
        rror.jsp").forward(req, res);
    } catch (Exception e){
        log(e.getMessage());
    }
    }
    }
    }
}

```

Figure 91. RetrieveHistoricalDataServlet - part 3

The RetrieveHistoricalDataCommand bean was modified to hard code (for testing purposes) the user ID and password for the historical database.

```
conn = ds.getConnection("USERID", "PASSWORD");
```

In general, it is good idea to get the PDK application code running in the WebSphere Test Environment of VisualAge for Java for testing.

To deploy the changed application, simply export the changed classes to a directory as both Java and class files. Then copy these files to their respective directories in the WebSphere Application Server. Restart the Web

application server hosting the PDK's Web application in the WebSphere administration console.

11.2 Using the PDK in WebSphere Studio

If you are interested in the Web site source code of the Pattern Development Kit (the HTML files, JSP files, and image files) use WebSphere Studio as the tool to view and modify the code.

The following steps are necessary to get the Pattern Development Kit's Web site source code inside the IDE of WebSphere Studio:

1. Get the artifacts from the Pattern Development Kit CD located in:
`C:\Asg\Sdk\U2BTop\TestDrive\config\artifacts`
2. Open the project file, called U2BTop.wao, with WebSphere Studio by double clicking on the file in Windows NT explorer.

11.2.1 Changing the PDK application with WebSphere Studio

We changed the PDK Section A application from using a dynamic start page (JSP) to using a static page (HTML with JavaScript) because we wanted to call the servlet from an HTML file and not from a JSP file.

We used the WebSphere Page Designer to edit the HTML and JSP files as follows:

1. First, we created a sectionaForm.html file to replace the initial interface into the topologyone application code, using the sectionaForm.jsp file as a template. We did this by taking sectionaForm.jsp, changing its file type from .jsp to .html, and deleting all JSP scripts from the file. We then added JavaScript code into the file to achieve the input format checking locally in the user's browser. The resulting HTML source is shown below.

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<META name="GENERATOR" content="IBM WebSphere Page Designer V3.0.1 for
Windows">
<META http-equiv="Content-Style-Type" content="text/css">
<TITLE> WebSphere Skeleton Topology 1 - Historical Data </TITLE>
</HEAD>

<BODY>

<TABLE cellpadding="0" cellspacing="0">
  <TBODY>
    <TR>
      <TD><IMG src="images/T1HistData.jpg" width="450" height="148"
border="0"></TD>
    </TR>
  </TBODY>
</TABLE>

<P>

<SCRIPT>
function checkData () {
dateRE = new RegExp("\\d{1,2}/\\d{1,2}/\\d{2,4}");
if (!dateRE.test(document.inputForm.startDate.value)) {
alert("Start date not valid.\n(Format is: dd/mm/yyyy)");
return false;
}
if (!dateRE.test(document.inputForm.endDate.value)) {
alert("End date not valid.\n(Format is: dd/mm/yyyy)");
return false;
}
return true;
}
</SCRIPT>

```

Figure 92. Source code of sectionForm.html file - part 1

```

<!-- Form begins here -->
<FORM NAME="inputForm" METHOD=GET
ACTION="/webapp/topologyone/histData">

<!-- This table is inside the Form -->
<CENTER>
<TABLE border="0">
  <TBODY>
    <!-- Row 1 Planet Name -->
    <TR>
      <TD bgcolor="#cccccc">Planet Name</TD>
      <TD bgcolor="#ffff80">
        <SELECT name="planetName" onclick="window.status = 'Select A
Planet';">
          <OPTION value="Mars" selected>Mars</OPTION>
          <OPTION value="Saturn">Saturn</OPTION>
          <OPTION value="Mercury">Mercury</OPTION>
          <OPTION value="Moon">Moon</OPTION>
          <OPTION value="Pluto">Pluto</OPTION>
        </SELECT>
      </TD>
    </TR>
    <!-- Row 2 Start Date -->
    <TR>
      <TD bgcolor="#cccccc">Start Date</TD>
      <TD bgcolor="#ffff80"><INPUT size="20" type="text"
onclick="window.status = 'Enter a Start date'; return true;"
name="startDate" value=""></TD>

    </TR>
    <!-- Row 3 End Date -->
    <TR>

      <TD bgcolor="#cccccc">End Date</TD>
      <TD bgcolor="#ffff80"><INPUT size="20" type="text"
onclick="window.status = 'Enter an End Date'; return true;"
name="endDate" value=""></TD>
    </TR>

```

Figure 93. Source code of sectionForm.html file - part 2

```

<!-- Blank Row-->
<TR>
  <TD height="10"></TD>
  <TD height="10"></TD>
</TR>
<!-- User Prompt -->
<TR>
  <TD bgcolor="#cccccc"><I>Test Start Date:</I></TD>
  <TD bgcolor="#ffff80"><I>01/01/2000</I></TD>
</TR>
<TR>
  <TD bgcolor="#cccccc"><I>Test End Date:</I></TD>
  <TD bgcolor="#ffff80"><I>01/01/3000</I></TD>
</TR>
<!-- Buttons -->
<TR>
  <TD height="10"></TD>
  <TD>
    <CENTER>
      <TABLE cellpadding="4" cellspacing="1">
        <TBODY>
          <TR valign="middle" align="center">
            <TD width="60"><INPUT type="submit" onclick="window.status =
'Press To Submit Form'; return checkData();" name="submit"
value="Submit"></TD>
            <TD width="60"><INPUT type="reset" onclick="window.status =
'Press To Clear Form'; return true;" name="reset" value="Clear"></TD>
          </TR>
        </TBODY>
      </TABLE>
    </CENTER>
  </TD>
</TR>

</TBODY>
</TABLE>
</CENTER>

<!-- The form ends here -->
</FORM>

```

Figure 94. Source code of sectionForm.html file - part 3

```

<P>

<!-- A new table -->
<TABLE border="0">
  <TBODY>
    <!-- Row 1 -->
    <TR>
      <TD><H2>What is Happening ?</H2></TD>
    </TR>
    <!-- Row 2 -->
    <TR>
      <TD align="center"><IMG src="images/sectionAWhatis.jpg"
width="500" height="264" border="0"></TD>
    </TR>
    <!-- Row 3 -->
    <TR>
      <TD></TD>
    </TR>
  </TBODY>
</TABLE>
</BODY>
</HTML>

```

Figure 95. Source code of sectionForm.html file - part 4

2. Next, we copied sectionaForm.html to the Web server in the IBM HTTP Server\htdocs\u2btopsamplesite directory.
3. Next, we copied the graphic files used by the html form (T1HistData.jpg and sectionaWhatis.jpg) to the Web server in the IBM HTTP Server\htdocs\u2btopsamplesite\images directory.
4. Last, we changed the menu.html file on the Web server in the IBM HTTP Server\htdocs\U2bTopWebApp directory so that every link to sectionaForm.jsp file (../webapp/topologyone/sectionA/sectionaForm.jsp) now points to sectionaForm.html.

Chapter 12. Step 2: Expanding the PDK to multiple machines

The Pattern Development Kit is for demonstration purposes and is designed to run on a single machine.

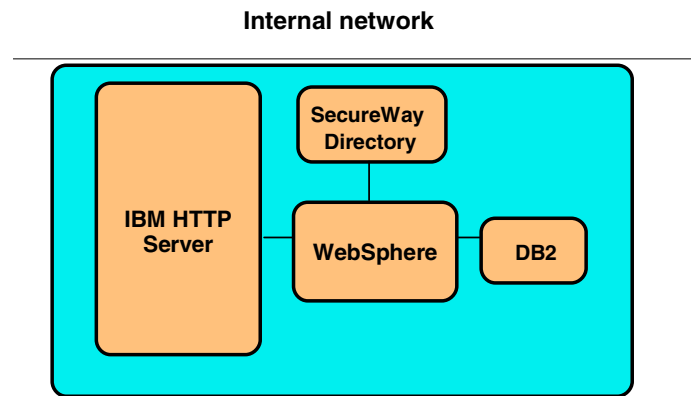


Figure 96. PDK Section A application on one machine

Obviously, this setup is not very realistic for production purposes. Normally, the different components of the Pattern Development Kit would run on separate machines, with the Web server on one machine, the application server on another, and the database on a third.

In this chapter we will outline the steps used to spread the topologyone application function across a distributed environment, including how to create a Web server, application server, and the database. The business logic portion of the application usually has access to business data and even back-end systems. This part of the application should be running in a secure environment.

Note: This assumes that you have installed the PDK on a test machine and you are now going to create a new test environment with two new machines. It also assumes you have made the application changes described in Chapter 11, “Step 1: Modifying the PDK application” on page 239.

12.1 Setting up the network environment

The remaining parts of the example in this and the following chapters assume that the proper networking environment has been set up. This means that two

firewalls have been introduced into the environment, creating three separate networks:

- The internal (secure) network
- The DMZ
- The outside (unsecure) network

It also assumes that the IBM SecureWay directory will be used for the security registry and that it has been set up in the secure network.

If you are using these instructions to create your own test environment, you may choose to do this at any time, but you may have to adjust the following instructions slightly for your own use.

Directions for setting up this environment can be found in:

- Chapter 15, “Setting up a standalone servlet redirector” on page 297
- Chapter 16, “Setting up firewalls” on page 305
- Chapter 17, “SecureWay Directory Configuration” on page 325

12.2 Separating the application server from the Web server

The first step in our configuration was to create a Web server. We are going to introduce a new machine (Machine A) into the configuration and install the Web server and Web server redirector on it.

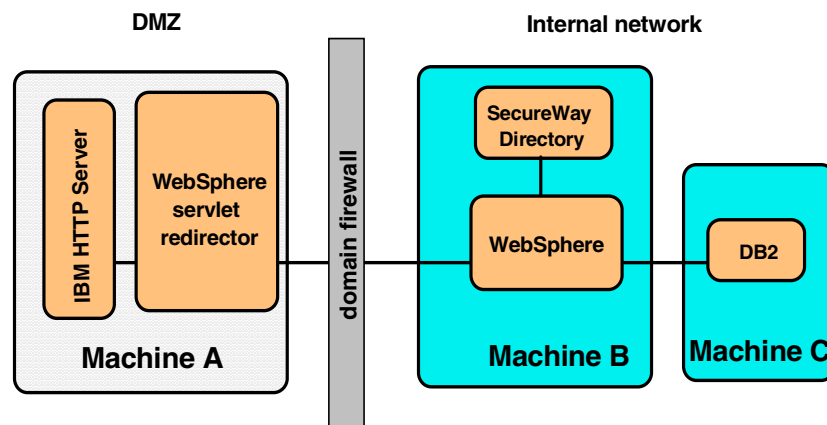


Figure 97. New configuration

The steps to set up Machine A are:

1. Install the IBM HTTP Server on Machine A.

2. Copy the static contents of the Pattern Development Kit from the application server to the Web server root directory. These files can be found in the documents directory of the Web server. For the IBM HTTP Server with the PDK installed, this will be the \IBM HTTP Server\htdocs\u2btopsamplesite directory. The files are:
 - index.html
 - menu.html
 - content.html
 - frameset.html
 - \theme\Master.css
 - all image files in \images
3. In Chapter 11, “Step 1: Modifying the PDK application” on page 239, we modified the PDK application code so that the start page was not a JSP, but an HTML file with JavaScript. The new HTML file calls the servlet.

Copy the following files into the IBM HTTP Server\htdocs\u2btopsamplesite directory:

- sectionaForm.html
- images/T1HistData.jpg
- images/sectionAWhatis.jpg

Note

The example assumes that you are going to move the HTML files into the root directory of the Web server (usually htdocs) and the image files into the images subdirectory. This will replace index.html, making the initial entry into the Web server the topologyone application index page. As an alternative, you could copy the u2btopsamplesite directory as a subdirectory and make an alias in the HTTP Server configuration file (httpd.conf) to point to it.

12.3 Setting up the application server on Machine B

Next, we need to create an application server machine in the secure network. This will involve installing the appropriate product code and installing the modified version of the PDK topologyone application.

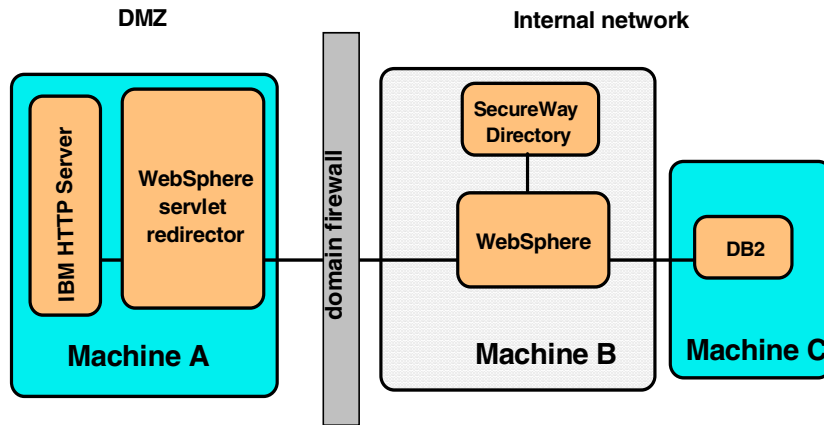


Figure 98. New configuration

Prepare machine B using the following steps:

1. Install the IBM WebSphere Application Server code.
2. Start the WAS administration server (IBM WS AdminServer) service from the services window in the Windows NT control panel.
3. Once the service has started, start the WAS configuration console (**Start -> Programs -> IBM WebSphere -> Application Server V3.0 -> Administrator's Console**)

You will use the administrator's console to perform the next tasks.

12.3.1 Creating the JDBC driver and DataSource definition

In order to use the DB2 application database, you will need to define a JDBC driver and you will need to define the database as a DataSource.

1. Under the Types tab highlight **JDBC Drivers**, right click, and select **Create....** Fill in the window with the values shown in Figure 99.

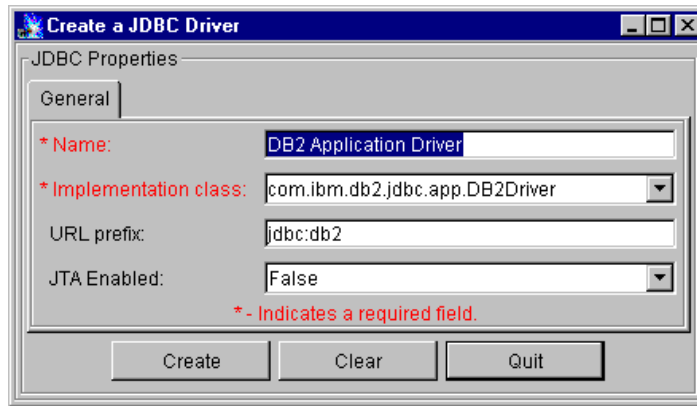


Figure 99. Create a JDBC Driver

Click **Create**.

2. Under the Types tab highlight **DataSources**, right click, and select **Create...** Fill in the window with the values shown in Figure 100.

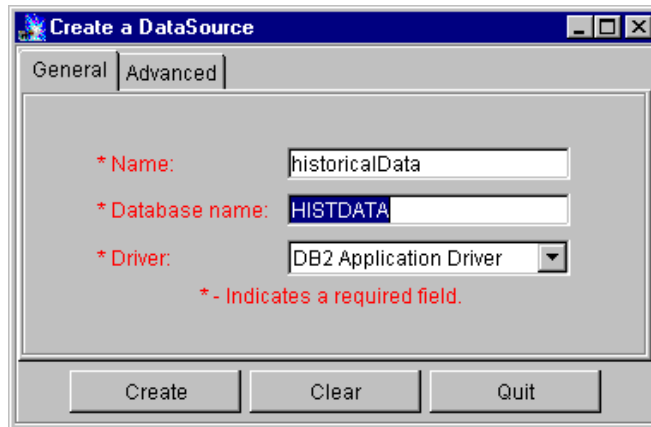


Figure 100. Create a DataSource

Specify `historicalData` as the name of the data source. This name is used in the `RetrieveHistoricalDataCommand` Java code.

Use the database name, `HISTDATA`. This is the name of the database that comes with the PDK. Later, in 12.4, “Separating the database from the Web application server” on page 268, you will create this database.

Choose the driver you just defined in step 1 (Figure 99).

Click **Create**.

Note: We do not create a DataSource for the second database in the PDK application since our modified code does not create journal entries.

12.3.2 Create an application server

The next step is to create an application server called Topology One.

Note: The PDK does not do this. It uses the default server.

1. Under the Tasks tab, expand **Configuration**. Then select **Configure an application server** and click the **Start Task** button (green circle on the smart-icon bar).

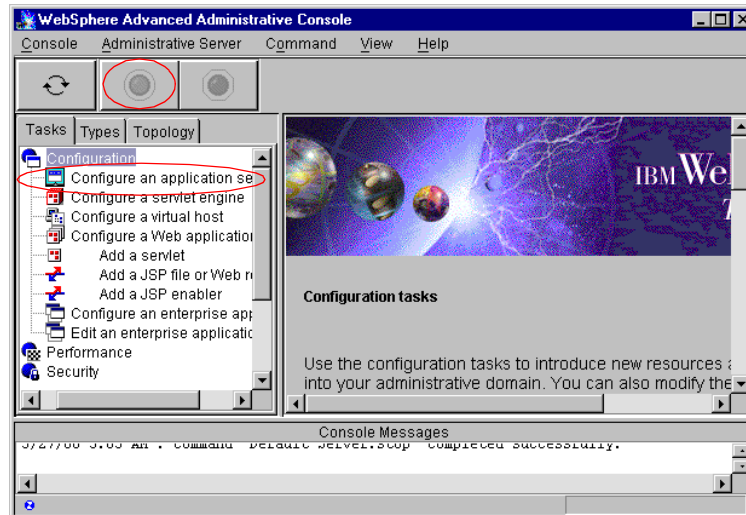


Figure 101. Configure an application server

2. On the first page select **Web Applications** as the needed resource type (uncheck Enterprise Beans).

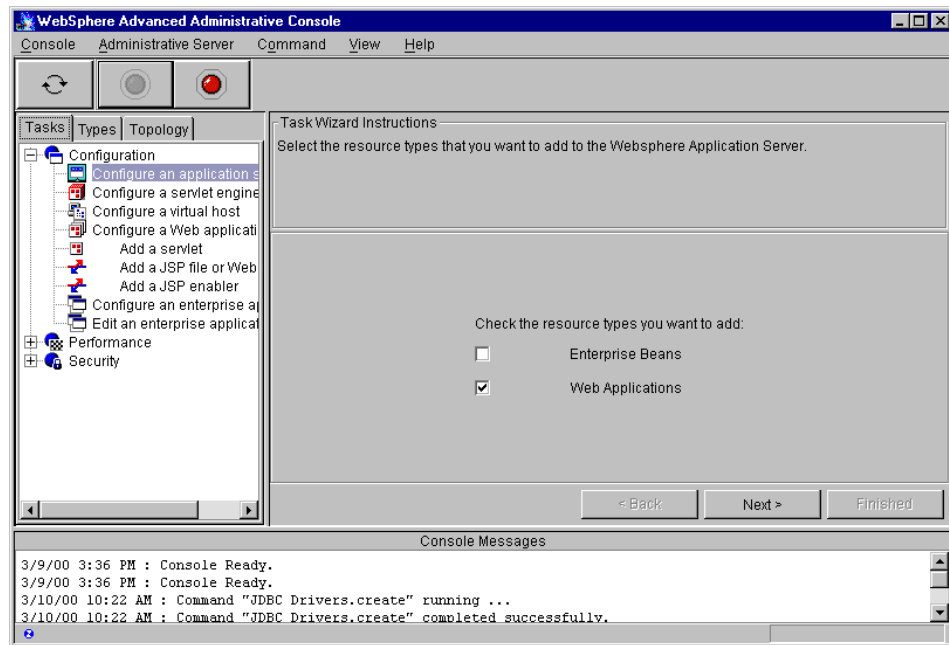


Figure 102. Configure an Application server - resource types

Click **Next**.

3. On the next page, specify a name for the application server. In our case we chose "Topology One".

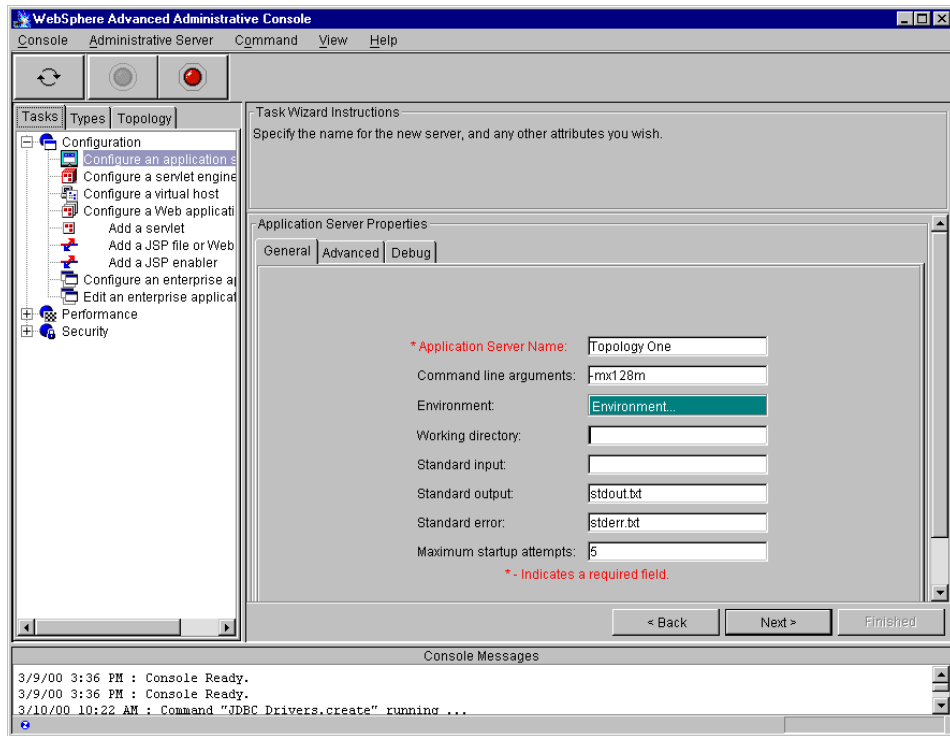


Figure 103. Configure an Application server - Application Server Properties

Click **Next**.

4. On the next page, take the defaults for the start behavior.

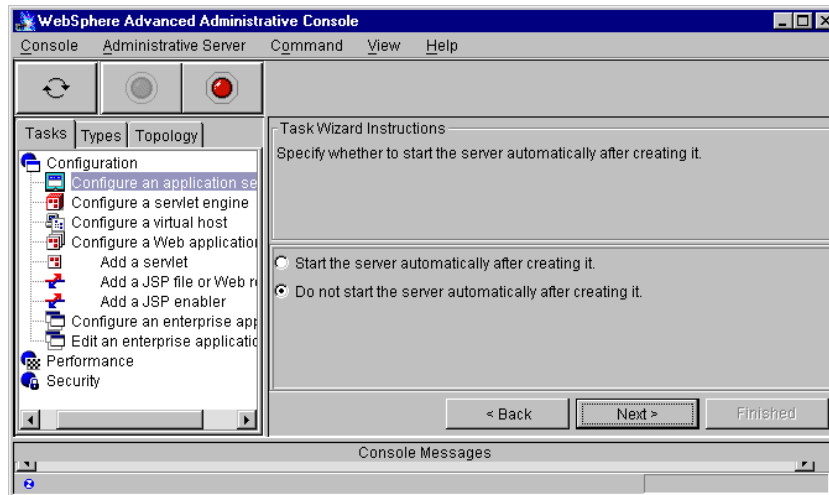


Figure 104. Configure an application server - start behavior

Click **Next**.

5. On the next page, select the machine's default node. In this case the machine name is 23bk63z.

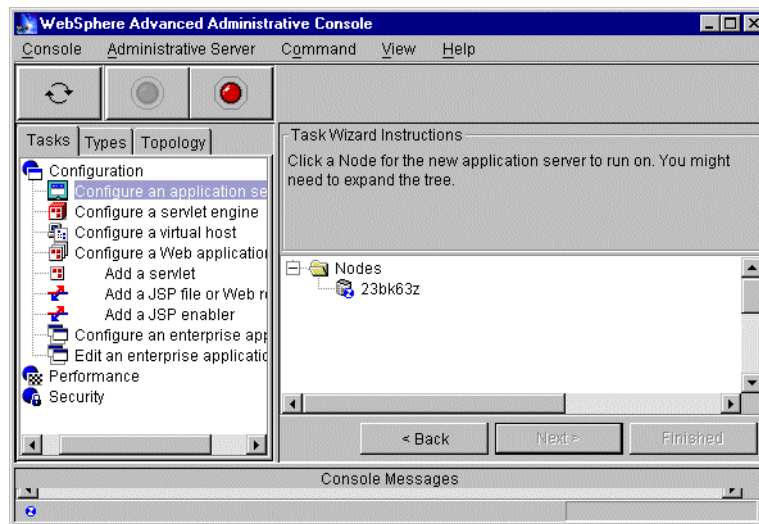


Figure 105. Configure an application server - node selection

Click **Next**.

6. On the next page, choose the default_host as the virtual host.

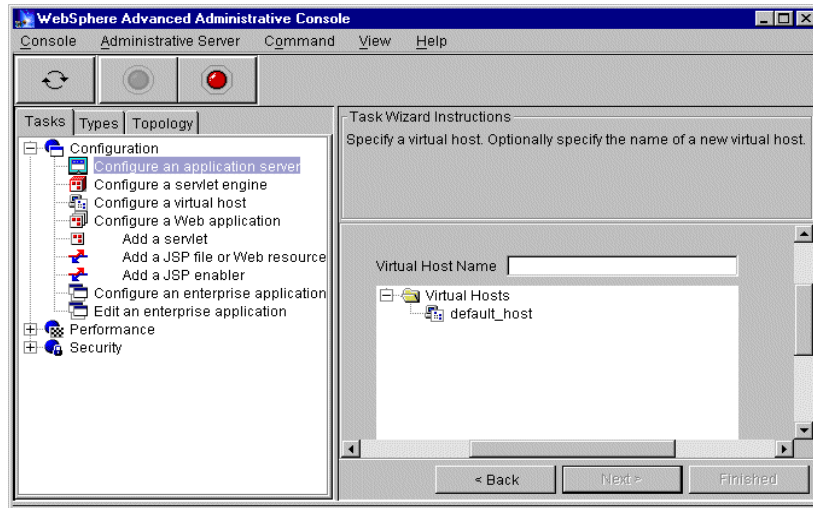


Figure 106. Configure an application server - virtual host selection

Click **Next**.

7. On the next page, specify a name for the servlet engine. A default name will be filled in for you. We chose to use the default.

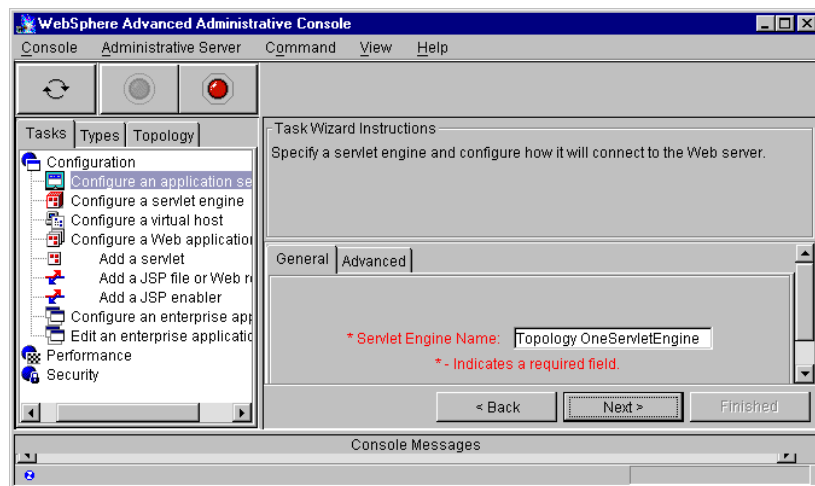


Figure 107. Configure an application server - servlet engine properties

Click **Next**.

8. On the next page, specify a name for the Web application (once again, a default will be filled in for you) and specify `/webapp/topologyone` for the application Web path. This is important because it must match the name coded in the servlet call in `sectionaForm.html`.

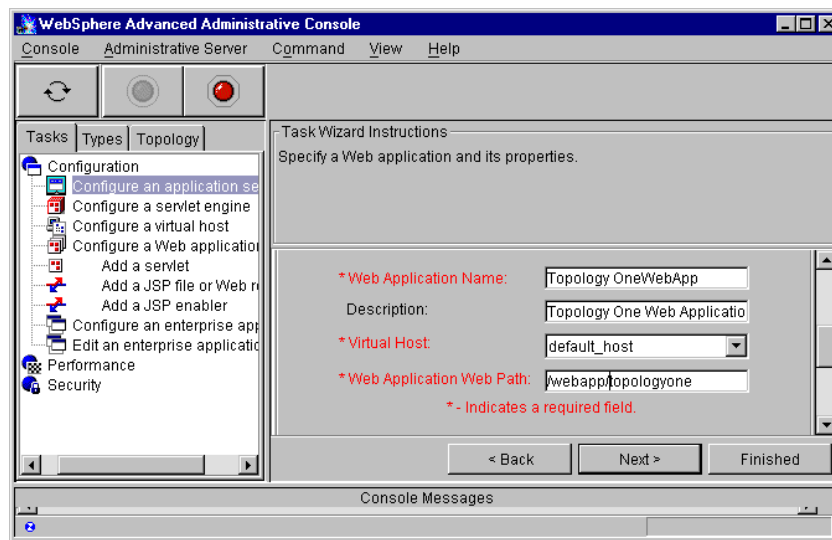


Figure 108. Configure an application server - Web application properties

Click **Next**.

9. On the next page, select **Enable File Servlet** and **Enable JSP 1.0**. Click **Finished**.

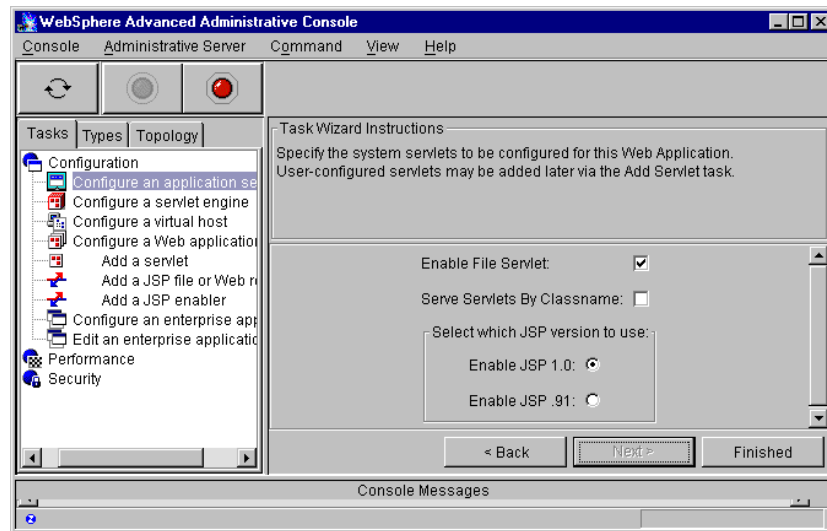


Figure 109. Configure an application server - system servlets properties

12.3.3 Set up the application file structure

Once the application server is defined, switch to the Topology tab. By expanding the topology structure on the left, find the Topology OneWebApp application and highlight it. Click the **Advanced** tab to view the path structure to be used for the application.

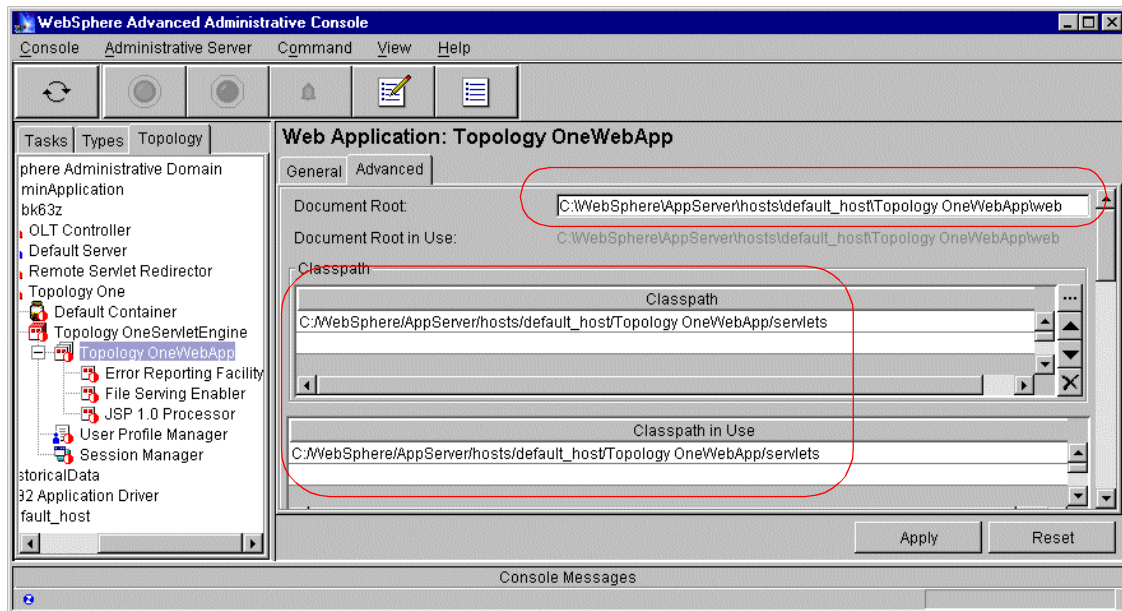


Figure 110. Web application topologyone: advanced properties

Now you need to put the application files where the application server will find them.

1. Create the document root directory for the newly created application server. You will find the directory structure in the first line of Figure 110.
`c:\WebSphere\AppServer\hosts\default_host\Topology OneWebApp\web`
2. Create the directory for Java servlets. You will find the directory structure in the Classpath field in Figure 110.
`c:\WebSphere\AppServer\hosts\default_host\Topology OneWebApp\servlets`

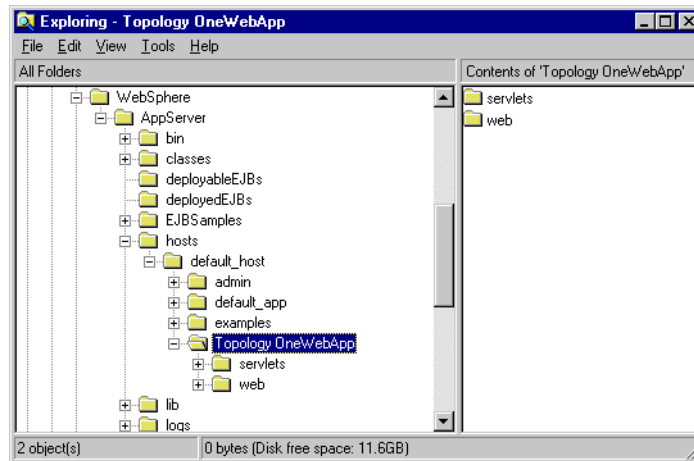


Figure 111. Directory structure

3. Copy the Java files from the PDK to the application server.

The files will be located on the PDK machine in the subdirectories under:

```
C:\WebSphere\AppServer\hosts\default_host\topologyone\servlets
  -com\ibm\hursley\asg\ws\skeleton\topologyone\sectiona
  -com\ibm\hursley\asg\ws\skeleton\command
  -com\ibm\hursley\asg\ws\skeleton\view
```

Copy the com subdirectory structure to the new application server machine in:

```
C:\WebSphere\AppServer\hosts\default_host\Topology OneWebApp\servlets\com
```

4. Copy the JSP files from the PDK to the application server. The files will be located on the PDK machine under:

```
C:\WebSphere\AppServer\hosts\default_host\topologyone\web\sectionA
```

Copy the sectionA subdirectory to the new application server machine under:

```
C:\WebSphere\AppServer\hosts\default_host\Topology OneWebApp\web\sectionA
```

12.3.4 Define the servlet

The next step is to define a servlet.

1. Under the Tasks tab select **Add a servlet** under **Configure a Web application**. Click the **Start Task** button (green circle on the smart-icon bar).

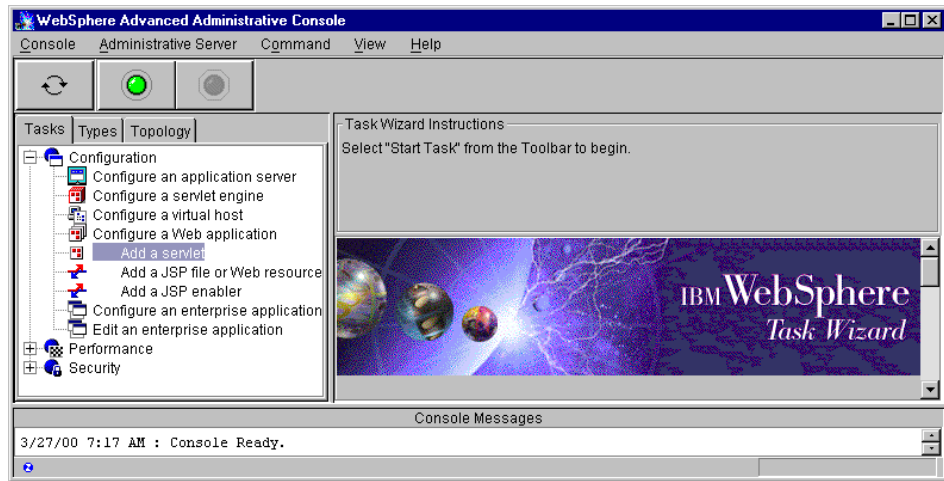


Figure 112. Defining a servlet

2. On the first page, select **No**.

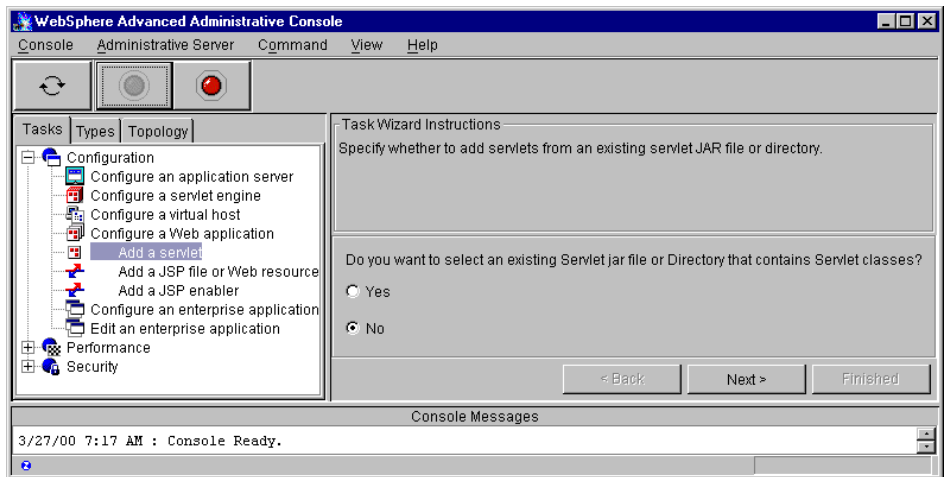


Figure 113. Add a servlet - creation method

Click **Next**.

3. On the next page, select Topology OneWebApp as the Web application.

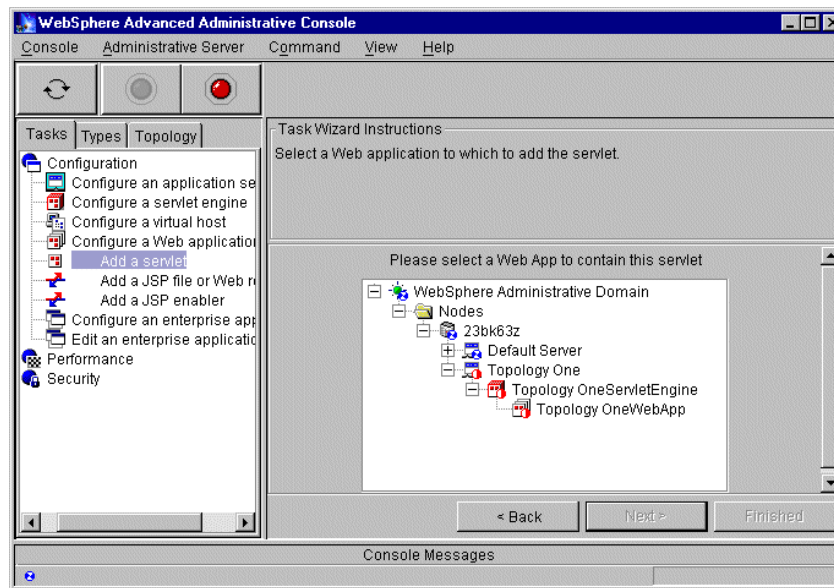


Figure 114. Add a servlet - select the Web application

Click **Next**.

4. On the next page, select **Create User-Defined Servlet**.

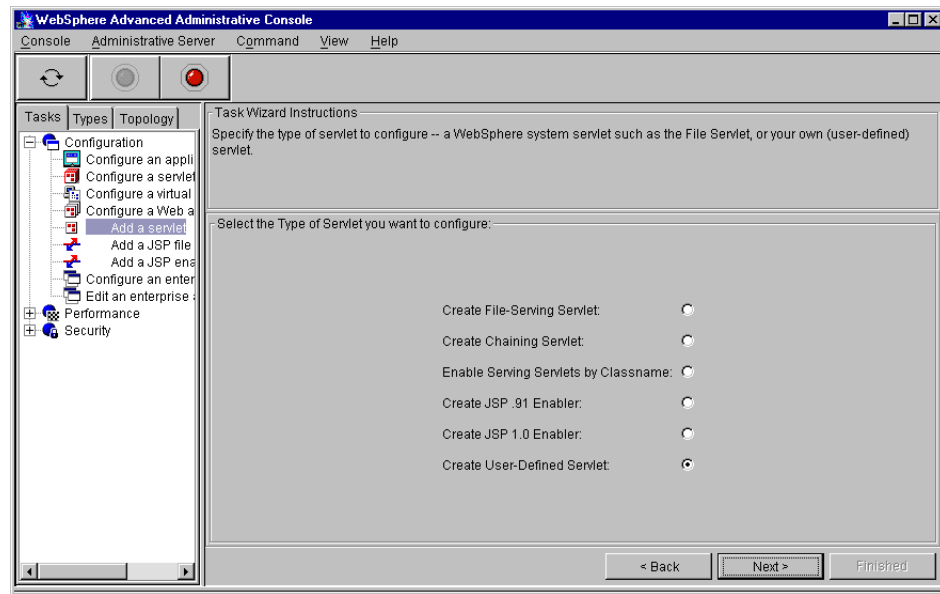


Figure 115. Add a servlet - Type

Click **Next**.

5. On the next page you define the servlet.

- Specify a servlet name. We used "Retrieve Historical Data Servlet" as the name.
- Select Topology OneWebApp as Web Application (defined in step 8 on page 259).
- Specify the servlet class name. This is the fully qualified class name without the file extension.

```
com.ibm.hursley.asg.ws.skeleton.topologyone.sectiona.RetrieveHistorical  
DataServlet
```

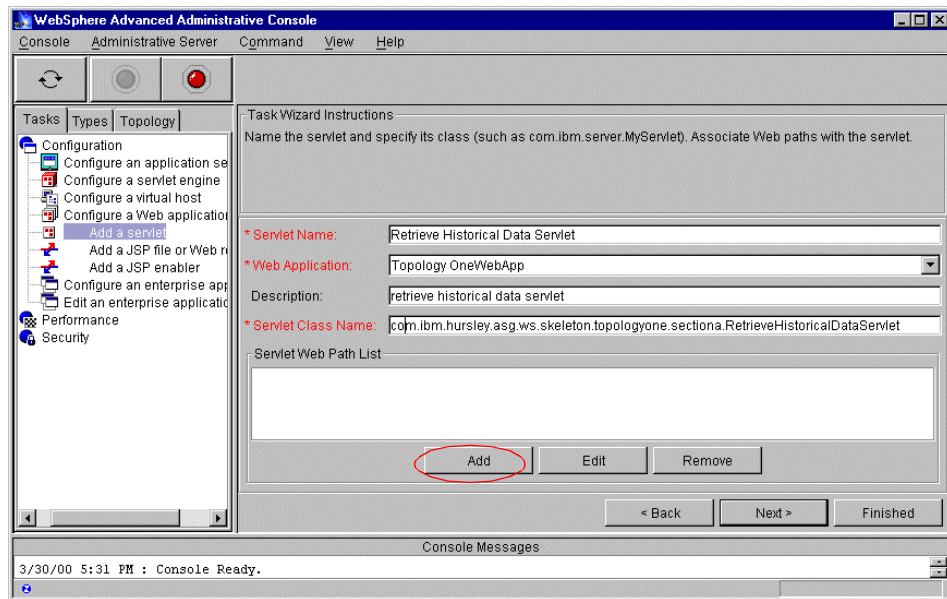


Figure 116. Add a servlet - Properties

Click **Add** to add an entry to the Servlet Web Path List.

6. In the following pop-up window, specify `histData` as the Web path of the servlet. It is important to specify this path just as is, since it must match what is used by `sectionaForm.html`.

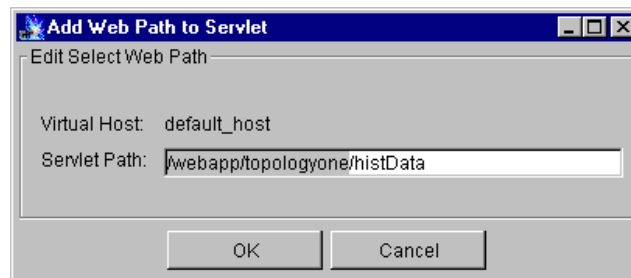


Figure 117. Add a servlet - add Web path to servlet

Click **OK**.

Click **Next**.

7. On the next page, take the defaults and click **Finished**.

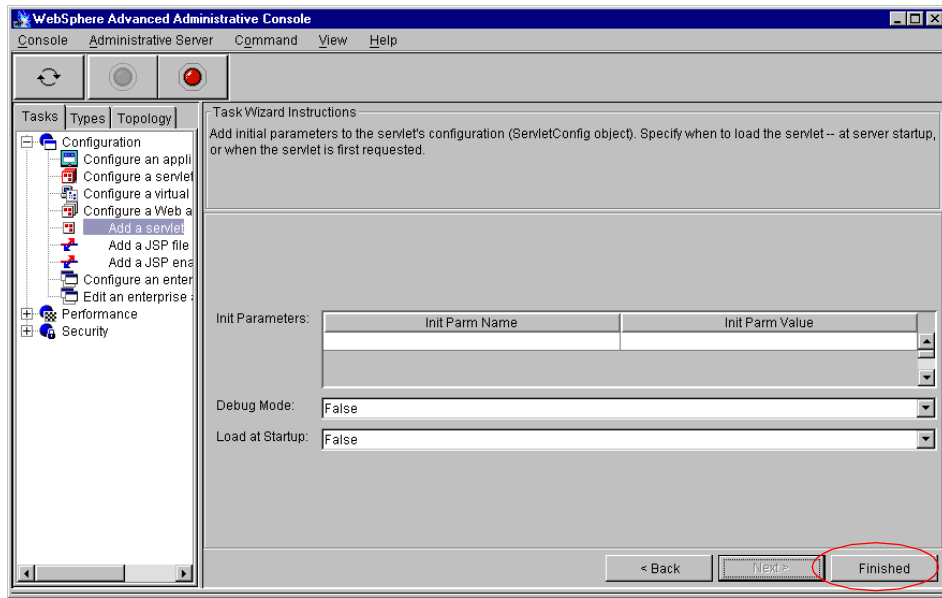


Figure 118. Add a servlet - more properties

8. Finally restart the Web application. Do this under the Topology tab. Highlight the Topology OneWebApp under the Topology OneServletEngine in the Topology One Web application server, right click, and select **Restart Web App**.

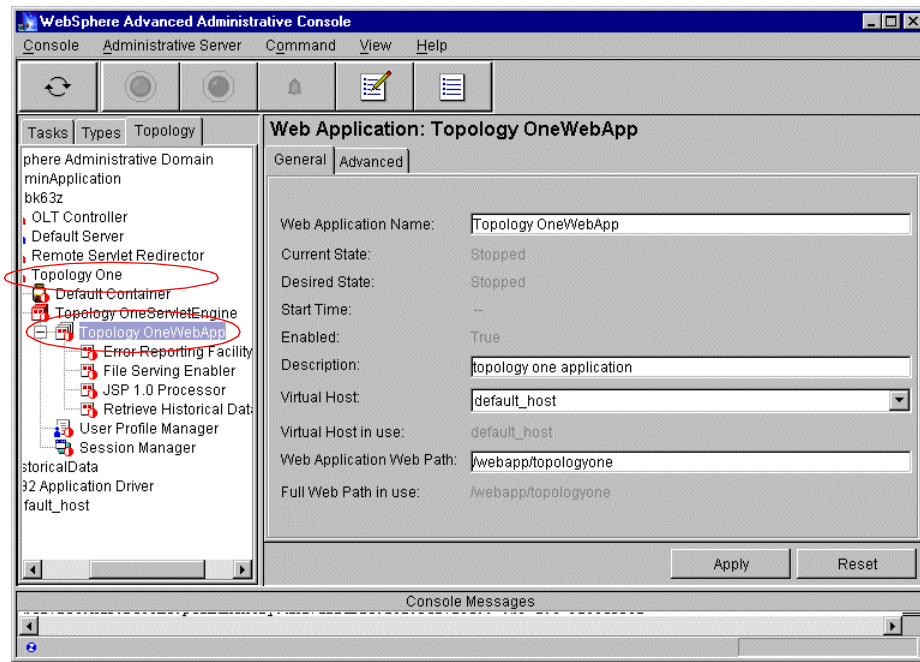


Figure 119. Restarting the Web Application

If the Topology One application server is not running, start it (highlight, right click, and select **Start**).

12.4 Separating the database from the Web application server

The next step in distributing the application functions was to put the application database on a third machine, Machine C.

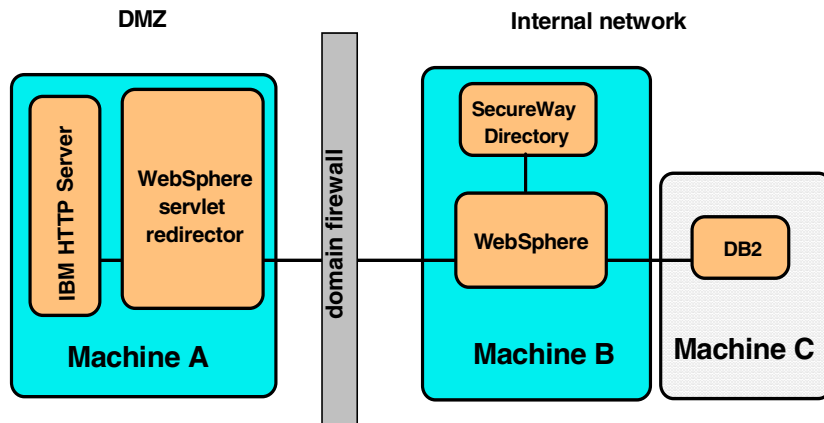


Figure 120. Separating the DB2 application database

To achieve this we did the following:

1. First we installed DB2 on Machine C. In this case we used Windows NT with DB2 5.2 + fix pack 11. To be consistent with the PDK, we created a user ID (USERID) with password PASSWORD which had administrative privileges and specified this as the DB2 administrative ID.
2. The following procedure was used to create the sample DB2 database:

- Copy the following directories from the PDK machine to a diskette:

```
c:\ASG\SDK\U2BTop\TestDrive\base\cmd\sectiona
```

```
c:\ASG\SDK\U2BTop\TestDrive\base\artifacts\sectiona
```

- Open a DB2 command window on Machine C:

```
Start->Programs->DB2 for Windows NT-> Command Window
```

Execute the following script from the solution kit:

```
c:\ASG\SDK\U2BTop\TestDrive\base\cmd\sectiona\crthistdata.cmd
```

This script executes the following DB2 commands:

```

SET DB2INSTANCE=DB2
DB2START
DB2 CREATE DATABASE HISTDATA
DB2 CONNECT TO HISTDATA
DB2 CREATE TABLE SKELETON.STAFF(STAFF_ID VARCHAR(8) not null,
ACCESS_LEVEL VARCHAR(2), JOB_DESCRIPTION VARCHAR(40), primary key
(STAFF_ID))

DB2 CREATE TABLE SKELETON.PLANETS(PLANET_ID INTEGER not null, NAME
VARCHAR(50), primary key (PLANET_ID))

DB2 CREATE TABLE SKELETON.WEATHER_READINGS(READING_ID INTEGER not null,
PLANET_ID INTEGER, TEMP VARCHAR(3), HUMIDITY VARCHAR(3), WINDSPEED
VARCHAR(3), LOCAL_TIME TIME, LOCAL_DATE DATE, ACCESS_LEVEL VARCHAR(2),
primary key (READING_ID))

DB2 CONNECT RESET

```

Figure 121. DB2 commands to create the HISTDATA database

Next, execute the script to import the data:

```
c:\ASG\SDK\U2BTop\TestDrive\base\cmd\sectiona\imphistdata.cmd
```

3. If it exists, drop the local database on the Web application server (Machine B). You can do this using the DB2 Control Center.

**Start->Programs->DB2 for Windows NT->Administration
Tools->Control Center**

Right click the HISTDATA database, and choose **Drop**.

4. Use the Client Configuration Assistant (CCA) on the Web application server to create a new database that points to the database on the database server:

**Start->Programs->DB2 for Windows NT->Client Configuration
Assistant**

The initial window for the CCA shows the existing DB2 databases. Click **Add** to begin defining the remote database you just created.

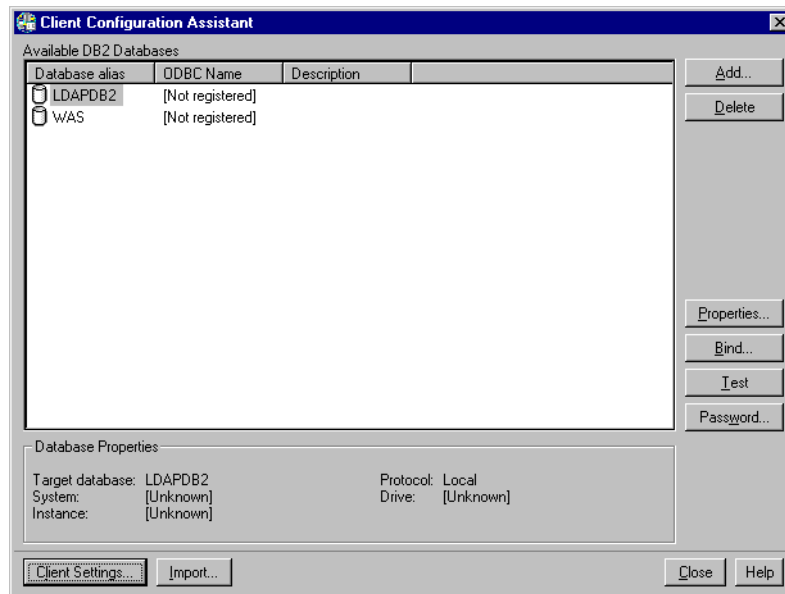


Figure 122. CCA initial window

In the next window, click **Manually configure a connection to a DB2 database**.

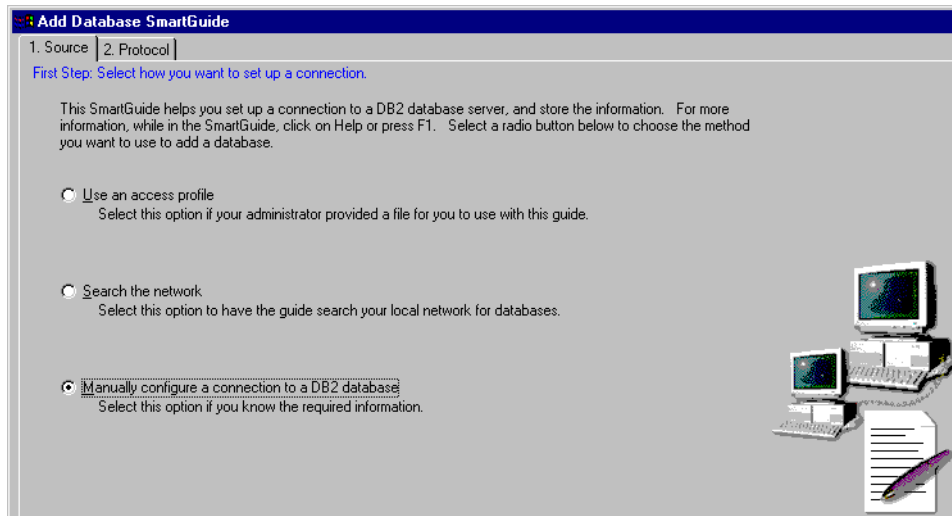


Figure 123. Adding a database connection

Choose **TCP/IP** (or the appropriate protocol) as the communications protocol. For TCP/IP, the next window will ask for:

- Hostname: We entered the IP address
- Port number: 50000

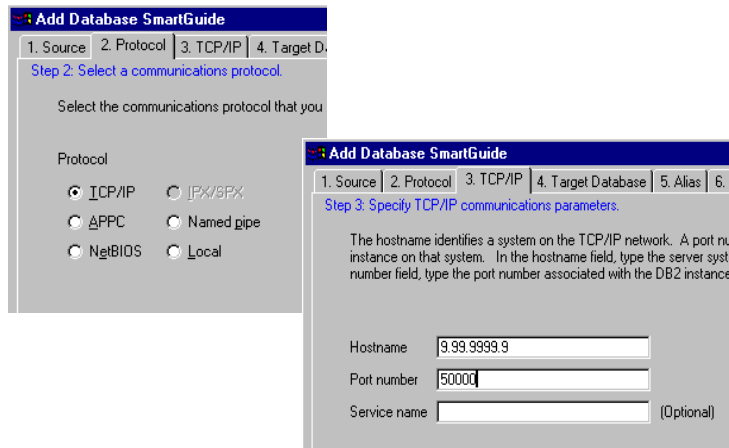


Figure 124. Defining the communications protocol

The next two windows define the database and its alias. We used the database name (HISTDATA) for both.

Note: If you have not already dropped the original database you will need to do this before executing this step.

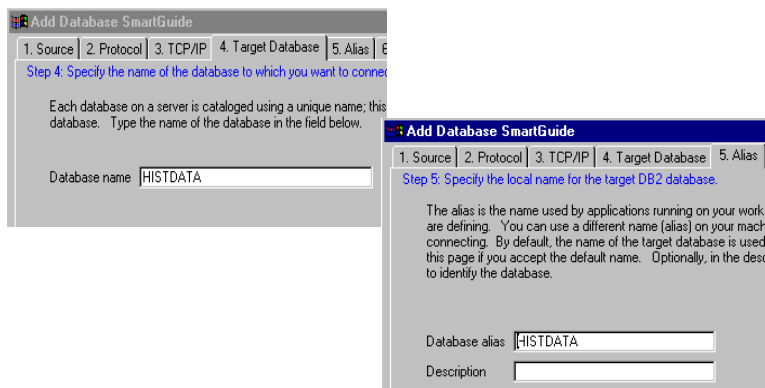


Figure 125. Defining the database and its alias

The last window gives you the option of registering this database as an ODBC source.

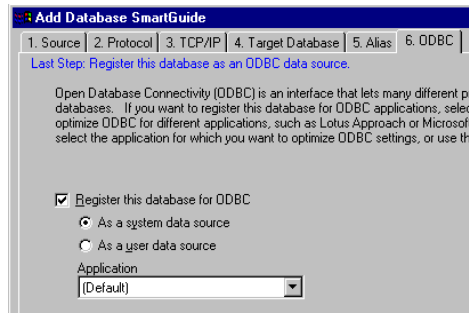


Figure 126. Register the database to ODBC

Once the database is defined, you are given the option to test the database connection. This is always a good idea since it ensures that the connection is defined properly. When you click the **Test Connection** button, you will be prompted for a user ID and password (USERID and PASSWORD in our example) with access to the database.

5. Restart the Web application server on Machine B (see step 8 on page 267).

12.5 Testing the application

At this point, you should have a working application. You can test it by opening a browser on the Web server machine and typing in the URL:

`http://localhost`

If you replaced the default files in the IBM HTTP server, as described in 12.2, “Separating the application server from the Web server” on page 250, you will see the initial window for the PDK (see Figure 84 on page 233). Follow the links for the Topology 1 application to verify that you can see the final inter-planetary data.

Chapter 13. Step 3: Securing the PDK application

The next step in our example is to secure our application. The example will use basic authentication (user ID and password) and the IBM SecureWay directory as the user registry.

Defining security for the new Topology One application involves:

- Configuring the IBM Secureway Directory. The instructions for this can be found in Chapter 17, “SecureWay Directory Configuration” on page 325.
- Enabling global security in WebSphere and configuring it to use the SecureWay Directory.
- Configuring application security in WebSphere.

13.1 Enabling application security in WebSphere

WebSphere security is enabled and configured using the Security task. Using the WebSphere administrative console, switch to the Tasks tab and expand the security item.

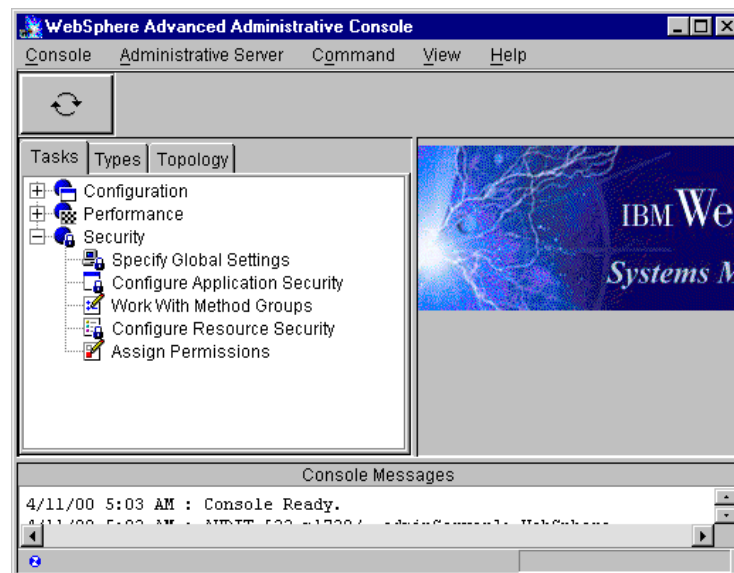


Figure 127. Security configuration task

There are five tasks listed under security:

- Specify Global Settings

- Configure Application Security
- Work with Method Groups
- Configure Resource Security
- Assign Permissions

Each of these tasks is designed to be performed in the order listed to enable WebSphere security.

13.2 Enabling WebSphere global security

The first step is to define the global security settings for WebSphere. At the completion of this step, the administrative server will be protected from unauthorized access.

Note

Once you have done this and restarted the administration server, you must have a working security registry, either LDAP or basic operating system, in order to bring the administrative console back up. Do not perform this step until this is done!

In this example, an IBM SecureWay Directory LDAP server is going to serve as the user registry.

1. Make sure the LDAP server is running.
2. Bring up the WebSphere administrative console. Under the Tasks tab, highlight **Specify Global Settings** under the Security item. Then click the green light to start the security task.
3. In the following window, under the General tab, check the **Enable Security** checkbox.

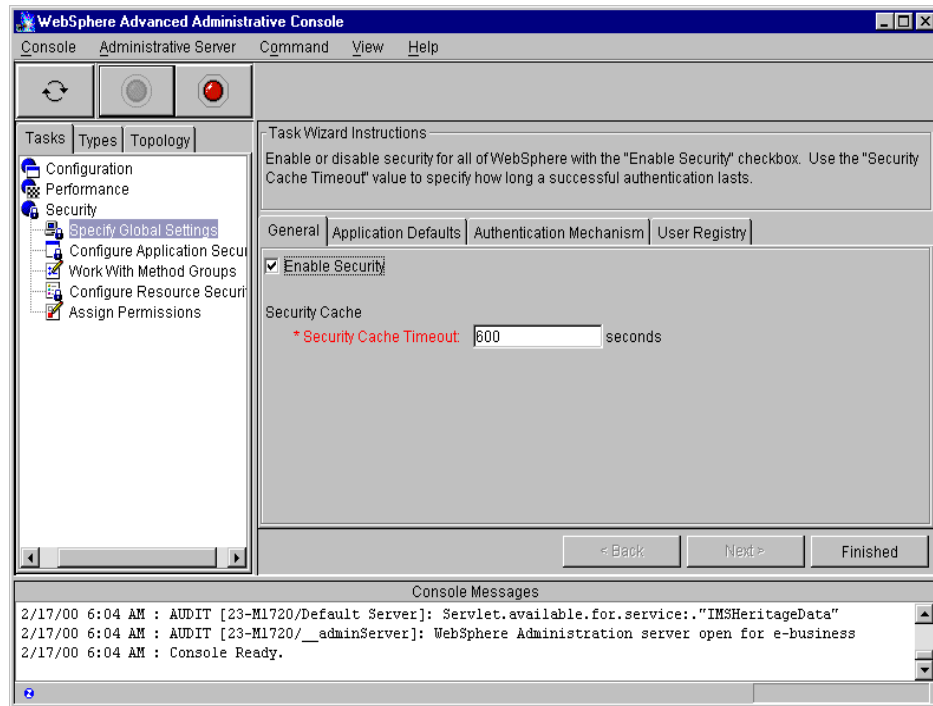


Figure 128. WebSphere global security settings

4. In the Application Defaults tab, a Realm Name is automatically entered. The security realm is the domain in which a security system operates. You can name the realm anything you like. All applications will need to belong to this security realm. Under Challenge Type, select the **Basic** radio button.

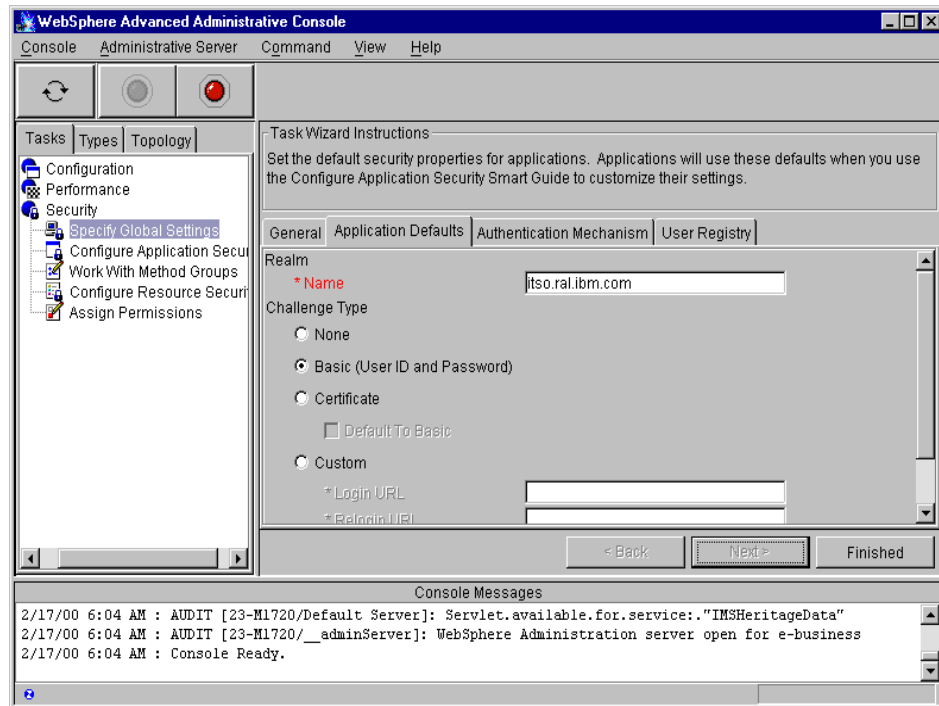


Figure 129. WebSphere security: application defaults

5. In the Authentication Mechanism tab specify **Lightweight Third Party Authentication**. (For Lightweight Third Party Authentication (LTPA) testing use the default token expiration of 30 minutes.)

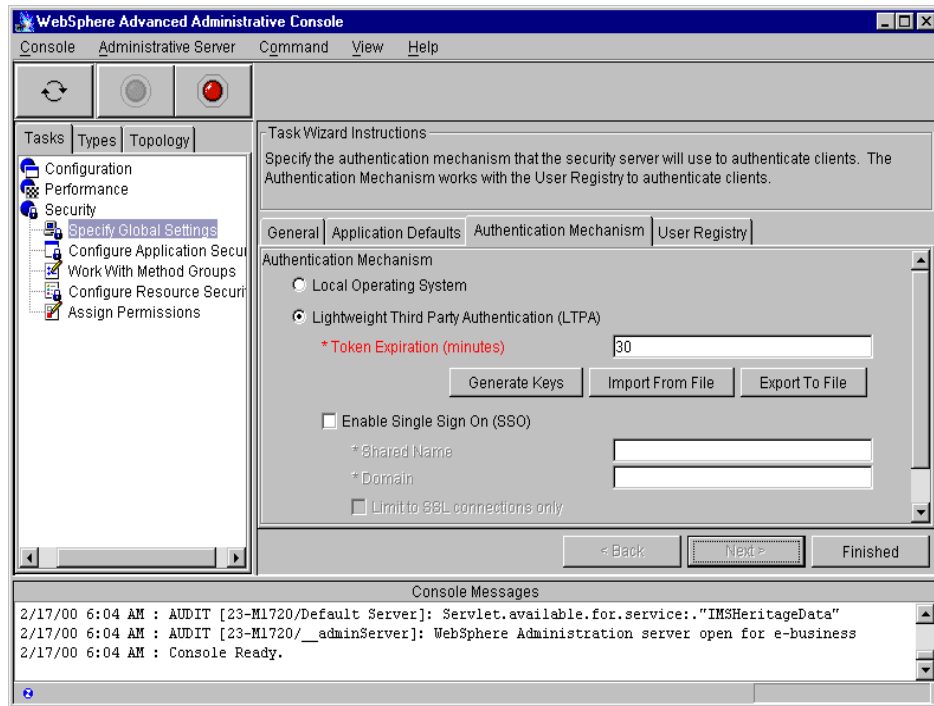


Figure 130. WebSphere security: authentication mechanism

6. Under the User Registry tab in the Global Settings for Security:

- Enter the top level administrator distinguished name (DN) for the security server ID (`cn=USERID`) and the corresponding password (`PASSWORD`).
- Choose SecureWay for the directory type.
- Enter the SecureWay Directory host name or IP address.
- Port 389 should be selected. This is the default for the IBM SecureWay Directory. If you changed this, you must reflect that here.
- Enter the base distinguished name that you want to use. The base DN identifies the point in the directory that you want to start searching. It could be the root of a tree in the directory (for example, `o=ibm, c=uk`), or you could narrow the search down to a particular organizational unit, as we have done here.
- Enter `cn=USERID` for the bind DN.
- Enter the password for the bind DN in the bind password field.
- Click **Finished**.

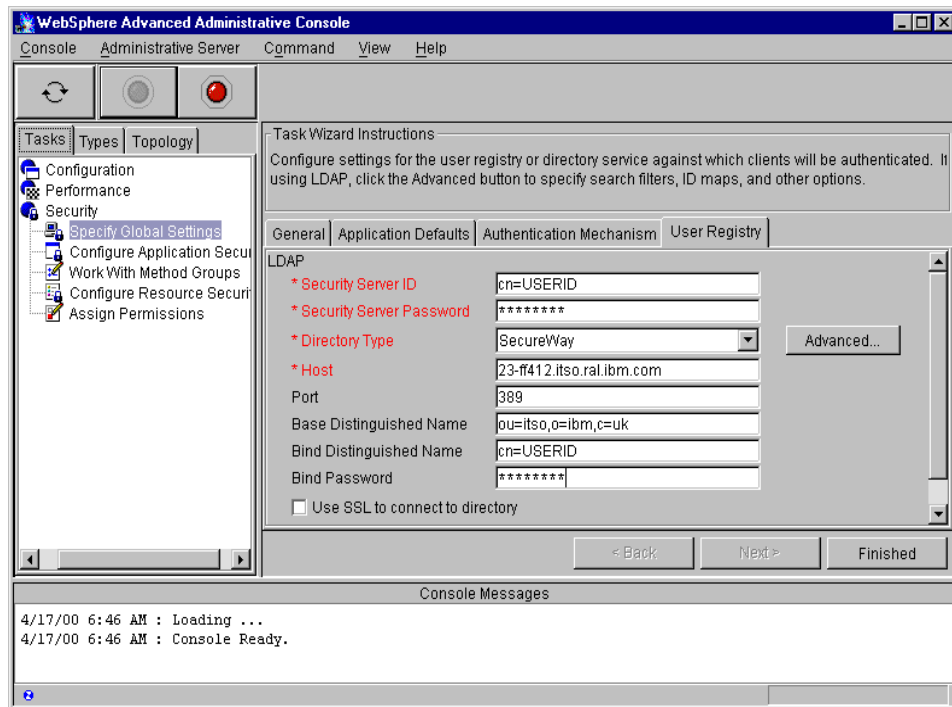


Figure 131. WebSphere security: user registry

You have now enabled security and specified the user registry to use. The administrative server is currently the only resource protected. The next step is to protect the application.

Activating global security requires stopping the administrative server and restarting it.

13.2.1 Protecting the application

In order to be protected by WebSphere security, a Web application has to be a part of an enterprise application. The first task will be to create an enterprise application that includes our example application.

1. Bring up the WebSphere administrative console. Under the Tasks tab, double click **Configuration** to expand it. Click **Configure an enterprise application**. Click the green light to start the task.
2. Give your application a name. Click **Next**.

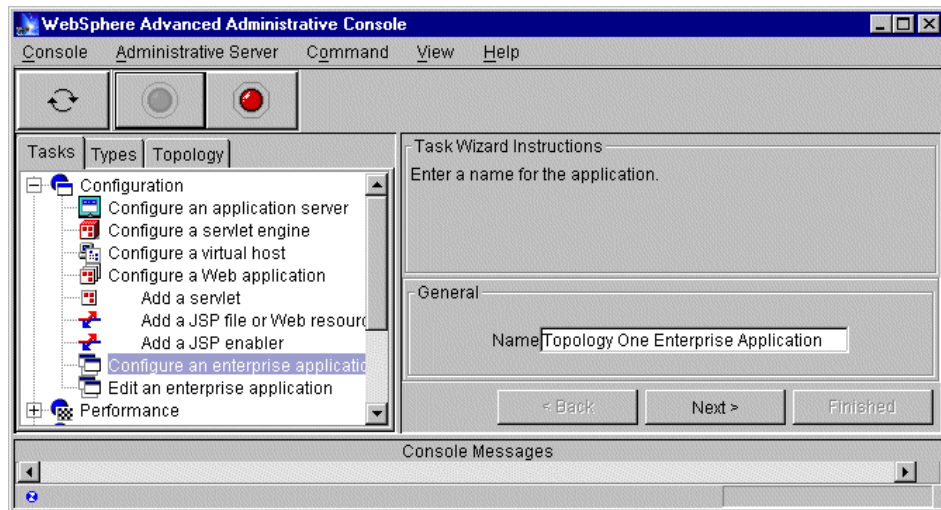


Figure 132. Configuring an enterprise application

3. On the next window add the resource `Topology OneWebApp`. Click **Add**, then **Finished**.

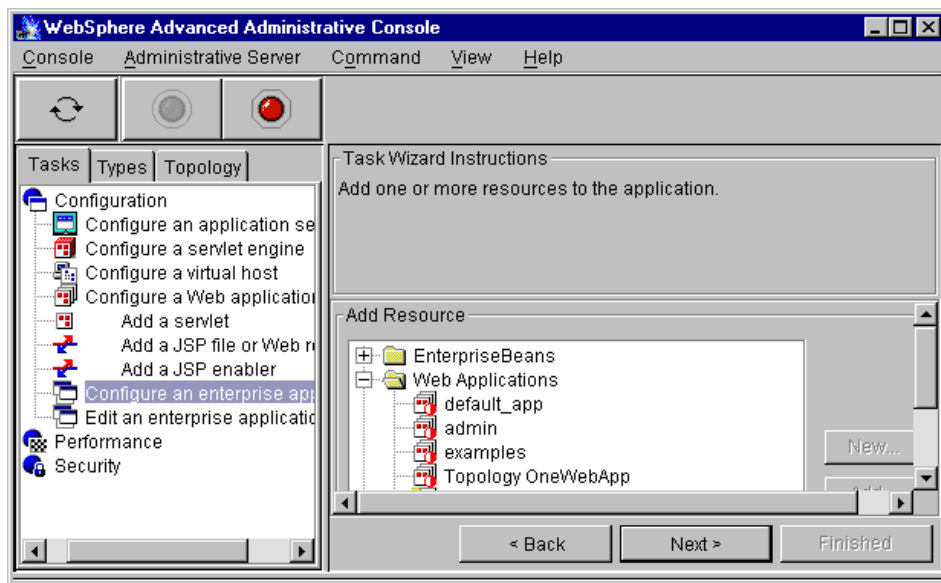


Figure 133. Adding resources to an enterprise application

4. Click **Finished**. If you see any `DuplicationRelationInstanceException` warnings, you can safely ignore them.

The next step in defining security is to configure application security. This is the next item in the GUI under security.

1. Highlight **Configure Application Security** and click the green start button.
2. Choose the enterprise application you just created and click **Next**.
3. You do not need to change anything on the next window but it is important that you complete this task and click **Finished**.

If you would like to define a new method group, you can do so with the next task, Work With Method Groups. For our example, this is not necessary.

You are now ready to configure the resource security. Without doing so, every application running on the application server will be accessible by anyone.

1. In the Tasks tab, click on the next security task, **Configure Resource Security**. Click the green light to start that task.
2. On the right side of the window, expand `Virtual Hosts` and then `default_host`.
3. From the list of resources, select the resource you want to configure. In our example we click `/webapp/topologyone/histData`. This is the Web path for the servlet that retrieves the data. Click **Next**.

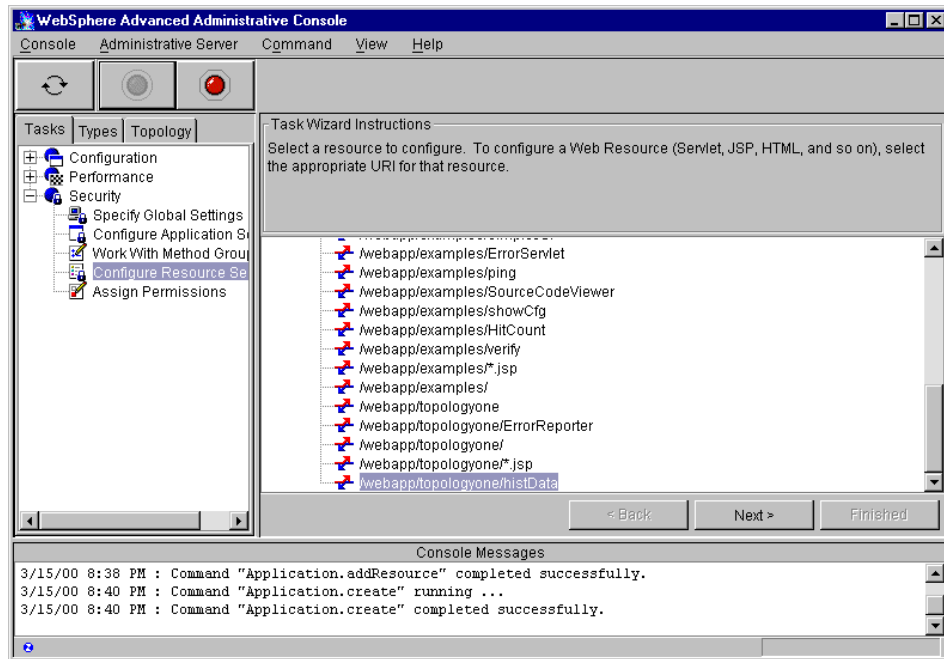


Figure 134. Selecting a resources to be configured

4. When asked if you want to use the default method groups, click **Yes**.
5. When the next window comes up, click **Finished**.

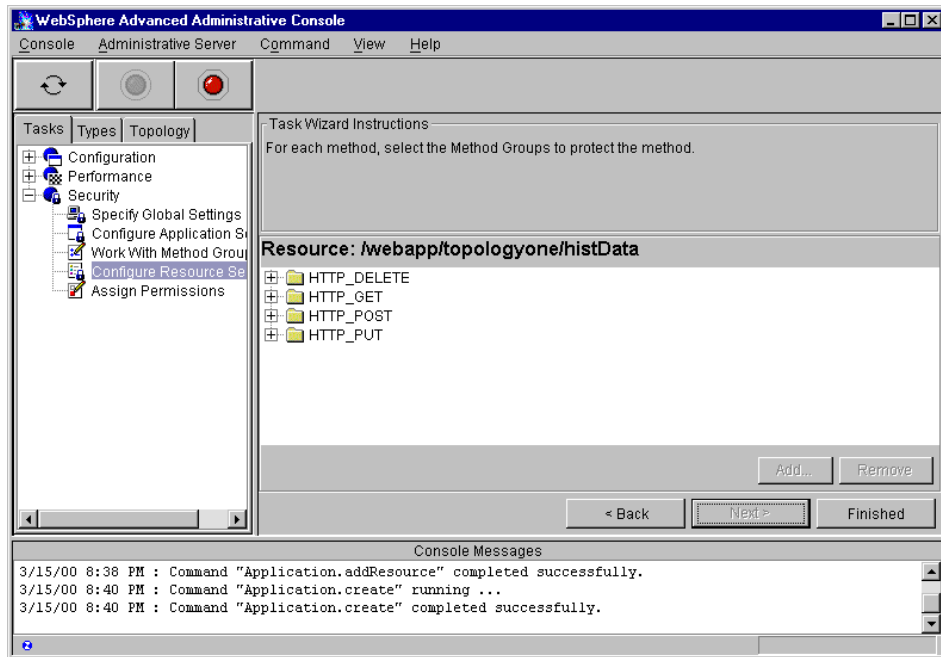


Figure 135. Adding Default Method Groups to resources

You can repeat the steps described above as many times as needed. Each time configuring a different resource.

Now that the resources have been configured for security, you will assign permissions to the enterprise application you created before.

6. In the Tasks tab under Security, highlight **Assign Permissions**.
7. In the next window, select all entries that begin with *Topology One*. You can do this by clicking the first entry and then, while pressing the CTRL key, select the other entries one at a time. Then click **Add**.

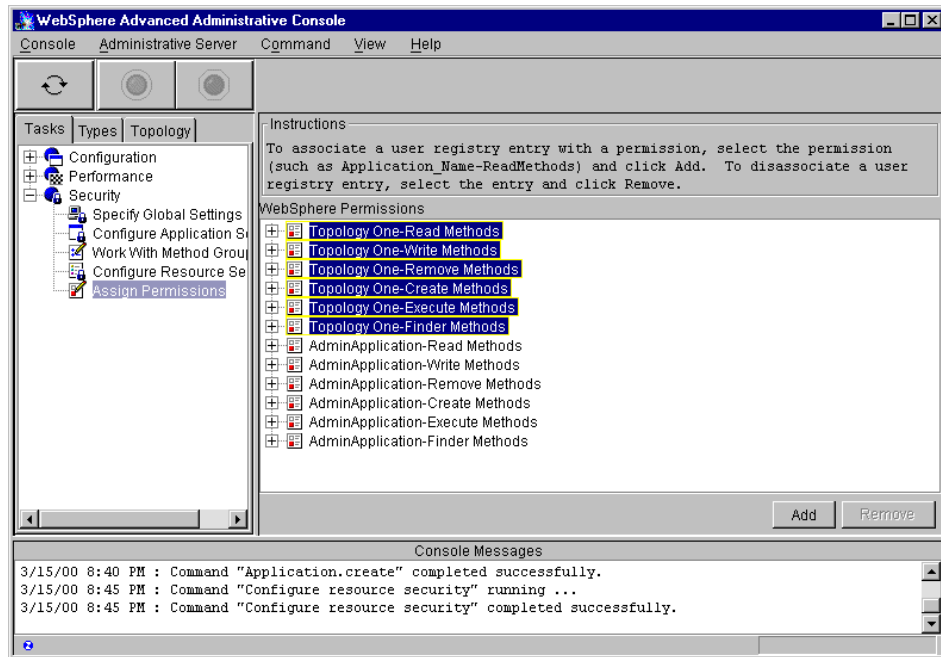


Figure 136. Selecting an application to be secured

8. Select **All Authenticated Users** and click on **OK**.

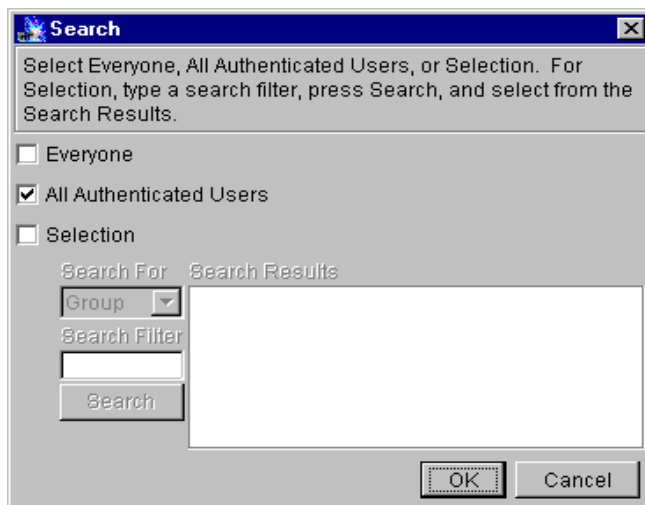


Figure 137. Selecting users who can access an application

The Topology One enterprise application is now secure from unauthorized access.

You will need to stop and restart the administrative server task.

13.3 Hints and tips

If you turn on security using LDAP, the LDAP server must be available and the definitions must be correct. If you have made any mistakes, you will not be able to start the WebSphere administrative console. The IBM WS AdminServer service will start but the console GUI interface will fail to start. Unfortunately, you use the administrative console to correct the security definitions (if that is the problem). This is known as a catch-22.

Tip: If you use the task bar to start the administrative console the messages appear in a window that closes quickly. Open a command prompt window and enter `adminclient` to start the console manually. The advantage of this is that the error messages will not go away.

Tip: If you have a problem starting the administrative console you can turn the security feature off manually. Open the DB2 command line processor and enter the following commands:

- `connect to was`
- `update ejssadmin.securitycfg_table set securityenabled = 0`
- `commit`

Stop and restart IBM WS AdminServer service.

Warning

Turning off security this way may not always work. Try this only if everything else fails. You may have to drop all applications from the database or even to reboot the machine.

Chapter 14. Step 4: Cloning an application server

This chapter describes how to set up cloning on the application servers. Implementation of cloning for this example requires:

1. Creating a model of the Topology One application server.
2. Creating clones of the application server model on both the AIX and the Windows NT servers.

14.1 Topology and product mapping

Figure 138 shows the environment for our cloning example.

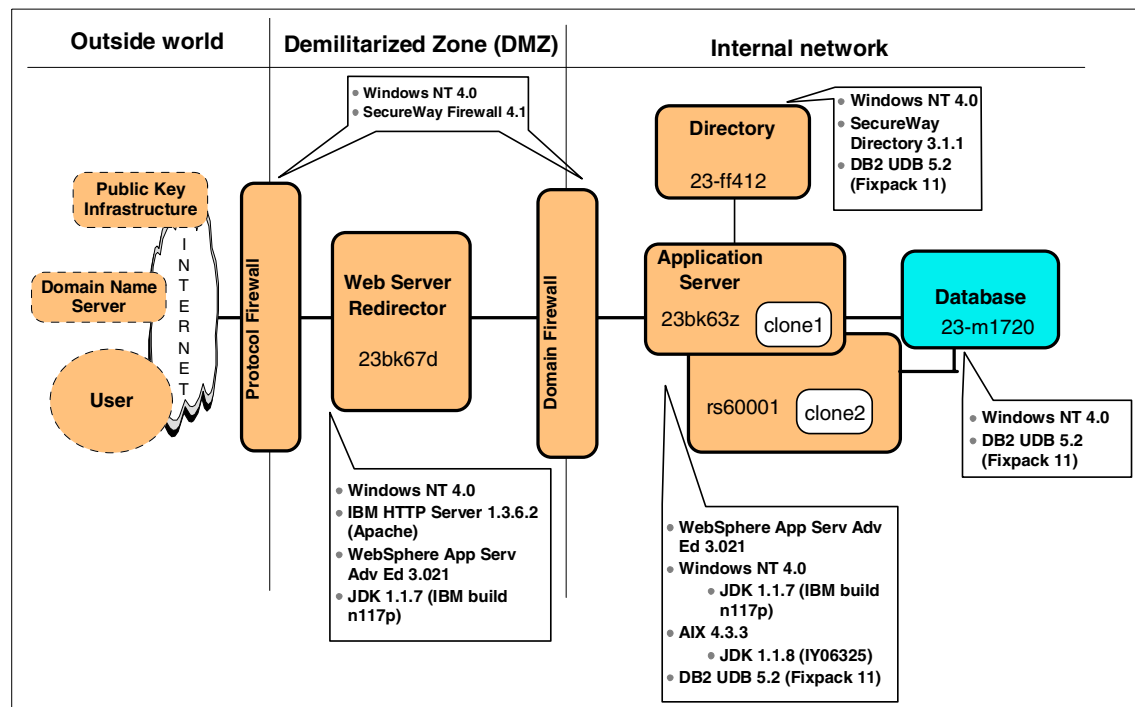


Figure 138. Cloning example

The instructions given here assume that the base products on each node and their prerequisites have been installed. The initial setup is as follows:

- RS60001 is an AIX 4.3.3 system with WebSphere Application Server V3.021 and its prerequisites installed.

- 23bk63z is a Windows NT Server V4 system with WebSphere Application Server V3.021 and its prerequisites installed. No application servers have been defined.
- Initially, both rs60001 and 23bk63z are in the same WebSphere administrative domain, meaning they share the same database.

Note

This assumes that the firewalls and servlet redirector are in place.

In this example, we will use the Topology One application server created in 12.3, “Setting up the application server on Machine B” on page 251. As an alternative, if you simply want to try cloning, you can clone the default server and use the showCfg sample servlet that comes with the WebSphere examples. This servlet dumps the configuration of the servlet engine it is running in, including the server name and clone index, making it easy to verify which clone is actually being used.

14.2 Preparing the WebSphere administrative domain

Notice that we have introduced a WebSphere Application Server machine (rs60001) into the WebSphere administrative domain. After a node has been added to the domain, the `default_host` alias list must be updated to contain the names, both in short and long form, as well the IP address of all the nodes in the domain.

To do this, select **default_host** on the Topology tab and enter the values in the host alias list and apply the changes. In our case, we have two nodes in the domain (RS60001 and 23bk63z). In addition we have a servlet redirector (23bk67d). Our list would include the following:

- localhost
- 127.0.0.1
- 23bk63z
- 23bk63z.itso.ral.ibm.com
- 9.24.104.110 (IP address of 23bk63z)
- rs60001
- rs60001.itso.ral.ibm.com
- 9.24.104.105 (IP address of rs60001)
- 23bk67d
- 23bk67d.itso.ral.ibm.com
- 9.24.104.62 (IP address of the servlet redirector)

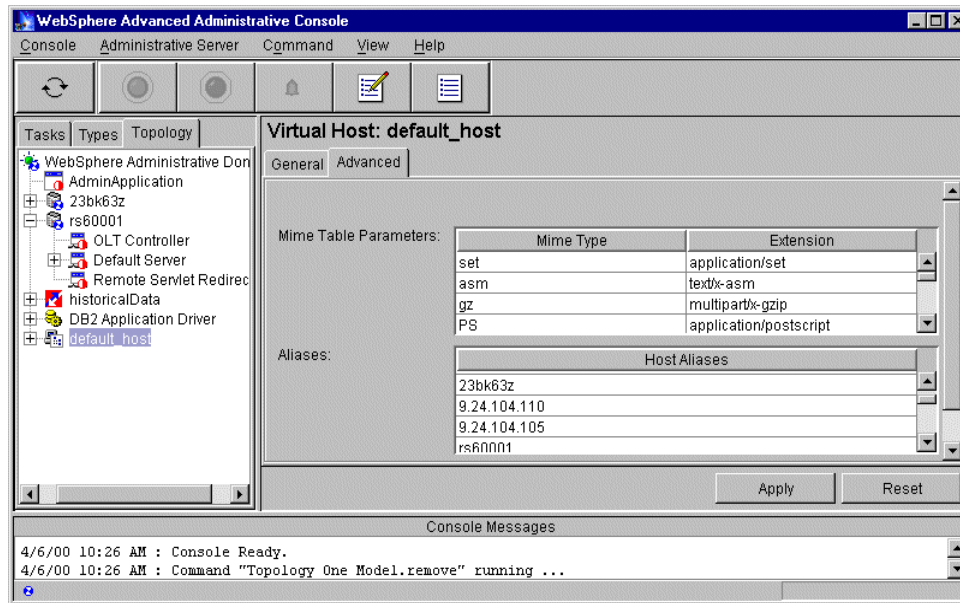


Figure 139. Setting Host Aliases

Changing the host alias list requires a restart of the application servers to take effect.

14.3 Creating a model

The first step in cloning is to create a model of the application server. This model serves as a template that can be later cloned. All changes made to this model are automatically propagated to the clones.

1. If it has been started, stop the Topology One application server. This can be done under the Topology tab by highlighting the **Topology One** application server, right clicking, and selecting **Stop**.
2. Highlight the Topology One application server. Right click and select **Create->Model**.

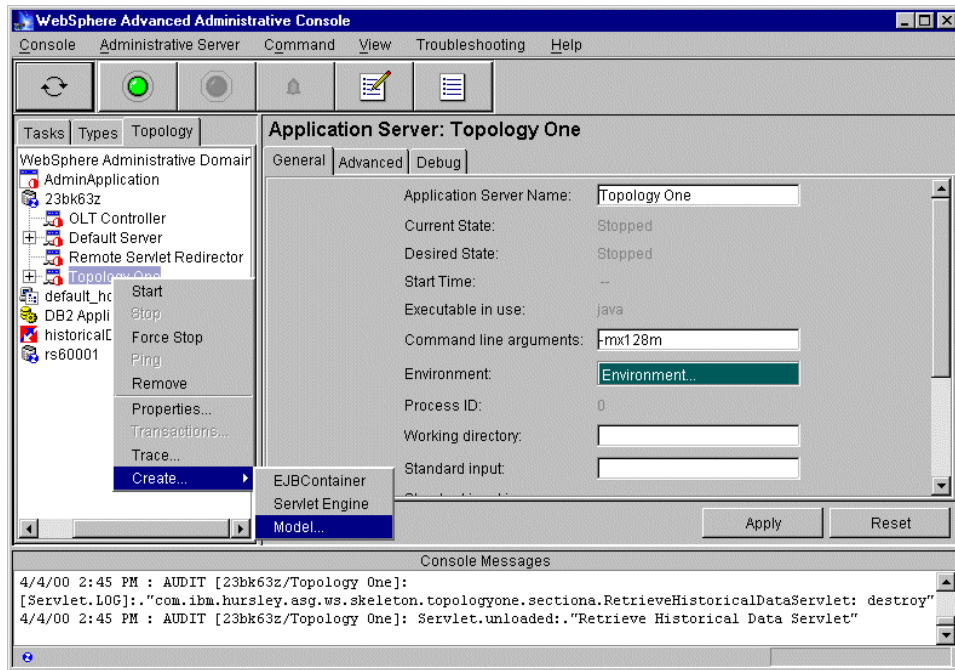


Figure 140. Creating a model of the Default Server

3. In the General tab of the Clone Properties window:

- Select **Make Topology One a Clone**. This makes the original application server a clone of the new model. All updates to the model will update this clone.
- Select **Recursively Model all instances under Topology One**. This causes all objects in the application server to be cloned as well.

Note

In all of the property windows, make sure all directory paths are valid for both the AIX node as well as the Windows NT node. Normally, you would use paths on a shared file system. To make the paths valid for both Windows NT and AIX we did the following:

- Removed the drive designation (C:) from the directory path and changed all back slashes (\) to forward slashes (/)
- Created a link on AIX from /WebSphere to /usr/WebSphere.

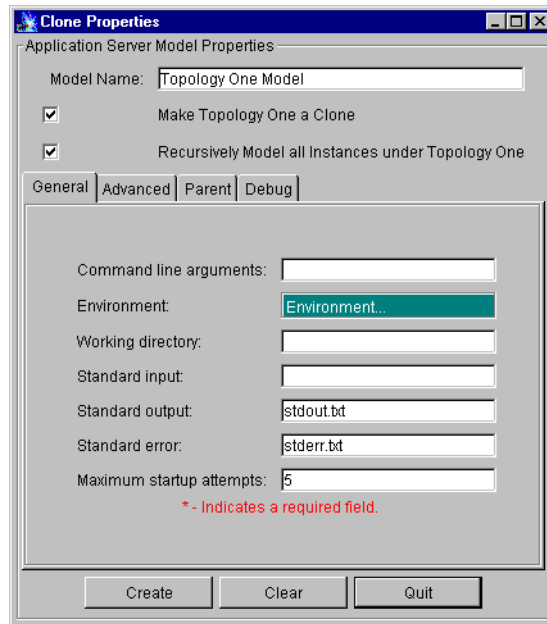


Figure 141. Clone properties: general properties

4. In the Advanced tab, select any Workload management selection policy you want. We chose to use a round-robin workload scheme.

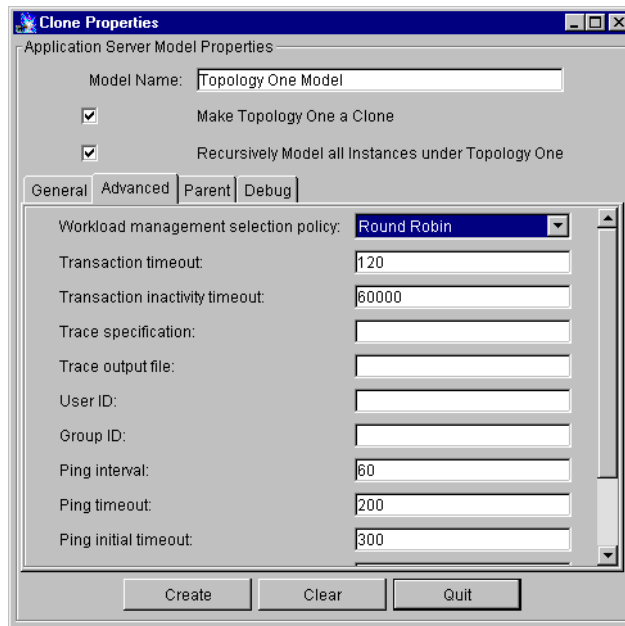


Figure 142. Clone properties: advanced

5. Switch to the Parent tab and highlight **Models**.

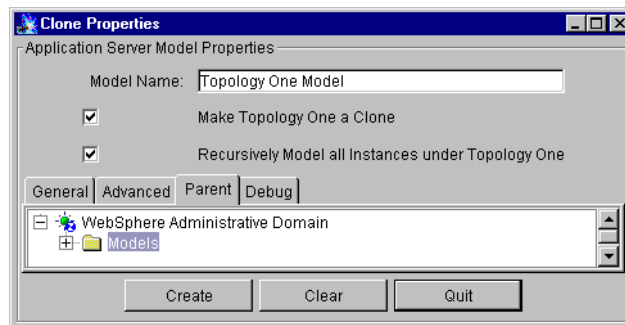


Figure 143. Filling out clone properties

6. After you have filled out the properties, click **Create** to create the model.
7. In the model, verify that the workload management selection policy is indeed set to the value (for example, Round Robin) you selected when creating the model. There is a known defect in WebSphere 3.02 that the workload management selection is not set during the model create process. You can check this and change it by highlighting the model

(Topology One Model) and clicking the **Advanced** tab in the window. This will be fixed in a future release.

8. Make sure that each Web application's document root and class path is correct. These values should be valid file and directory names. In this case we had to change the document root and class path to a format that both AIX and Windows NT could use.

The document root was changed from:

`C:\WebSphere\AppServer\hosts\default_host\Topology OneWebApp\web`

to:

`/WebSphere/AppServer/hosts/default_host/Topology OneWebApp/web`

The class path was changed from:

`C:/WebSphere/AppServer/hosts/default_host/Topology OneWebApp/servlets`

to:

`/WebSphere/AppServer/hosts/default_host/Topology OneWebApp/servlets`

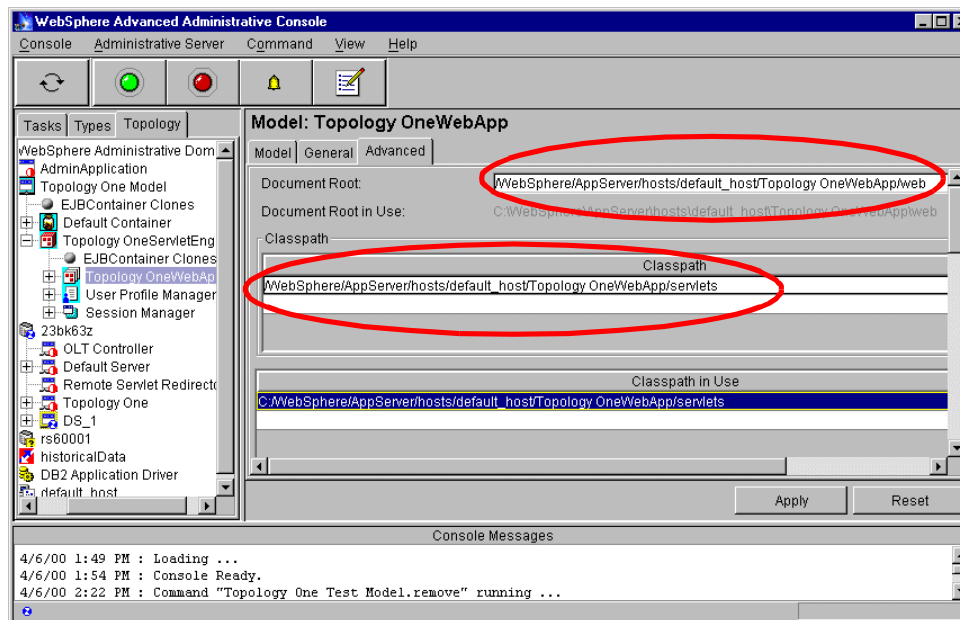


Figure 144. Verifying important Web Application settings

14.4 Creating the first clone

You are now ready to create the first clone from the model. The first clone will be created on the same node as the model (23bk63z). The clone will retain the application security established earlier.

1. Under the Topology tab, highlight **Topology One Model**, right click, and select **Create->Clone**.

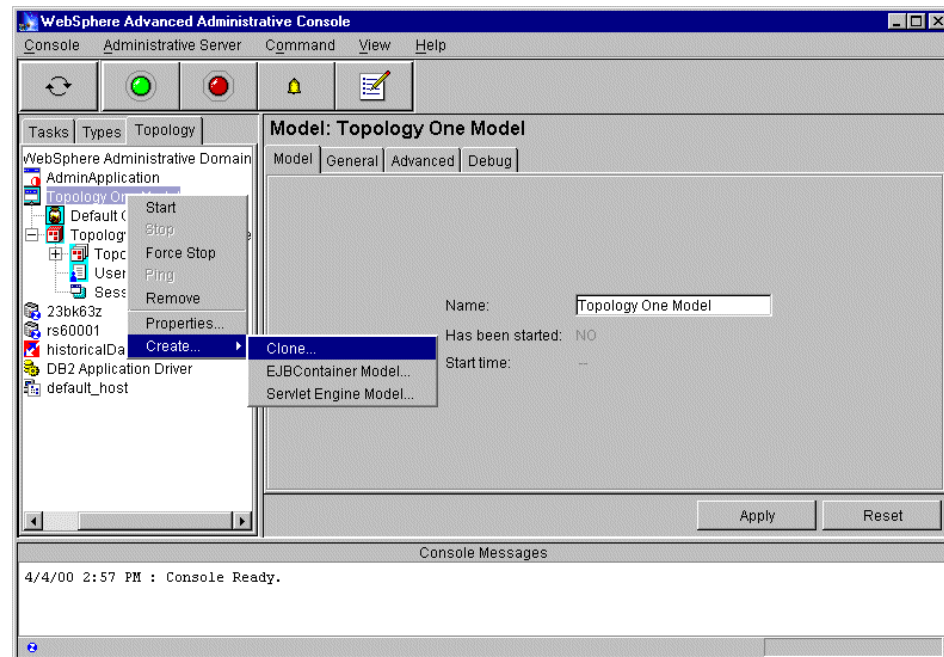


Figure 145. Cloning a model

2. Give the clone a name, for example, DS 1, and select the node for the clone to run on.

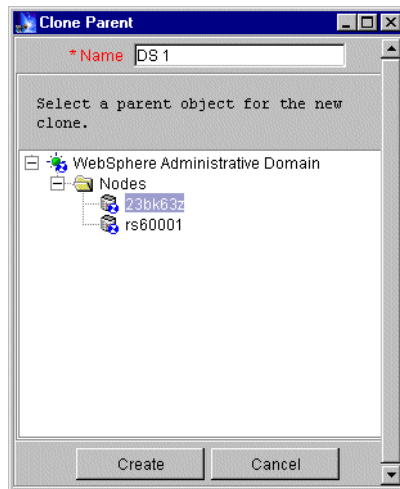


Figure 146. Creating a clone

Click **Create** to create the clone.

3. Under the Topology tab, highlight the **Topology One Model** and click the green button on top of the administrative console to start it.
4. Verify that the application is working by going to a Web browser and entering the application URL. In our example, the initial window for the Web server links to the application. See 10.2, “PDK section A” on page 233 for a description of the application.

`http://localhost`

For the next steps, stop the model (select it and click the red button on top of the administrative console).

14.5 Creating the next clone

The next step was to create a clone on the AIX system. The AIX system has been installed with WebSphere and its prerequisites. It is in the same WebSphere administrative domain as the Windows NT system.

1. Follow the instructions in 14.4, “Creating the first clone” on page 294 to create a clone of the Topology One Model called DS 2 on rs60001.
2. Next, copy the application files to the AIX system. We created the necessary directory structure and copied the application files into it. The file structure for the Topology One application is explained in 12.3.3, “Set up the application file structure” on page 260. For the AIX machine this

meant creating the following directory structures and copying the application files to them:

```
/WebSphere/AppServer/hosts/default_host/Topology OneWebApp/web  
/WebSphere/AppServer/hosts/default_host/Topology OneWebApp/servlets
```

Note: /WebSphere is linked to /usr/WebSphere.

3. Make a DB2 alias on rs60001 to point to the application database.

On the AIX machine, we used the following commands to do this:

```
su - db2inst1  
DB2  
catalog tcpip node histdb remote 23-m1720 server 50000  
catalog db HISTDATA as HISTDATA at node histdb  
connect to HISTDATA user USERID using PASSWORD
```

Figure 147. Creating a DB2 alias on AIX

Note

Traffic from the Web server will be alternated among clones on the first machine (Windows NT). In order for clones on the AIX machine to be used, a servlet redirector must be set up.

Chapter 15. Setting up a standalone servlet redirector

This chapter describes how to set up the servlet redirector used in the environment described in Chapter 10, “The Pattern Development Kit and an example topology” on page 231. The redirector used in the example is a standalone (“thin”) redirector. Redirectors are described in 4.1.1, “Implementing a redirector” on page 43.

Note

If you are running the servlet redirector, it is recommended that you be at WebSphere 3.021 level.

Figure 148 shows the environment for this example.

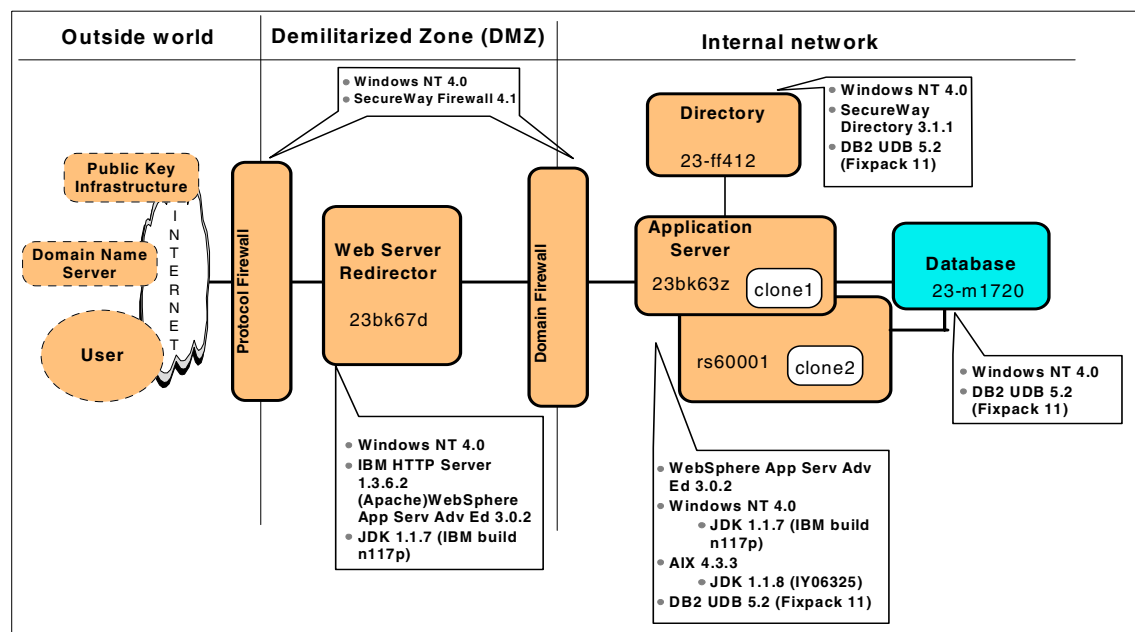


Figure 148. Servlet redirector example

15.1 Creating a standalone redirector

In our example, we are using a standalone redirector. The redirector will be configured on a Windows NT machine, node 23bk67d. Use the custom install

for WebSphere to install the application server and the plug-in for your HTTP server. A standalone redirector does not use a DB2 database.

The Topology One HTML and image files required for the Web server should also be copied to the redirector machine (see 12.2, “Separating the application server from the Web server” on page 250).

1. First, you need to add the IP address and hostname of the new servlet redirector machine to the default_host alias list using the administrative console on the application server machine (23bk63z). Instructions for this can be found in 14.2, “Preparing the WebSphere administrative domain” on page 288.
2. Stop and restart the Topology One Model. This is necessary for the changes you just made to take effect.
3. On the redirector machine, 23bk67d, create a batch file called pluginConfig.bat in WebSphere’s bin directory. In our example, this is in C:\WebSphere\AppServer\bin. A sample file is given in Figure 149. The three lines that make up the Java command are actually on one line. They have been broken up here to fit the frame. This file will fetch the plug-in configuration from the administrative domain.


```

@echo off
setlocal
call setupCmdLine.bat

rem setup the classpath
set WAS_CP=%WAS_HOME%\lib\ibmwebas.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\properties
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\servlet.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\webtlsrm.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\lotusxsl.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ns.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ejb.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ujc.jar
set WAS_CP=%WAS_CP%;%DB2DRIVER%
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\repository.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\admin.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\swingall.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\console.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\tasks.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\xml4j.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\x509v1.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\vaprt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\iioprt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\iioprttools.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\dertrjrt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\sslight.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ibmjndi.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\deployTool.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\databeans.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\classes
set WAS_CP=%WAS_CP%;%JAVA_HOME%\lib\classes.zip
set WAS_CP=%WAS_CP%;%JAVA_HOME%\lib\jsp10.jar
set CLASSPATH=%WAS_CP%

java com.ibm.servlet.engine.oselister.systemsmgmt.StandalonePluginCfg
-serverRoot %WebSphere\AppServer -adminNodeName 23bk63z -queueProps
%WebSphere\AppServer\properties\iiopredictor.xml

endlocal

```

Figure 149. pluginConfig.bat

4. In WebSphere's properties directory, (C:\WebSphere\AppServer\properties), edit the iiopredictor.xml file and update the admin-node-name and name-service-node-name as shown in Figure 150. The node name should be the same as specified in the pluginConfig.bat file.

```

<?xml version="1.0"?>
<iiop-redirector>
  <active-transport>ose</active-transport>
  <transport>
    <name>ose</name>
    <code>com.ibm.servlet.engine.oselister.SMQTransport</code>
    <arg name="port" value="8110"></arg>
    <arg name="queueName" value="queue1"></arg>
    <arg name="maxConcurrency" value="50"></arg>
    <arg name="type" value="local"></arg>
    <arg name="server_root" value="$server_root"></arg>
    <arg name="cloneIndex" value="1"></arg>
  </transport>
  <admin-node-name>23bk63z</admin-node-name>
  <qualify-home-names>true</qualify-home-names>
  <name-service-node-name>23bk63z</name-service-node-name>
  <name-service-port>900</name-service-port>
</iiop-redirector>

```

Figure 150. *iiopredirector.xml*

5. In WebSphere's bin directory, create a batch file called `redirector.bat`. You can take the file contents of Figure 151 as an example. This file will be the actual standalone redirector.

```

@echo off
setlocal
call setupCmdLine.bat

rem setup the classpath

set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ibmwebas.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\properties
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\servlet.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\webtlsrn.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\lotusxsl.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ns.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ejb.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ujc.jar
set WAS_CP=%WAS_CP%;%DB2DRIVER%
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\repository.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\admin.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\swingall.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\console.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\tasks.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\xml4j.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\x509v1.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\vaprt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\iioprt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\iioprttools.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\dertrjrt.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\sslight.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\ibmjndi.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\deployTool.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\lib\databeans.jar
set WAS_CP=%WAS_CP%;%WAS_HOME%\classes
set WAS_CP=%WAS_CP%;%JAVA_HOME%\lib\classes.zip
set CLASSPATH=%WAS_CP%

java -Dcom.ibm.CORBA.ListenerPort=12301 -Dserver.root=%WAS_HOME%
com.ibm.servlet.engine.ejs.IIOPRedirector

endlocal

```

Figure 151. *redirector.bat*

Note: the first argument in the Java command is required for communication over a firewall. This is explained in the next section.

15.2 Preparing to use firewalls

The communication between the servlet redirector and the WebSphere Application Servers is done using RMI/IIOP over TCP. When the WebSphere process starts, a TCP port is selected to listen on from a range of available ports. The process registers its port with the WebSphere Location Service Daemon (LSD), which runs on a well-known port. Processes that need to

communicate with a WebSphere process obtain the port information from the administrative server.

When you introduce a firewall into the configuration, this process would mean the firewall would have to be open on the entire range of possible ports. To avoid this, you will need to assign ports to the WebSphere administrative server and to the application servers for this communication.

1. First, you need to assign a port for the administrative server on 23bk63z to listen on. The port chosen should be a valid (unused) TCP port. In this case, we chose port 12101.

The administrative server is on 23b6k3z. On that node, edit `C:\WebSphere\AppServer\bin\admin.config`. Modify the following line:

```
com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs = -mx128m
```

to:

```
com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs=  
-Dcom.ibm.CORBA.ListenerPort=12101 -mx128m
```

as the first argument. Any other arguments should follow.

2. Next, you need to assign a port for the administration server on rs60001 to listen on. In this case the port will be 12102.

On rs60001, open the file `/usr/WebSphere/bin/admin.config` and modify the line containing `com.ibm.ejs.sm.util.process.Nanny.adminServerJvmArgs` and add `-Dcom.ibm.CORBA.ListenerPort=12102` as the first argument. The argument `-mx128m` should follow.

3. Next, you need to assign a port for the application server's Object Request Broker (ORB) to listen on.

Using the administrative console, highlight DS 1 under the Topology tab. Add `-Dcom.ibm.CORBA.ListenerPort=12201` as the first command-line argument.

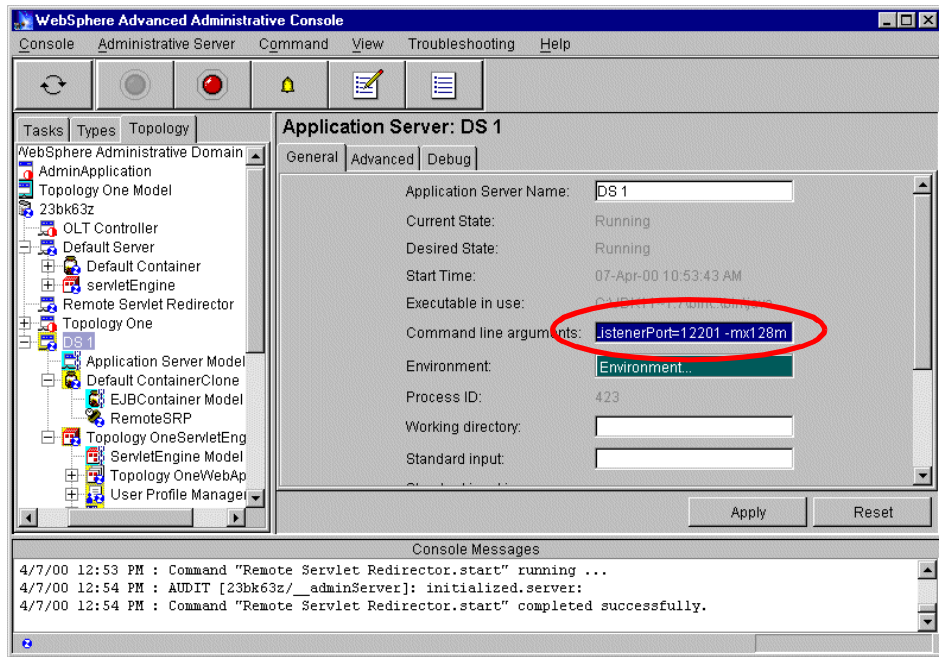


Figure 152. Adding ListenerPort to Web servers

4. Make the same changes to DS 2 using port 12202, and to DS 3 using port 12203.

Note: The ports mentioned in the previous four steps will need to be configured on the domain firewall when you are setting up your firewall configuration. The firewall configuration is discussed in Chapter 16, “Setting up firewalls” on page 305.

5. Stop the standalone redirector on 23bk67d.
6. In the redirector.bat file, add `-Dcom.ibm.CORBA.ListenerPort=12301` as the first parameter of the Java command.
7. Stop the administrative server on both rs60001 and 23bk63z.
8. Start the administrative server on both rs60001 and 23bk63z.
9. Verify that Topology One is running. If this is not the case, start it.
10. Start the standalone redirector on 23bk67d.

11. Configure the domain firewall. The configuration of the firewall should contain the rules described in Table 15.

Table 15. Firewall configuration

source	target	port
23bk67d	rs60001	900 (bootstrap)
		9000 (LSD)
		12101 (EJS)
		12202 (DS 2)
	23bk63z	900 (bootstrap)
		9000 (LSD)
		12102 (EJS)
		12201 (DS 1)
		12203 (DS 3)
rs60001	23bk67d	12301 (redirector)
23bk63z		

15.3 Testing the redirector

The next step is to test your redirector and application:

1. Stop the Web server on the redirector machine, 23bk67d.
2. Run pluginConfig.bat. You will have to run this batch file each time you make some changes to the application servers' configuration. You will temporarily need to turn off security on the WebSphere Application Server node.
3. Restart the Web server.
4. Run redirector.bat. If you want to stop the redirector, press CTRL+C.
5. Verify that the servlet redirector is working properly. On the redirector machine, invoke the URL:

`http://localhost/`

and follow the path for the Topology One application.

Chapter 16. Setting up firewalls

This chapter will show the definitions used to set up the protocol and domain firewalls for our example. The firewall software used was IBM SecureWay Firewall for Windows NT V4.1. This is not an in-depth discussion but is designed to show firewall security experts what is needed specifically for this network environment. For more information on the IBM Firewall see the following two redbooks:

- *A Secure Way to Protect Your Network: IBM SecureWay Firewall for AIX V4.1*, SG24-5855-00
- *Guarding the Gates Using the IBM eNetwork Firewall V3.3 for Windows NT*, SG24-5209

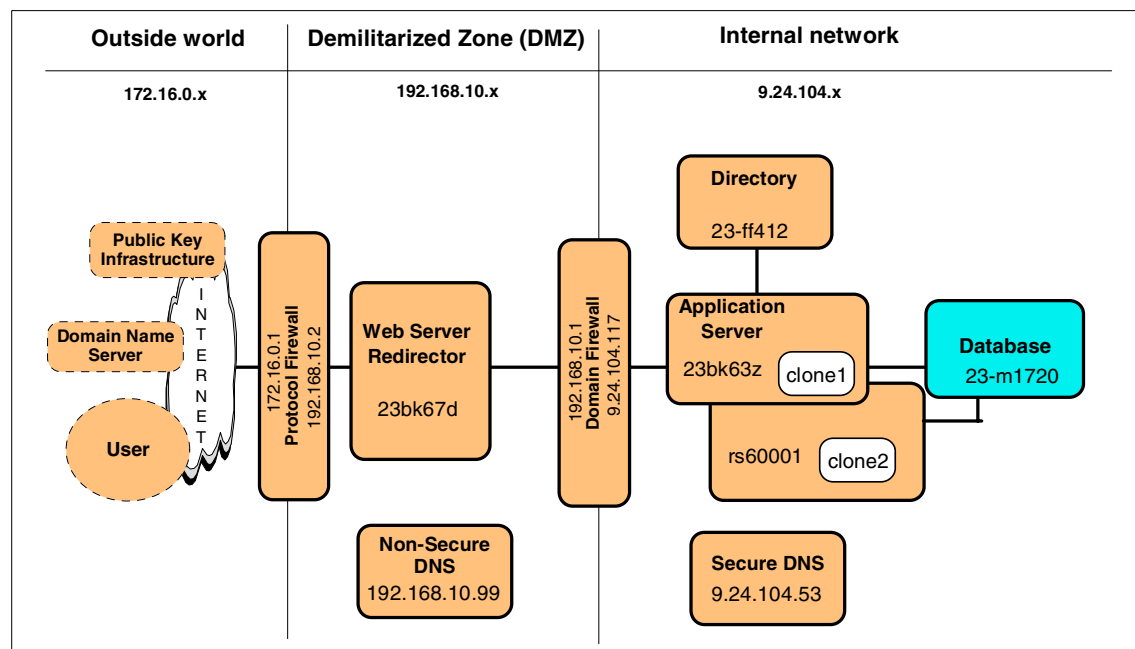


Figure 153. Setting up firewalls

This scenario consists of three network segments:

- The public network (outside world). This is usually the origin of the client requests. This network is considered to be non-secure.
- The Demilitarized Zone (DMZ). The DMZ is the network between the two firewalls. The DMZ is protected from the public network by a protocol

firewall, which limits the type of access that passes through the firewall to the nodes in the DMZ.

- The internal (or enterprise) network. The internal network is where the resources you want to protect reside. It is separated from the DMZ by a domain firewall that further limits traffic that has passed through the DMZ.

Before you install the firewall software, please use the Planning and Network Configuration Planning Worksheet in *IBM SecureWay Firewall for Windows NT User's Guide V4.1*. When installing the software, follow the instructions in *IBM SecureWay Firewall for Windows NT Setup and Installation V4.1*.

After installation, there are seven steps to configure the firewall. The configuration is done using the SecureWay Firewall Configuration Client. The first time you start the configuration client, a setup wizard will take you through the configuration steps.

16.1 Designating the network interfaces

The first step is to define which network interfaces the firewall is using and if they are secure or not. There must be at least one secure and one non-secure interface.

16.1.1 Domain firewall

The domain firewall is situated between the DMZ, which uses IP addresses in the range of 192.168.10.xx, and the internal network, which uses addresses in the range of 9.24.104.xx. The domain firewall machine has two token-ring network adapters, with one adapter and IP address in each network. In this instance, the internal network is considered to be the secure network.

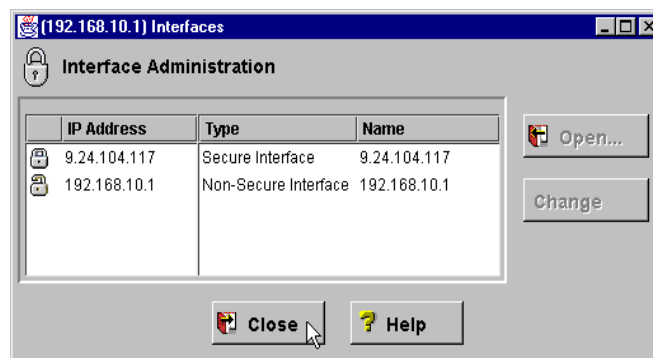


Figure 154. Configure interfaces for domain firewall

16.1.2 Protocol firewall

The protocol firewall is situated between the DMZ, which uses IP addresses in the range of 192.168.10.xx, and the external network, which uses addresses in the range of 172.16.0.xx. The protocol firewall machine has two token-ring network adapters, with one adapter and IP address in each network. In this instance, the DMZ is considered to be the secure network.

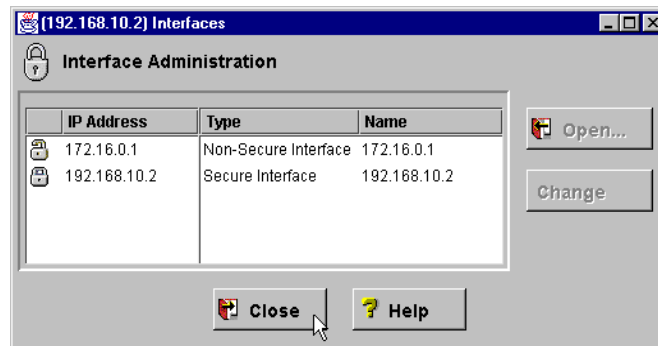


Figure 155. Configure interfaces for protocol firewall

16.2 Setting up the general security policy

The next step is to set up the general security policy. We have selected the following options as part of the general security policy on both firewalls.

- Permit DNS queries
- Deny broadcast messages
- Deny SOCKs

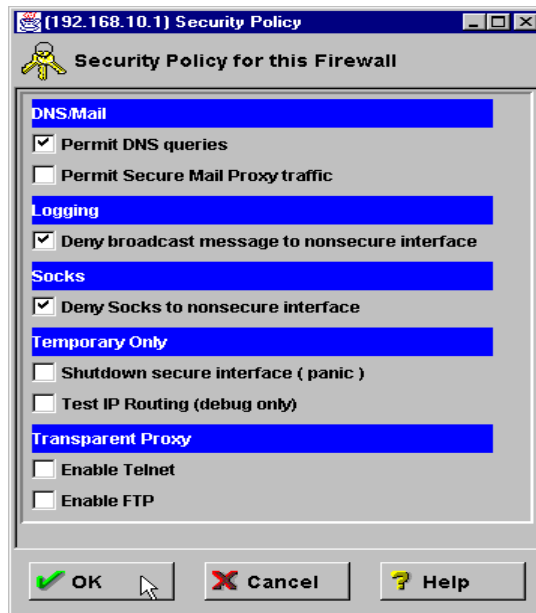


Figure 156. Security Policy configuration for both domain and protocol firewall

16.3 Creating the network objects

Next, you create the network objects. Network objects will be used as the source objects and/or the destination objects when you build your firewall connections. You can create either single objects (a host, firewall, router) or group objects (a group of single objects).

16.3.1 Domain firewall

Figure 157 shows the network objects created for the domain firewall.

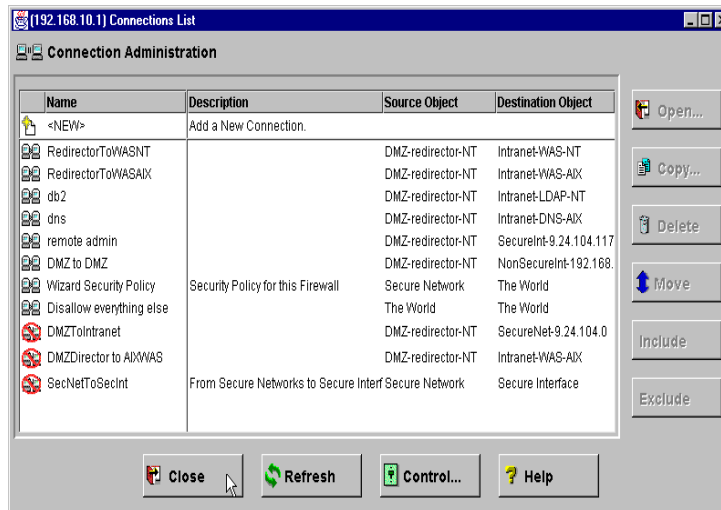


Figure 157. Network objects for the domain firewall

16.3.2 Protocol firewall

Figure 158 shows the network objects created for the protocol firewall.

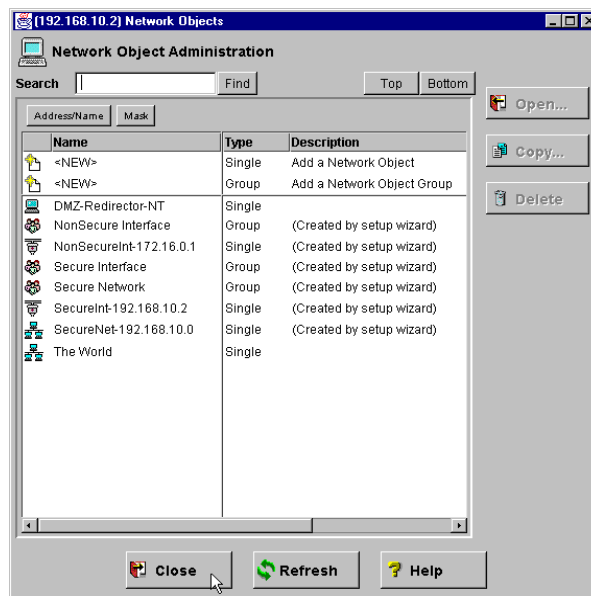


Figure 158. Network objects for the protocol firewall

16.4 Configuring the domain name service

The next step is to configure domain name service. We used the following configuration:

- Secure Domain Name - itso.ral.ibm.com
- Secure Domain Name Server - 9.24.104.53
- Non-Secure Domain Name Server - 192.168.10.99

16.5 Creating the firewall rules and services

The next step is to create the firewall rules and services.

16.5.1 Domain firewall rules and services

Each service is described below with the rule definitions that make up the service.

16.5.1.1 Service name: WAS-Bootstrap

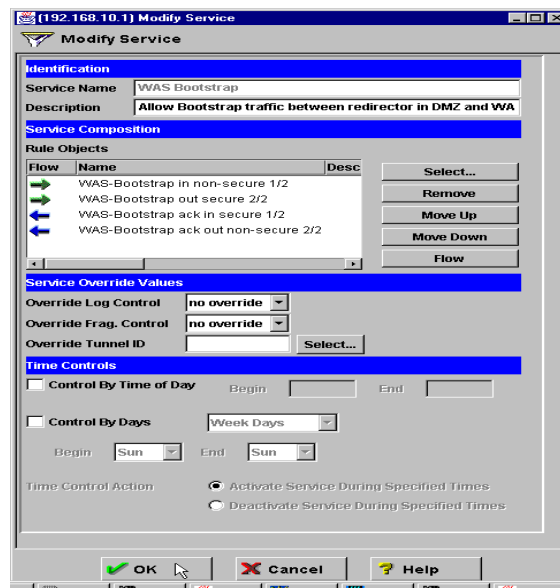


Figure 159. WAS-Bootstrap service in domain firewall

Rule name 1: WAS-Bootstrap in non-secure 1/2

Table 16. WAS-Bootstrap in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	900	non-secure	route	inbound

Rule name 2: WAS-Bootstrap out secure 2/2

Table 17. WAS-Bootstrap in non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	900	secure	route	outbound

Rule name 3: WAS-Bootstrap ack in secure 1/2

Table 18. WAS-Bootstrap ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp/ack	eq	900	gt	1023	secure	route	inbound

Rule name 4: WAS-Bootstrap ack out non-secure 2/2

Table 19. WAS-Bootstrap ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp/ack	eq	900	gt	1023	non-secure	route	outbound

16.5.1.2 Service name: WAS-LSD

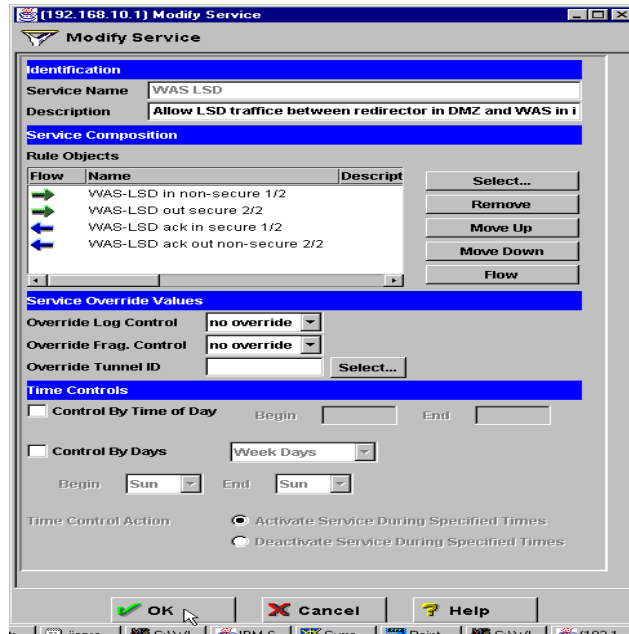


Figure 160. WAS-LSD service in domain firewall

Rule name 1: WAS-LSD in non-secure 1/2

Table 20. WAS-LSD in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	9000	non-secure	route	inbound

Rule name 2: WAS-LSD out secure 2/2

Table 21. WAS-LSD out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	9000	secure	route	outbound

Rule name 3: WAS-LSD ack in secure 1/2

Table 22. WAS-LSD ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	9000	gt	1023	secure	route	inbound

Rule name 4: WAS-LSD ack out non-secure 2/2

Table 23. WAS-LSD ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	9000	gt	1023	non-secure	route	outbound

16.5.1.3 Service name: WAS-EJS-AIX

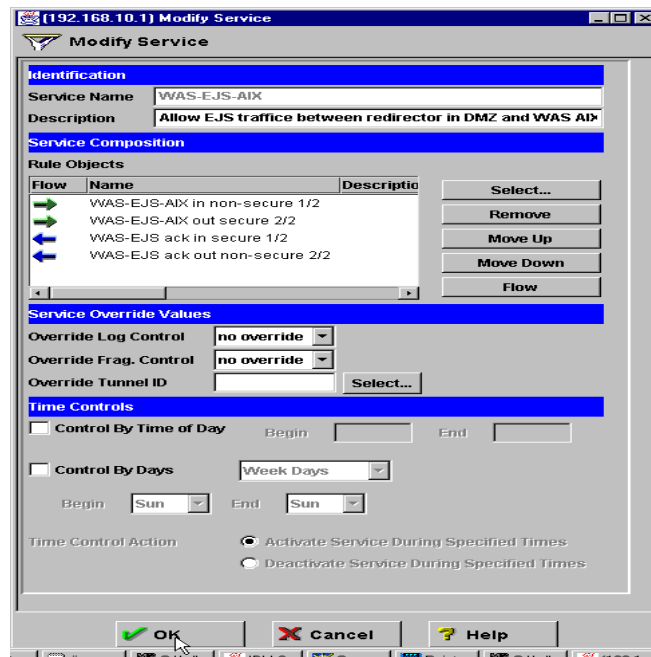


Figure 161. WAS-EJS-AIX service in domain firewall

Rule name 1: WAS-EJS-AIX in non-secure 1/2

Table 24. WAS-EJS-AIX in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12101	non-secure	route	inbound

Rule name 2: WAS-EJS-AIX out secure 2/2

Table 25. WAS-EJS-AIX out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12101	secure	route	outbound

Rule name 3: WAS-EJS-AIX ack in secure 1/2

Table 26. WAS-EJS-AIX ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12101	gt	1023	secure	route	inbound

Rule name 4: WAS-EJS-AIX ack out non-secure 2/2

Table 27. WAS-EJS-AIX ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12101	gt	1023	non-secure	route	outbound

16.5.1.4 Service name: WAS-DS1 Service

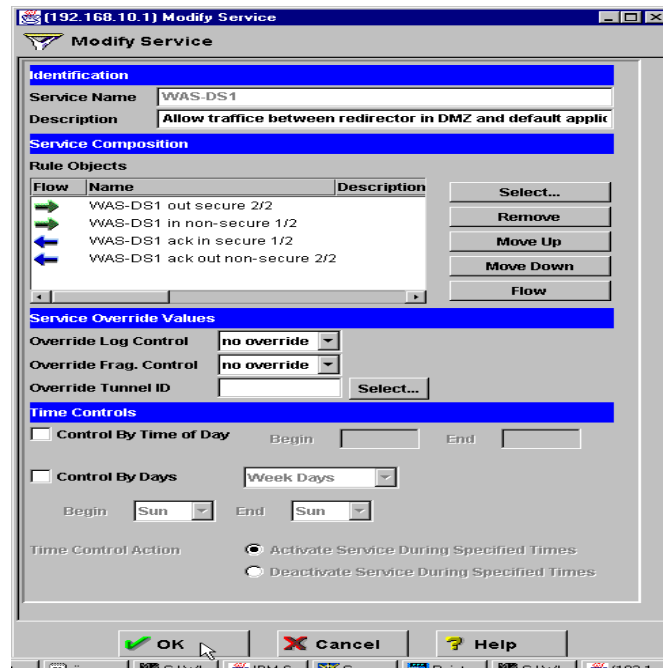


Figure 162. WAS-DS1 service in domain firewall

Rule name 1: WAS-DS1 in non-secure 1/2

Table 28. WAS-DS1 in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12201	non-secure	route	inbound

Rule name 2: WAS-DS1 out secure 2/2

Table 29. WAS-DS1 out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12201	secure	route	outbound

Rule name 3: WAS-DS1 ack in secure 1/2

Table 30. WAS-DS1 ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12201	gt	1023	secure	route	inbound

Rule name 4: WAS-DS1 ack out non-secure 2/2

Table 31. WAS-DS1 ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12201	gt	1023	non-secure	route	outbound

16.5.1.5 Service name: WAS-DS2

Rule name 1: WAS-DS2 in non-secure 1/2

Table 32. WAS-DS2 in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12202	non-secure	route	inbound

Rule name 2: WAS-DS2 out secure 2/2

Table 33. WAS-DS2 out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12202	secure	route	outbound

Rule name 3: WAS-DS2 ack in secure 1/2

Table 34. WAS-DS2 ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12202	gt	1023	secure	route	inbound

Rule name 4: WAS-DS2 ack out non-secure 2/2

Table 35. WAS-DS2 ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12202	gt	1023	non-secure	route	outbound

16.5.1.6 Service name: WAS-DS3

Rule name 1: WAS-DS3 in non-secure 1/2

Table 36. WAS-DS3 in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12203	non-secure	route	inbound

Rule name 2: WAS-DS3 out secure 2/2

Table 37. WAS-DS3 out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12203	secure	route	outbound

Rule name 3: WAS-DS3 ack in secure 1/2

Table 38. WAS-DS3 ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12203	gt	1023	secure	route	inbound

Rule name 4: WAS-DS3 ack out non-secure 2/2

Table 39. WAS-DS3 ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12203	gt	1023	non-secure	route	outbound

16.5.1.7 Service name: WAS-Redirector

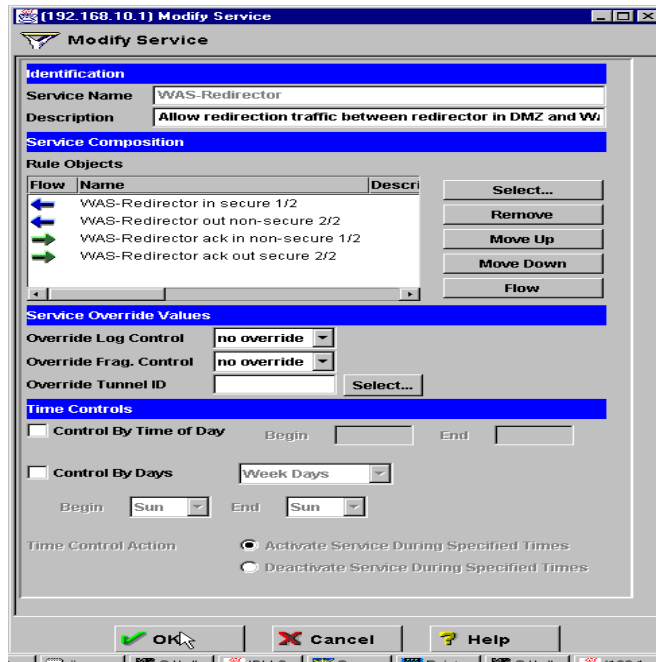


Figure 163. WAS-Redirector service in domain firewall

Rule name 1: WAS-Redirector in secure 1/2

Table 40. WAS-Redirector in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12301	secure	route	inbound

Rule name 2: WAS-Redirector out non-secure 2/2

Table 41. WAS-Redirector out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	12301	non-secure	route	outbound

Rule name 3: WAS-Redirector ack in non-secure 1/2

Table 42. WAS-Redirector ack in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12301	gt	1023	non-secure	route	inbound

Rule name 4: WAS-Redirector ack out secure 2/2

Table 43. WAS-Redirector ack out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	12301	gt	1023	secure	route	outbound

16.5.2 Protocol firewall rules and services

The following are the rules and services defined for the protocol firewall.

16.5.2.1 Service name: HTTP Internet to DMZ

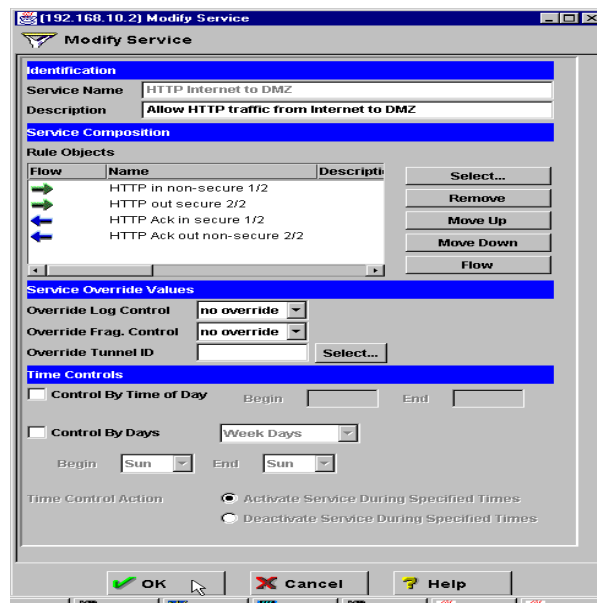


Figure 164. HTTP Internet to DMZ service in the protocol firewall

Rule name 1: HTTP in non-secure 1/2

Table 44. HTTP in non-secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	80	non-secure	route	inbound

Rule name 2: HTTP out secure 2/2

Table 45. HTTP out secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	gt	1023	eq	80	secure	route	outbound

Rule name 3: HTTP ack in secure 1/2

Table 46. HTTP ack in secure 1/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	80	gt	1023	secure	route	inbound

Rule name 4: HTTP ack out non-secure 2/2

Table 47. HTTP ack out non-secure 2/2

Action	Protocol	Operation at Source	Port # at source	operation at dest	port # at dest	interface	routing	direction
permit	tcp	eq	80	gt	1023	non-secure	route	outbound

16.5.2.2 Service name: DNS DMZ to Intranet

The rules for this service are pre-defined with the SecureWay Firewall installation.

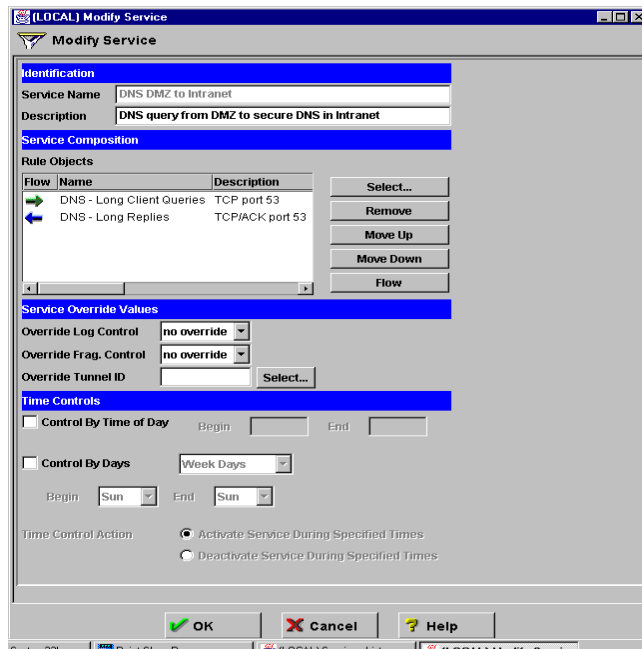


Figure 165. DNS DMZ to intranet

16.6 Building connections on the firewall

The next step is to define the connections on the firewalls. Once the connections for the various network objects have been built, you have to enable the connections for the firewall rules to be active.

16.6.1 Domain firewall

Figure 166 shows the connections used for the domain firewall in this example.

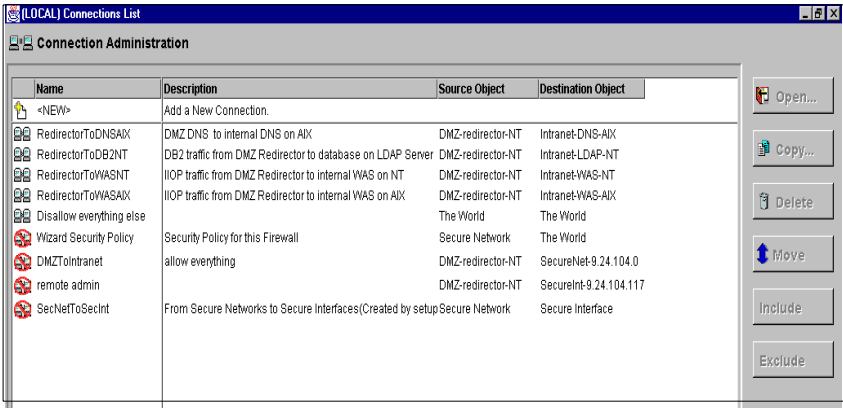


Figure 166. Connection configuration for domain firewall

16.6.2 Protocol firewall

Figure 167 shows the connections used for the domain firewall in this example.

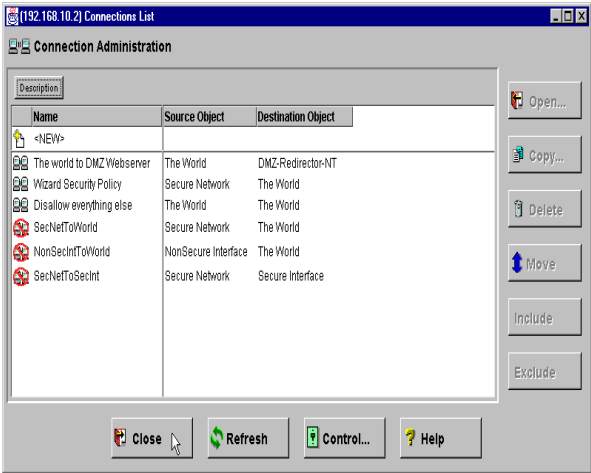


Figure 167. Connection configuration for the protocol firewall

16.7 TCP/IP routing

TCP/IP routing plays a very important role in firewall setup. For machines in the DMZ (for example, the Web server redirector), ensure that their routing table includes entries to both the protocol and domain firewalls. The firewalls should allow IP forwarding.

The Web server redirector machine (192.168.10.99) in the DMZ should have the following routes added:

Table 48. Routing example for DMZ Web server redirector

Destination	Mask	Gateway	Interface
0.0.0.0	0.0.0.0	192.168.10.1	192.168.10.99
0.0.0.0.	0.0.0.0	192.168.10.2	192.168.10.99

Chapter 17. SecureWay Directory Configuration

If you want to use SecureWay Directory for security, you will need to set up the directory and define the users. Once the IBM SecureWay Directory is installed, use the following steps as an example of how to configure the directory.

1. Immediately after installation, the Directory product will need to have the initial configuration defined. Open the directory configuration GUI by clicking **Start -> Programs -> IBM SecureWay Directory-> SecureWay Directory Configuration**. Select all the components to configure.

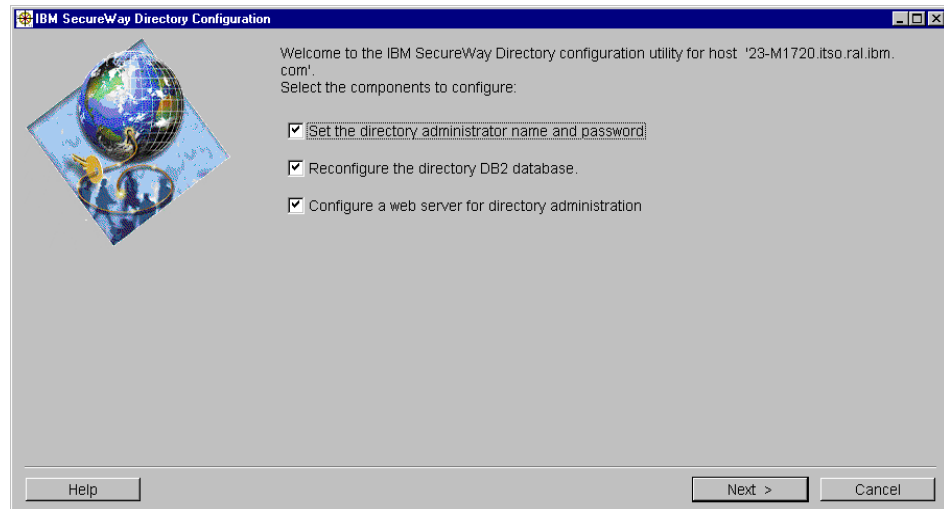


Figure 168. Admin console - security

Click **Next**.

2. Enter a user ID and password for the directory administrator. To stay consistent with the PDK, we used the distinguished name (DN) of `cn=USERID` and password of `PASSWORD`.

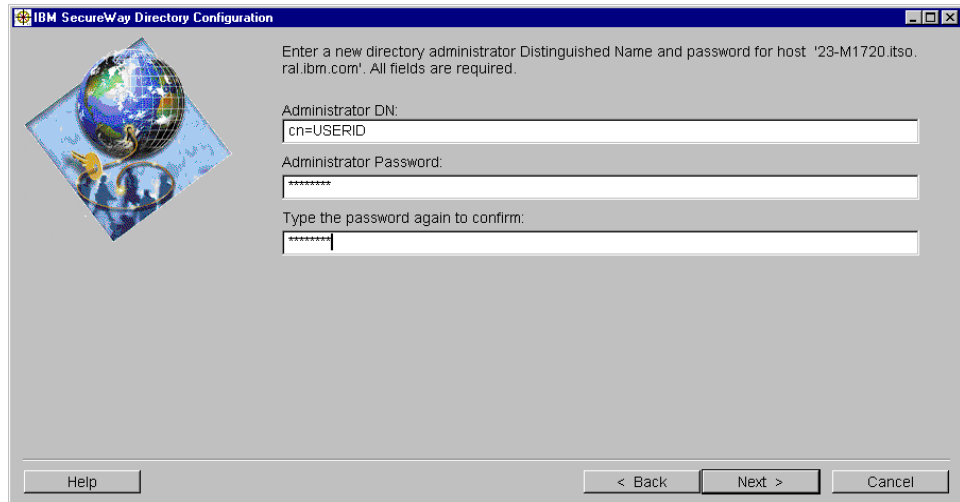


Figure 169. Admin console - security

Click **Next**.

In the next windows:

3. Use the default database (LDAPDB2).
4. Use the appropriate character set for the database.
5. Choose the radio button to create the default DB2 database.
6. Choose the drive location for the LDAP database.
7. Select the Web server that is already installed on your system. The correct Web server configuration file and path should be automatically selected.
8. On the confirmation window, click **Configure** to begin configuration. If everything goes well, each step will show a success status on the final window.
9. Click **OK**. At this point the LDAPDB2 service should have started.

Tip: If the DB2 database creation fails, make sure that any old DB2 and LDAP directories from previous installs are removed.

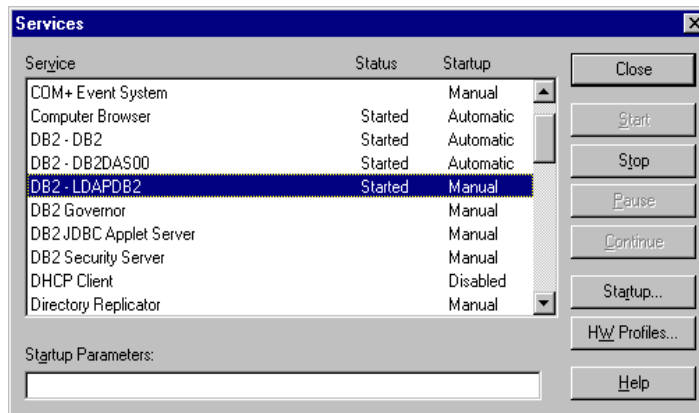


Figure 170. Admin console - security

10. Stop and restart the Web server.

11. Start the IBM SecureWay Directory service. You can do this from the Windows NT Services panel.

17.0.1 Administering the SecureWay Directory LDAP server

The LDAP server can be administered via the Web-based GUI. To bring up the GUI enter the following URL: <http://localhost/ldap> in your browser. Enter the user ID (cn=USERID) and password (PASSWORD), that was entered during the SecureWay Directory configuration steps. Then click **Logon**.

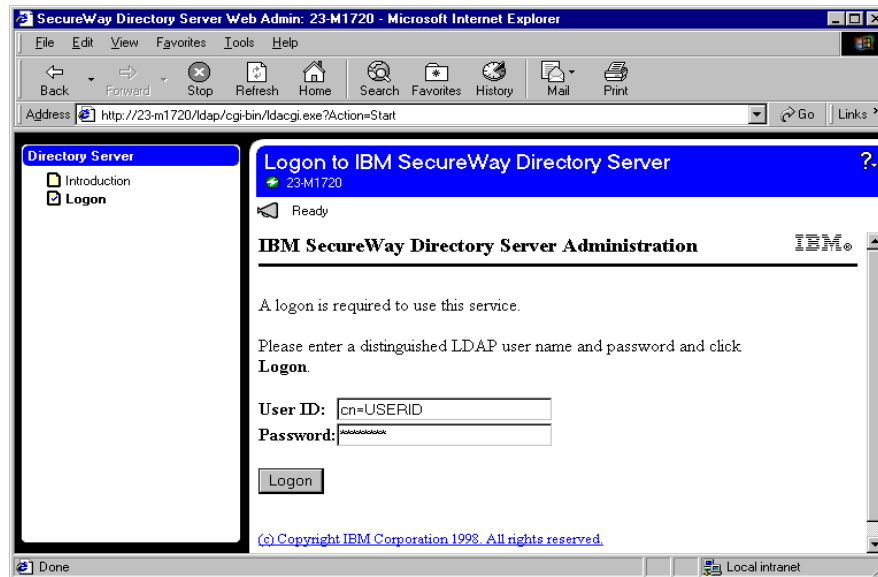


Figure 171. SecureWay Directory Server administration

17.0.2 Working with the directory tree

The first step is to add a suffix. A suffix is a distinguished name that identifies the top entry in a locally held directory hierarchy. A directory server may have multiple suffixes, each identifying a locally held directory hierarchy.

To add a suffix using the browser GUI, expand the Suffixes category and choose **Add a suffix**. For the sample, enter `o=ibm,c=uk` as the suffix and click **Add a new suffix**.

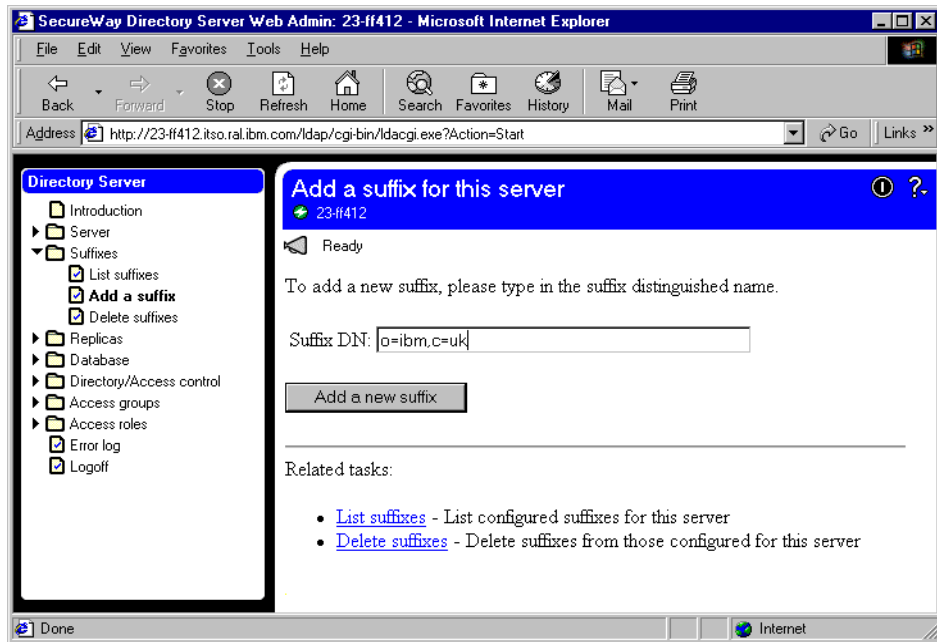


Figure 172. Adding a suffix

12. After adding the sample suffix, the directory data needs to be imported to the database.

- Expand the Database category and choose **Add entries**.
- The sample LDIF (LDAP Data Interchange Format) file and path should be selected.
- Click **Add entries to database**.

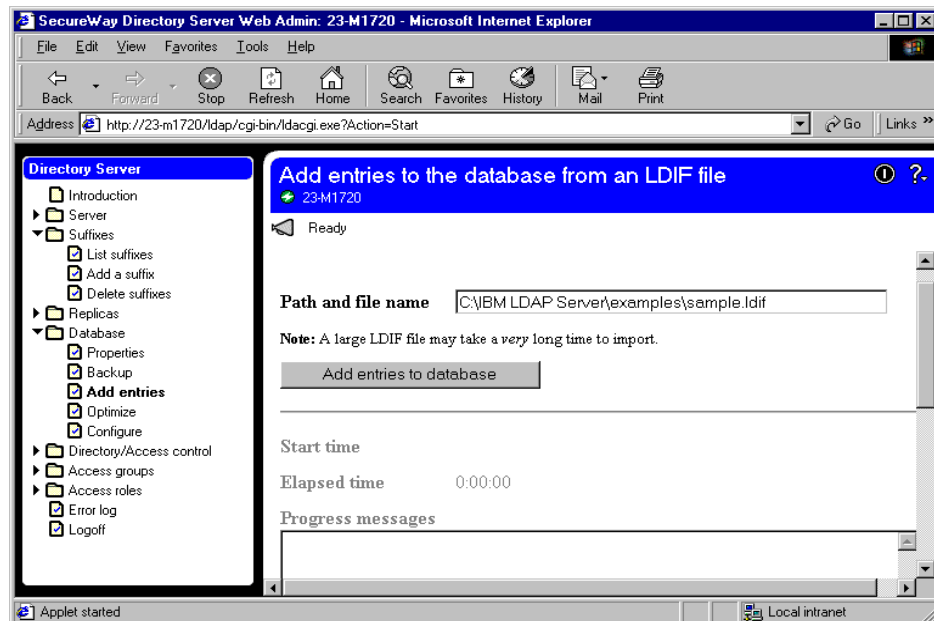


Figure 173. Adding entries to the database

Wait until all of the processing is done. If successful, 50 entries should be added and the message will indicate that.

Close the GUI and restart the server from the NT Services panel.

17.0.3 Using DMT to add your own directory entries

The next step will be to populate the directory with user entries. For our example, we are only going to add an organizational unit and one new user. To add entries to the directory:

1. Bring up the Directory Management Tool (DMT):

Start ->Programs ->IBM SecureWay Directory ->Directory Management Tool

If you go to the Properties window under the Server category, you can see the port being used by the LDAP server (389). This must match what you specify in the WebSphere security configuration (see Figure 131 on page 280).

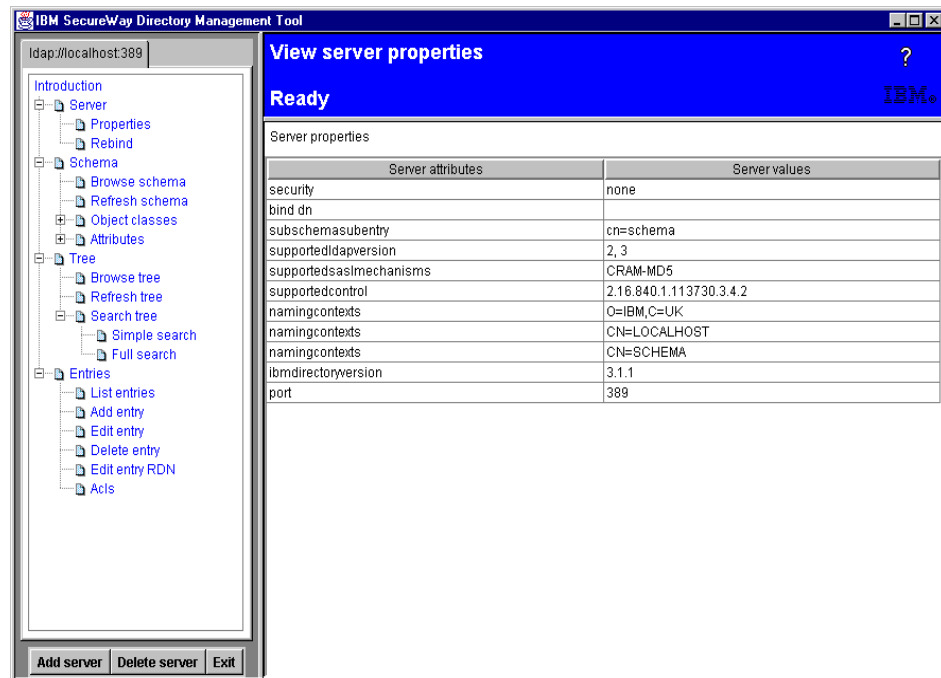


Figure 174. Directory server properties

- Go to the Rebind window and authenticate yourself. Use `cn=USERID` for the user and `PASSWORD` for the password. Then click **OK**.

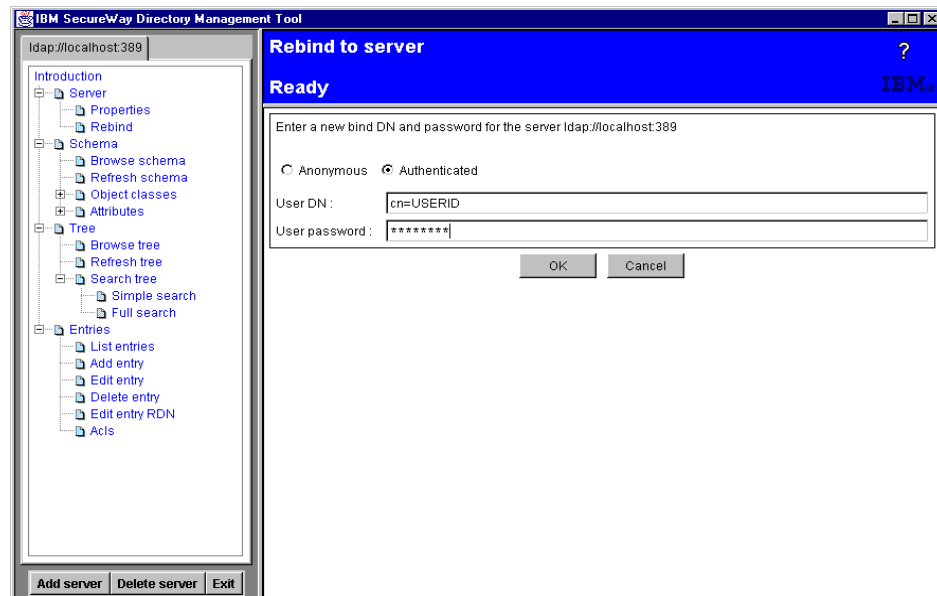
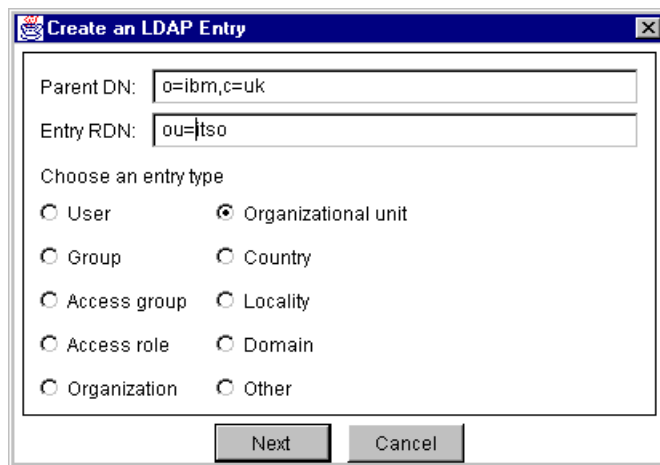


Figure 175. Rebinding the server

3. As soon as you click OK, the Browse directory tree window will be displayed. To add your personal entry, highlight the level at which you want to add and click **Add**.

For example, to add a new Organizational Unit (OU) of "itso", in the Entry RDN text field type `ou=itso` and choose **Organizational unit** for the entry type. Then click **Next**.



Create an LDAP Entry

Parent DN:

Entry RDN:

Choose an entry type

☐ User
 ☒ Organizational unit

☐ Group
 ☐ Country

☐ Access group
 ☐ Locality

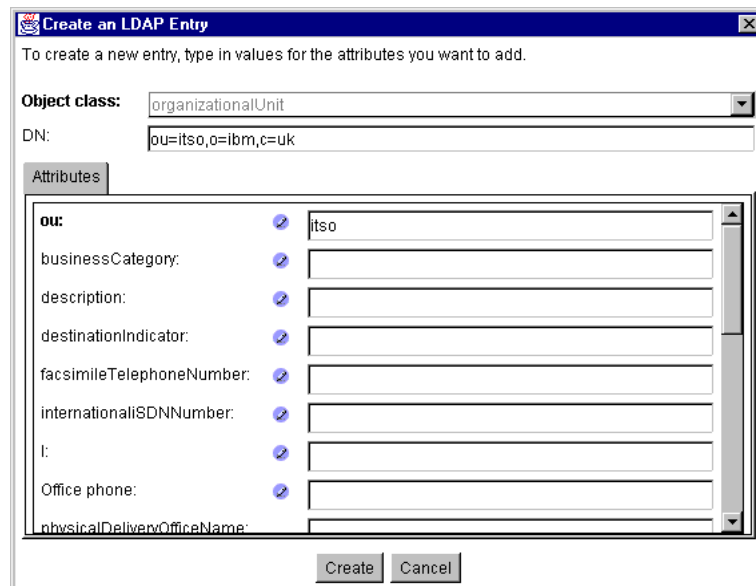
☐ Access role
 ☐ Domain

☐ Organization
 ☐ Other

Next Cancel

Figure 176. Creating an entry

- The attributes window will pop up. Enter `itso` in the `ou` field. Fill in the other details you wish and click **Create**.



Create an LDAP Entry

To create a new entry, type in values for the attributes you want to add.

Object class:

DN:

Attributes

ou:	<input checked="" type="checkbox"/>	<input type="text" value="itso"/>
businessCategory:	<input checked="" type="checkbox"/>	<input type="text"/>
description:	<input checked="" type="checkbox"/>	<input type="text"/>
destinationIndicator:	<input checked="" type="checkbox"/>	<input type="text"/>
facsimileTelephoneNumber:	<input checked="" type="checkbox"/>	<input type="text"/>
internationalISDNNumber:	<input checked="" type="checkbox"/>	<input type="text"/>
l:	<input checked="" type="checkbox"/>	<input type="text"/>
Office phone:	<input checked="" type="checkbox"/>	<input type="text"/>
physicalDeliveryOfficeName:	<input checked="" type="checkbox"/>	<input type="text"/>

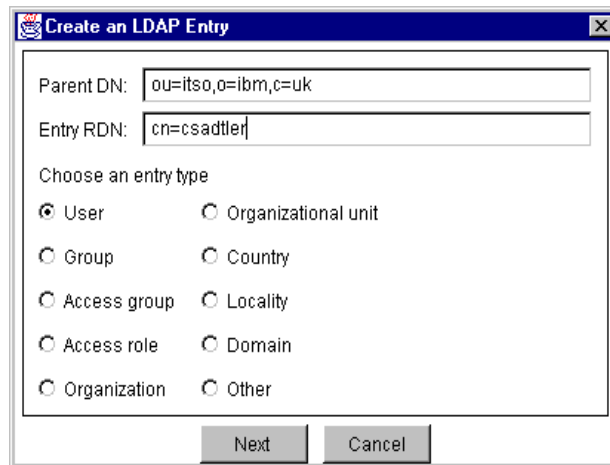
Create Cancel

Figure 177. Adding an organizational unit

17.0.4 Use DMT to add a new user and assign a password

Repeat step 3 in 17.0.3, “Using DMT to add your own directory entries” on page 330 to add an entry.

1. Choose User as the entry type and type the common name in the Entry RDN field (cn=Carla Sadtler). Click **Next**.



The screenshot shows a Windows-style dialog box titled "Create an LDAP Entry". It has two text input fields: "Parent DN:" with the value "ou=itso,o=ibm,c=uk" and "Entry RDN:" with the value "cn=csadtler". Below these fields is a section titled "Choose an entry type" containing ten radio button options arranged in two columns. The first column includes "User" (which is selected), "Group", "Access group", "Access role", and "Organization". The second column includes "Organizational unit", "Country", "Locality", "Domain", and "Other". At the bottom of the dialog are two buttons: "Next" and "Cancel".

Figure 178. Adding a user

2. In the attributes window, type in the password and other details. This is the password you will use when you are challenged. Then click **Create**.

Create an LDAP User

To create a new user, type in a common name, last name, and any other information for the user.

Object class:

DN:

Common name: ☒

Last name: ☒

Initials: ☒

Business **Personal** **Other**

Employee type: ☒

Manager: ☒

Office number: ☒

Office phone: ☒

Pager number: ☒

Secretary: ☒

Title: ☒

userPassword: ☒

Figure 179. User attributes

3. You can test your entry and password addition by logging on to the LDAP server via the Web GUI using the DN (`cn=sadtler,ou=ITSO,o=IBM,c=US`) that you just created.

Note: If you want to avoid the hassle of typing in the complete DN, set the `uid` property. You can use this one instead of the DN whenever WebSphere challenges you.

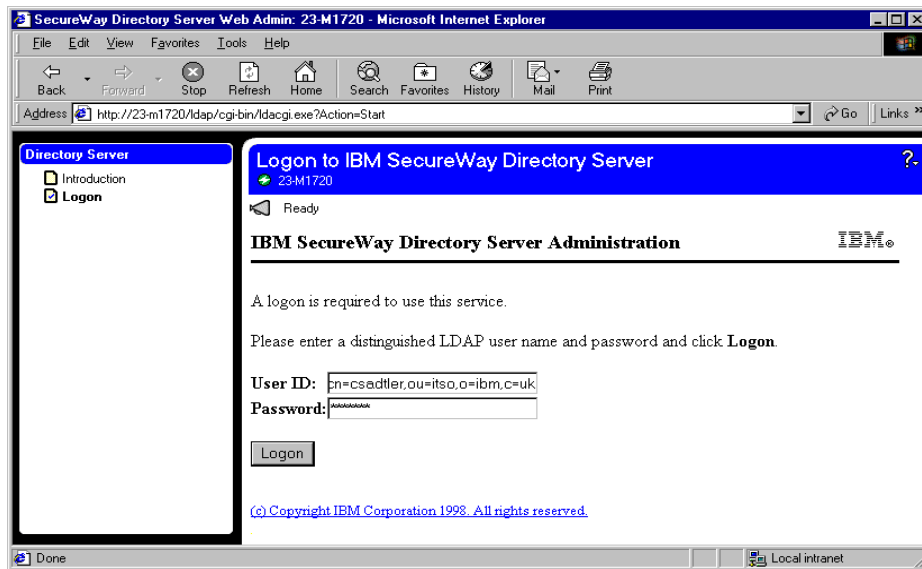


Figure 180. Logging in with the new user ID

Note: If you use the administrator ID (cn=USERID), you can see the entire directory tree.

Appendix A. Special notices

This publication is intended to help IT architects and IT specialists in the design and deployment of e-business applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by WebSphere. See the PUBLICATIONS section of the IBM Programming Announcement for IBM WebSphere Application Server V 3.0.2, Advanced Edition, 41L0696, for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM's product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM's intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM Corporation, Dept. 600A, Mail Drop 1329, Somers, NY 10589 USA.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The information about non-IBM ("vendor") products in this manual has been supplied by the vendor and IBM assumes no responsibility for its accuracy or completeness. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item may have

been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Any pointers in this publication to external Web sites are provided for convenience only and do not in any manner serve as an endorsement of these Web sites.

Any performance data contained in this document was determined in a controlled environment, and therefore, the results that may be obtained in other operating environments may vary significantly. Users of this document should verify the applicable data for their specific environment.

This document contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples contain 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.

Reference to PTF numbers that have not been released through the normal distribution process does not imply general availability. The purpose of including these reference numbers is to alert IBM customers to specific information relative to the implementation of the PTF when it becomes available to each customer according to the normal IBM PTF distribution process.

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

AIX	AS/400
CICS	DB2
IBM	MQSeries
Nways	OS/390
OS/400	RS/6000
S/390	SecureWay
SP	TeamConnection
VisualAge	WebSphere
Wizard	Lotus
Approach	Domino
Tivoli	Tivoli Certified

The following terms are trademarks of other companies:

Tivoli, Manage. Anything. Anywhere., The Power To Manage., Anything. Anywhere., TME, NetView, Cross-Site, Tivoli Ready, Tivoli Certified, Planet

Tivoli, and Tivoli Enterprise are trademarks or registered trademarks of Tivoli Systems Inc., an IBM company, in the United States, other countries, or both. In Denmark, Tivoli is a trademark licensed from Kjøbenhavns Sommer - Tivoli A/S.

C-bus is a trademark of Corollary, Inc. in the United States and/or other countries.

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

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

PC Direct is a trademark of Ziff Communications Company in the United States and/or other countries and is used by IBM Corporation under license.

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States and/or other countries.

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

SET 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.

Appendix B. Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

B.1 IBM Redbooks publications

For information on ordering these publications see “How to get IBM Redbooks” on page 345.

- *Understanding LDAP*, SG24-4986
- *Developing an e-business Application for IBM WebSphere Application Server*, SG24-5423-00
- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265-00.
- *WebSphere V3 Performance Tuning Guide for AIX*, SG24-5657
- *VisualAge for Java Enterprise Version 2: Data Access Beans - Servlets - CICS Connector*, SG24-5265-00
- *A Secure Way to Protect Your Network: IBM SecureWay Firewall for AIX V4.1*, SG24-5855-00
- *Guarding the Gates Using the IBM eNetwork Firewall V3.3 for Windows NT*, SG24-5209
- *WebSphere Studio and VisualAge for Java Servlet and JSP Programming*, SG24-5755-00, available as a redpiece at <http://www.redbooks.ibm.com>
- *Servlet/JSP/EJB Design and Implementation Guide*, SG24-5754

B.2 IBM Redbooks collections

Redbooks are also available on the following CD-ROMs. Click the CD-ROMs button at <http://www.redbooks.ibm.com/> for information about all the CD-ROMs offered, updates and formats.

CD-ROM Title	Collection Kit Number
System/390 Redbooks Collection	SK2T-2177
Networking and Systems Management Redbooks Collection	SK2T-6022
Transaction Processing and Data Management Redbooks Collection	SK2T-8038
Lotus Redbooks Collection	SK2T-8039
Tivoli Redbooks Collection	SK2T-8044
AS/400 Redbooks Collection	SK2T-2849

CD-ROM Title	Collection Kit Number
Netfinity Hardware and Software Redbooks Collection	SK2T-8046
RS/6000 Redbooks Collection (BkMgr)	SK2T-8040
RS/6000 Redbooks Collection (PDF Format)	SK2T-8043
Application Development Redbooks Collection	SK2T-8037
IBM Enterprise Storage and Systems Management Solutions	SK3T-3694

B.3 Other resources

These publications are also relevant as further information sources:

- John Barry et al, *Developing Object-oriented Software - An Experienced-Based Approach*, Prentice Hall, 1997, ISBN 0137372485
- E. Gamma, R. Helm, R. Johnson, J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995, ISBN 0201633612
- F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stahl, *Pattern-Oriented Software Architecture - A System of Patterns*, Wiley, 1996, ISBN 0471958697
- Coplien, J., *Advanced C++ Programming Styles and Idioms*, Addison-Wesley, 1991, ISBN 0201548550
- P. Monday, J. Carey, M. Dangler, *San Francisco Component Framework: An Introduction*, Addison-Wesley, 1999, ISBN 0201615878
- C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, S. Angel, *A Pattern Language*. Oxford University Press, 1977, ISBN 0195019199
- J. Vlissides, *Pattern Hatching - Design Patterns Applied*, Addison Wesley, 1998, ISBN 0201432935
- "Enterprise Solutions Structure" in *IBM Systems Journal*, Volume 38, No. 1, 1999, available at <http://www.research.ibm.com/journal/sj38-1.html>
- L. Bass, P. Clements, R. Kazman, *Software Architecture in Practice*, Addison Wesley, 1998, ISBN 0201199300
- Booch, Grady, *Object-Oriented Analysis and Design with Applications* (Addison-Wesley Object Technology Series), Addison-Wesley, 1994, ISBN 0805353402
- Jacobson, Ivar, *Object-Oriented Software Engineering; A Use Case Driven Approach*, Addison-Wesley, 1992, ISBN 0201544350

- Rumbaugh, James et al., *Object-Oriented Modeling and Design*, Prentice Hall, 1991, ISBN 0136298419
- Fowler, Martin, Kendall Scott (Contributor) and Ivar Jacobson, *UML Distilled; Applying the Standard Object Modeling Language*, Addison-Wesley, 1997, ISBN 0201325632
- *Designing e-business Solutions for Performance*, white paper by Maggie Archibald and Mike Schlosser, available at:
<http://www.ibm.com/software/developer/library/patterns/performance.html>
- JavaSoft: “*The Java HotSpot Performance Engine Architecture*” white paper, available at: <http://java.sun.com/products/hotspot/whitepaper.html>
- *IBM Application Framework for e-business*: white papers available at:
<http://www.ibm.com/software/ebusiness/>
- Flanagan, David, *JavaScript: The Definitive Guide*, Third Edition, O'Reilly & Associates, Inc., 1998, ISBN 1565923928
- Maruyama, Hiroshi, Kent Tamura and Naohiko Uramoto, *XML and Java: Developing Web Applications*, Addison-Wesley, 1999, ISBN 0201485435
- Flanagan, David, Jim Farley, William Crawford and Kris Magnusson, *Java Enterprise in a Nutshell*, O'Reilly & Associates, Inc. 1999, ISBN 1565924835
- Booch, Grady, *Object-Oriented Analysis and Design with Applications* (Addison-Wesley Object Technology Series), Addison-Wesley, 1994, ISBN 0805353402
- Jacobson, Ivar, *Object-Oriented Software Engineering; A Use Case Driven Approach*, Addison-Wesley, 1992, ISBN 0201544350
- Rumbaugh, James et al, *Object-Oriented Modeling and Design*, Prentice Hall, 1991, ISBN 0136298419
- Nagaratnam, Nataraj et al. 2000. *Security Overview of IBM WebSphere Standard/Advance 3.02*, IBM white paper, available at:
<http://www.ibm.com/software/webservers/appserv/whitepapers.html>
- *Developing Dynamic Web Sites Using the WebSphere Application Server* by Shane Claussen and Mike Conner, available at:
<http://service2.boulder.ibm.com/devcon/news0399/artpage2.htm>

17.1 Referenced Web sites

These Web sites are also relevant as further information sources:

- <http://www.ibm.com/software/developer/web/patterns>. IBM Patterns for e-business site.
- <http://www.ibm.com/software/webservers/appserv/> IBM WebSphere Application Server
- <http://www.as400.ibm.com/WebSphere> IBM WebSphere for AS/400
- <http://www.s390.ibm.com/oe/perform/dgwperf.html> Performance information for the Domino Go Webserver for OS/390
- <http://www.ibm.com/software/ebusiness> IBM's Application Framework for e-business
- <http://www.ecma.ch/stand/ECMA-262.htm> Standard ECMA-262 ECMAScript Language Specification
- <http://www.javasoft.com/products> Java product and API information
- <http://www.w3.org/MarkUp> W3C's home page for HTML
- <http://oss.software.ibm.com/developerworks/opensource/jsp/index.html> JSP Format Bean Library
- <http://www.omg.org> Object Management Group (OMG) home page
- <http://www.rational.com/products/rose/> Rational Rose product information
- <http://www.ibm.com/software/vadd> IBM VisualAge Developer Domain
- (<http://www.cert.org> CERT home page
- <http://www.as400.ibm.com/developer/java/deploy/deployguide.html> Information on deploying Java applications in an AS/400 environment
- <http://www.ibm.com/software/secureway/> IBM Secureway products
- <http://www.ibm.com/developer/features/framework/framework.html> IBM Application Framework for e-business: Web Application Programming Model

How to get IBM Redbooks

This section explains how both customers and IBM employees can find out about IBM Redbooks, redpieces, and CD-ROMs. A form for ordering books and CD-ROMs by fax or e-mail is also provided.

- **Redbooks Web Site** <http://www.redbooks.ibm.com/>

Search for, view, download, or order hardcopy/CD-ROM Redbooks from the Redbooks Web site. Also read redpieces and download additional materials (code samples or diskette/CD-ROM images) from this Redbooks site.

Redpieces are Redbooks in progress; not all Redbooks become redpieces and sometimes just a few chapters will be published this way. The intent is to get the information out much quicker than the formal publishing process allows.

- **E-mail Orders**

Send orders by e-mail including information from the IBM Redbooks fax order form to:

	e-mail address
In United States or Canada	pubscan@us.ibm.com
Outside North America	Contact information is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Telephone Orders**

United States (toll free)	1-800-879-2755
Canada (toll free)	1-800-IBM-4YOU
Outside North America	Country coordinator phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

- **Fax Orders**

United States (toll free)	1-800-445-9269
Canada	1-403-267-4455
Outside North America	Fax phone number is in the "How to Order" section at this site: http://www.elink.ibm.link.ibm.com/pbl/pbl

This information was current at the time of publication, but is continually subject to change. The latest information may be found at the Redbooks Web site.

IBM Intranet for Employees

IBM employees may register for information on workshops, residencies, and Redbooks by accessing the IBM Intranet Web site at <http://w3.itso.ibm.com/> and clicking the ITSO Mailing List button. Look in the Materials repository for workshops, presentations, papers, and Web pages developed and written by the ITSO technical professionals; click the Additional Materials button. Employees may access MyNews at <http://w3.ibm.com/> for redbook, residency, and workshop announcements.

IBM Redbooks fax order form

Please send me the following:

Title	Order Number	Quantity

First name	Last name
------------	-----------

Company

Address

City	Postal code	Country
------	-------------	---------

Telephone number	Telefax number	VAT number
------------------	----------------	------------

<input type="checkbox"/> Invoice to customer number	
---	--

<input type="checkbox"/> Credit card number	
---	--

Credit card expiration date	Card issued to	Signature
-----------------------------	----------------	-----------

We accept American Express, Diners, Eurocard, Master Card, and Visa. Payment by credit card not available in all countries. Signature mandatory for credit card payment.

Index

A

adaptive compilers 61
AFS 51
application server 23

B

base redirector 43, 51
Bean Managed Persistence (BMP) 78
Business Logic (Model) 94, 97, 102, 160, 173, 178
business logic developer 177

C

caching 56, 59
cascading style sheets (CSS) 68, 103
CGI 56, 57, 58
CGI/WSAPI 58
CICS 174, 188
CICS connectors 85
class diagram 158
class model 166
Class Responsibility Collaboration (CRC) 162
clone 25, 29, 45, 47
cloning 25, 34, 45, 47, 287
Command beans 117, 118, 128, 237
Common Connector Framework (CCF) 78
Common Gateway Interface (CGI) 55
component model 166
component view 183
connectors 65, 78, 85
Container Managed Persistence (CMP) 78
cookies 137
CORBA 39, 40, 188

D

Database server node 21
DataSource 252
DCE 117
Demilitarized Zone (DMZ) 20, 24, 32
deployment model 169
DHTML 21, 66, 67, 68, 69, 83
Directory and security services node 21
Directory Management Tool (DMT) 330
DMZ 22, 27, 28, 29, 30, 33, 36, 39, 51, 199, 209, 223, 305

Document Object Model (DOM) 69
Document Type Definition (DTD) 69, 75
domain firewall 22, 24, 26, 29, 31, 33, 35, 38, 40, 41, 49, 50, 52, 306, 309, 310, 322
Domain Name Service (DNS) 21

E

EJBs 78, 79, 86, 98, 102, 173, 194, 199
Enterprise Java beans 30, 45, 65, 77
Enterprise JavaBeans (EJBs) 77, 82, 85, 92
Enterprise Solution Structure (ESS) 1, 3, 19
Enterprise-out 11
eXtensible Stylesheet Language (XSL) 69

F

FastCGI 55
Formatter beans 116

G

Garbage Collection (GC) 60

H

horizontal scalability 25, 27, 29, 31, 33, 35, 38, 40
HTML 20, 56, 58, 65, 67, 68, 83, 84, 86
HTTP 22, 29, 73, 83
HttpServlet 86, 90
HttpServletRequest 87
HttpServletResponse 88
HttpSession 88

I

IBM Global Services methodology 1, 147
IIOP 39, 52, 83, 117, 231
IMS 174, 188
IMS connectors 85
integrated development environment (IDE) 188
interaction controller 94, 100, 102, 104, 144, 159, 161, 173, 178
interaction diagram 163
IPSEC 20
ISDN 41

J

Java applets 66, 70

Java Message Service (JMS) 79
Java Naming and Directory Interface (JNDI) 79
Java Server Pages 20, 22, 65, 73, 74, 82, 86, 90, 95, 102, 199
Java servlets 65, 86
Java Transaction API (JTA) 79
Java Virtual Machine (JVM) 60, 61, 73, 203
JavaBeans 74, 82, 86, 91, 98
JavaScript 66, 67, 68, 69, 70, 86
JDBC 65, 75, 85, 117, 174, 252
JNDI 85, 117, 202
JScript 69
Just-In-Time (JIT) compiler 60, 61

L

LDAP 49, 50, 52, 79, 85, 140, 213, 276, 326
LDAP Data Interchange Format (LDIF) 329
Lightweight Third Party Authentication (LTPA) 140, 214, 278
load balancer 22, 25, 26, 34, 35, 50, 58
Location Service Daemon (LSD) 301
locking 59

M

memory 29, 31, 38, 40, 56
memory leaks 60
model 25, 34, 45
Model-View-Controller (MVC) 65, 93, 94, 98, 102, 104, 159, 173
MQ 83
MQ connectors 85
MQSeries 174, 188

O

OSE Remote 43, 44

P

Page Construction services 85
Page Constructor (View) 94, 97, 102, 173, 178
Pattern Development Kit (PDK) 2, 4
Personal Digital Assistant (PDA) 14, 21, 81, 84, 103
pervasive computing device 21
presentation node 13
processes 29, 31, 38, 40
protocol firewall 22, 25, 27, 33, 36, 41, 49, 50, 52, 307, 309, 319, 322

Public Key Infrastructure (PKI) 21

R

Rational Rose 82, 154, 162, 181, 185
Remote Method Invocation (RMI) 79
Result bean 104, 105, 114, 144
reverse proxy 22, 29, 38, 43, 44
RMI 30, 39, 40, 52, 79, 117
RMI/IIOP 30, 39, 79, 301

S

SAX API 75
scalability 25, 27, 29, 31, 33, 35, 38, 40, 45
script developer 177
segmenting 58
server-side includes (SSI) 56
server-side Java 57
servlet redirector 22, 43, 51, 52, 297
ServletContext 89
servlets 73, 82, 86, 95, 199
session affinity 47
session beans 30
session clustering 139
session management services 85
session persistence 138
session sharing 46
shared file system 22, 26, 35
Single Sign-On (SSO) 142, 219
sockets 29, 31, 38, 40
SQL 60, 75, 76, 187
SQLJ 65, 76
SSL 20, 21, 44, 52, 85, 140
SSO cookie 142
standalone redirector 44, 297
State beans 118, 126
state diagram 166
static compiler 61
symmetric multiprocessors (SMP) 62

T

thin redirector 51
third-tier integration developer 177
threads 29, 31, 38, 40, 55

U

Unified Model Language (UML) 154
User node 21

V

VBScript 69
vertical scalability 25, 27, 29, 31, 33, 38, 40, 45
View bean 104, 106, 113, 115
view developer 176
VisualAge for Java 188, 239
VoiceXML 69

W

Web application server 20, 23, 27, 35, 49, 50, 59, 72, 84
Web application server node 20
Web server redirector 22, 36
WebTV 84
Web-up 11
Wireless Application Protocol (WAP) 68, 69
Wireless Markup Language (WML) 68, 69

X

XML 65, 67, 69, 75, 88, 90, 207, 235
XSL 75

IBM Redbooks review

Your feedback is valued by the Redbook authors. In particular we are interested in situations where a Redbook "made the difference" in a task or problem you encountered. Using one of the following methods, **please review the Redbook, addressing value, subject matter, structure, depth and quality as appropriate.**

- Use the online **Contact us** review redbook form found at <http://www.redbooks.ibm.com/>
- Fax this form to: USA International Access Code + 1 914 432 8264
- Send your comments in an Internet note to redbook@us.ibm.com

Document Number	SG24-5864-00
Redbook Title	Patterns for e-business: User-to-Business Patterns for Topology 1 and 2 using WebSphere Advanced Edition
Review	<div></div> <div></div> <div></div> <div></div> <div></div> <div></div>
What other subjects would you like to see IBM Redbooks address?	<div></div> <div></div> <div></div>
Please rate your overall satisfaction:	<input type="radio"/> Very Good <input type="radio"/> Good <input type="radio"/> Average <input type="radio"/> Poor
Please identify yourself as belonging to one of the following groups:	<input type="radio"/> Customer <input type="radio"/> Business Partner <input type="radio"/> Solution Developer <input type="radio"/> IBM, Lotus or Tivoli Employee <input type="radio"/> None of the above
Your email address: The data you provide here may be used to provide you with information from IBM or our business partners about our products, services or activities.	<input type="radio"/> Please do not use the information collected here for future marketing or promotional contacts or other communications beyond the scope of this transaction.
Questions about IBM's privacy policy?	The following link explains how we protect your personal information. http://www.ibm.com/privacy/yourprivacy/

