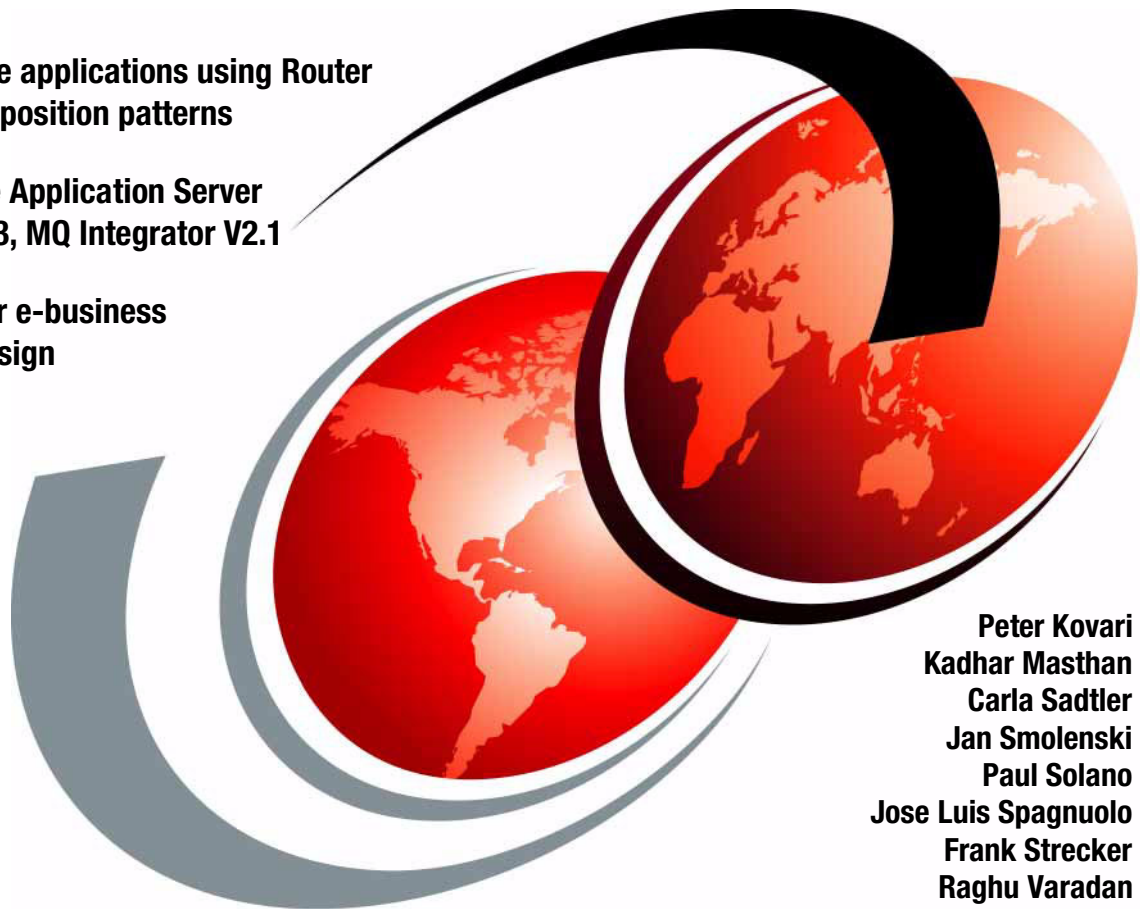


Self-Service Applications using IBM WebSphere V5.0 and IBM MQSeries Integrator

Self-Service applications using Router
and Decomposition patterns

WebSphere Application Server
V5, MQ V5.3, MQ Integrator V2.1

Patterns for e-business
solution design



Peter Kovari
Kadhar Masthan
Carla Sadtler
Jan Smolenski
Paul Solano
Jose Luis Spagnuolo
Frank Strecker
Raghu Varadan



International Technical Support Organization

**Self-Service Applications using IBM WebSphere
V5.0 and IBM MQSeries Integrator**

July 2003

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

First Edition (July 2003)

This edition applies to WebSphere Application Server V5, WebSphere MQ V5.3, WebSphere MQ Integrator V2.1 for use with the Windows 2000 Server, AIX 5L, and Red Hat Linux 7.2.

© Copyright International Business Machines Corporation 2003. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xi
Become a published author	xiv
Comments welcome	xiv
Part 1. Patterns for e-business	1
Chapter 1. Patterns for e-business	3
1.1 The Patterns for e-business layered asset model	5
1.2 How to use the Patterns for e-business	6
1.2.1 Selecting a Business, Integration, or Composite pattern, or a Custom design	7
1.2.2 Selecting Application patterns	12
1.2.3 Review Runtime patterns	13
1.2.4 Review Product mappings	16
1.2.5 Review guidelines and related links	17
1.3 Summary	17
Chapter 2. The Self-Service business pattern	19
2.1 Self-Service applications	20
2.2 Self-Service application patterns	20
2.3 Application patterns used in this book	23
2.3.1 Router pattern	23
2.3.2 Decomposition pattern	27
Chapter 3. Runtime patterns	33
3.1 Nodes	34
3.2 Basic Runtime pattern for the Router pattern	37
3.2.1 Variation 1	39
3.3 Basic Runtime pattern for Decomposition	40
3.3.1 Variation 1	41
3.4 For more information	42
Chapter 4. Product mapping	43
4.1 Runtime product mappings	44
4.2 Product summary	49

Part 2. Guidelines	57
Chapter 5. Technology options	59
5.1 Web client	61
5.1.1 Web browser	62
5.1.2 HTML	63
5.1.3 Dynamic HTML	63
5.1.4 CSS	64
5.1.5 JavaScript	65
5.1.6 Java applets	65
5.1.7 XML (client side)	67
5.1.8 XHTML 1.1 (HTML 4.01)	68
5.1.9 VoiceXML	69
5.1.10 XForms	69
5.1.11 XSLT	69
5.1.12 Mobile clients	70
5.2 Web application server	71
5.2.1 Java servlets	73
5.2.2 JavaServer Pages (JSPs)	74
5.2.3 JavaBeans	74
5.2.4 XML	75
5.2.5 Enterprise JavaBeans	79
5.2.6 Additional enterprise Java APIs	81
5.3 Integration technologies	82
5.3.1 Web services	82
5.3.2 J2EE Connector Architecture	86
5.3.3 Java Message Service	88
5.3.4 Message Oriented Middleware	91
5.3.5 Others	92
5.4 Where to find more information	93
Chapter 6. Application design	95
6.1 Application structure	96
6.1.1 Model-View-Controller design pattern	96
6.1.2 Struts	97
6.1.3 Sample application	99
6.2 EJB design guidelines	105
6.2.1 Local and remote home interfaces	105
6.2.2 Using the Singleton pattern	109
6.2.3 The Facade pattern	110
6.3 JMS design guidelines	112
6.3.1 Message models	112
6.3.2 JMS point-to-point model	114

6.3.3	JMS publish/subscribe model	115
6.3.4	JMS messages	119
6.3.5	Synchronous versus asynchronous design considerations	121
6.3.6	Where to implement message producers and consumers	128
6.3.7	Message-driven beans	129
6.3.8	Managing JMS objects	135
6.3.9	JMS and JNDI	136
6.3.10	Embedded JMS Provider versus WebSphere MQ	137
6.3.11	WebSphere to MQ connection options	138
6.3.12	Best practices for JMS and IBM WebSphere MQ	143
Chapter 7.	Application development	147
7.1	MVC development using the Struts framework	148
7.1.1	Creating a Web diagram	148
7.1.2	Coding Struts elements.	148
7.2	Developing a message-driven bean with WebSphere Studio	152
7.2.1	Message-driven bean implementation	153
7.2.2	Life cycle of a message-driven bean.	154
7.2.3	Creating an MDB using WebSphere Studio	156
7.2.4	Coding the message-driven bean	158
7.3	XML and XSLT development	160
7.3.1	XML as data transfer technology	160
7.3.2	Guidelines for creating an XML message	161
7.3.3	Performing XML transformations	162
7.3.4	Working with XSLTC	163
7.3.5	WebSphere Studio XML support.	165
7.3.6	Using XML JavaBeans	166
Chapter 8.	Developing WebSphere MQ Integrator message flows	169
8.1	What is a broker domain?	170
8.2	Developing message flows	171
8.2.1	Preparations: creating queue managers and defining queues	172
8.2.2	Using the Control Center.	172
8.2.3	Creating message flows	174
Chapter 9.	Security	185
9.1	End-to-end security	186
9.2	Applying security to our Runtime patterns.	188
9.3	Security guidelines	191
9.4	Application security	193
9.5	Messaging security	196
9.5.1	Securing WebSphere MQ resources	200
9.5.2	Securing WebSphere MQ Integrator resources	207
9.6	Security design principles summary	209

Chapter 10. Performance and availability	211
10.1 Introduction	212
10.2 Performance analysis	212
10.3 Performance considerations in messaging	212
10.3.1 Connection pooling	212
10.3.2 Multithreaded programs	214
10.3.3 Persistent versus non-persistent messages	215
10.3.4 One-phase commit optimization	216
10.3.5 Caching WebSphere MQ JMS objects	217
10.3.6 Message-driven beans performance considerations	217
10.4 High availability with WebSphere MQ	218
10.4.1 Overview of WebSphere MQ cluster components	220
10.4.2 WebSphere MQ simplified management	222
Part 3. Implementation	225
Chapter 11. Technical scenarios	227
11.1 Application flow	228
11.2 System setup	229
11.2.1 Products used to prove the scenarios	229
11.2.2 Development environment	230
11.2.3 Runtime environment	231
Chapter 12. Configuring WebSphere	235
12.1 Defining JMS resources to WebSphere	236
12.1.1 Determining the correct scope	236
12.2 Using the embedded JMS server	237
12.2.1 Defining a queue connection factory	237
12.2.2 Defining a queue destination	240
12.2.3 Define the queue for the JMS server	242
12.3 Using WebSphere MQ V5.3	243
12.3.1 Defining a queue connection factory	243
12.3.2 Define a queue destination	247
12.3.3 Define the queue for WebSphere MQ	253
12.4 Deploying message-driven beans in WebSphere V5.0	253
12.5 Testing, logging, debugging	257
Chapter 13. Configuring WebSphere MQ and MQ Integrator	259
13.1 WebSphere MQ objects	260
13.2 WebSphere MQ system management	263
13.2.1 Remote administration	266
13.3 Creating the WebSphere MQ Integrator databases	267
13.4 Creating the WebSphere MQ Integrator Configuration Manager	269
13.4.1 Creating the brokers	272

13.4.2 Transaction behavior.	276
13.5 Testing, logging, debugging	278
Appendix A. Additional material	281
Locating the Web material	281
Using the Web material	282
System requirements for downloading the Web material	282
How to use the Web material	282
13.5.1 Supplier application configuration	284
13.5.2 Running.	285
Abbreviations and acronyms	287
Related publications	289
IBM Redbooks	289
Other resources	289
Referenced Web sites	290
How to get IBM Redbooks	291
IBM Redbooks collections.	291
Index	293

Notices

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

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

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

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

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

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

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

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

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


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

COPYRIGHT LICENSE:

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

Trademarks

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

AIX 5L™	IMS™	SupportPac™
AIX®	Lotus®	Tivoli®
CICS®	MQSeries®	TXSeries™
DB2 Universal Database™	MVS™	VisualAge®
DB2®	OS/390®	VSE/ESA™
eServer™	RACF®	WebSphere®
Encina®	Redbooks(logo)™ 	z/OS™
Everyplace™	Redbooks™	
IBM®	S/390®	
ibm.com®	SOM®	

The following terms are trademarks of other companies:

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

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

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

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

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

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

Preface

This IBM® Redbook introduces the Router and Decomposition application patterns for Self-Service e-business applications. The book discusses the messaging and transactional capabilities of an application and is a valuable source for IT architects, IT specialists, application designers, application developers, system administrators, and consultants.

Part 1, “Patterns for e-business”, introduces the Patterns for e-business concept, focusing particularly on the Self-Service business pattern and the Router and Decomposition application patterns.

Part 2, “Guidelines”, provides guidelines for messaging and transactional applications, including application design and development, and some of the non-functional requirements for such applications, including security and system management and performance.

Part 3, “Implementation” provides information on how to set up and configure both the development and runtime environments for the sample application discussed in this book.

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 Team (left to right, top: Frank Strecker, Raghu Varadan, Jan Smolenski, Paul Solano; front: Jose Luis Spagnuolo, Peter Kovari)

Peter Kovari is a WebSphere® Specialist at the International Technical Support Organization, Raleigh Center. He writes extensively about all areas of WebSphere. His areas of expertise include e-business, e-commerce, security, Internet technologies and mobile computing. Before joining the ITSO, he worked as an IT Specialist for IBM in Hungary.

Kadhar Masthan is an e-business technical consultant in Cognizant Technology Solutions, India, a major IT solutions provider and a business partner of IBM. He has four years of experience in the IT industry. He holds a degree in computer engineering from the Central Institute of Technology, Chennai (India). He is also pursuing a Masters of Science degree at Bits Pilani, India. His areas of expertise include Web applications development using WebSphere, IBM DB2®, WebSphere MQ and IBM VisualAge® For Java. He writes articles about WebSphere architecture for Cognizant. He has written about the WebSphere platform and developing applications using WebSphere.

Carla Sadtler is a WebSphere Specialist at the International Technical Support Organization, Raleigh Center. She writes extensively in the WebSphere and Patterns for e-business areas. Before joining the ITSO in 1985, 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.

Jan Smolenski (*Jania*) is an Advisory Software Engineer and Certified MQ Solutions Expert/Specialist at the IBM Developer Technical Support Center in Dallas, Texas. He has over 20 years of experience in the IT industry, ranging from large systems to PC workstations. He holds a degree in Computer Science and Mathematics from the University of Warsaw, Poland. His areas of expertise include transaction processing, messaging systems and Web development.

Paul Solano is an IT Solution Architect working for GBM de Costa Rica, an IBM Alliance company located in San José, Costa Rica. He has five years of experience in Web-based application development and integration. He holds a degree in Computer Science from the Instituto Tecnológico de Costa Rica. His areas of expertise includes enterprise application design and development using the WebSphere family of products.

Jose Luis Spagnuolo is a Certified Consulting IT Architect in IBM Brazil. He has 20 years of experience in the IT industry and has worked with large systems, artificial intelligence, networking, client/server and e-business. He holds a degree in Technology of Data Processing and a degree in Business Administration from Mackenzie University. He is also a member of the IBM IT Architect Certification Board. Before joining IBM Brazil, he worked as an IT Specialist at the IBM International Education Centre in La Hulpe, Belgium.

Frank Strecker is an IT Specialist in IBM Germany. He holds a degree in Legal Sciences from Hannover University and has four years of experience in the IT industry. His areas of expertise include e-business and messaging systems.

Raghu Varadan is a Sr. Architect with the Architecture Center of Excellence in IBM Global Services Americas. He has over 14 years of experience in software consulting, demonstrating the broad business application of technology, strategic technology planning, and systems architecture. He has extensive experience in developing large-scale, complex enterprise-wide architectures, OO, and client-server architectures and development, and has been involved in both business and technical aspects of the full life cycle of software development. He has written publications on architecture, methodologies, and emerging technologies, and presented at public and internal conferences.

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

Gail Christensen
Mark Endrei

Michele Galic
Margaret Ticknor
Jeanne Tucker
International Technical Support Organization, Raleigh Center

Thanks to the following IBM employees:

Jonathan Adams, IBM UK, Software Group Technical Strategy
Greg Behrend, IBM US, Software Services for WebSphere
Ligia Fumi Tsubouchi, IBM Brazil

Become a published author

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

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

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

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 Building 662

P.O. Box 12195
Research Triangle Park, NC 27709-2195



Part 1

Patterns for e-business



Patterns for e-business

This IBM Redbook is part of the Patterns for e-business series. In this introductory chapter, we provide an overview of how IT architects can work effectively with the Patterns for e-business.

The role of the IT architect is to evaluate business problems and to build solutions to solve them. To do this, the architect begins by gathering input on the problem, creating an outline of the desired solution, and taking into account any special considerations or requirements that need to be factored into that solution. The architect then takes this input and designs the solution. This solution can include one or more computer applications that address the business problems by supplying the necessary business functions.

To enable the architect to do this better each time, we need to capture and reuse the experience of these IT architects in such a way that future engagements can be made simpler and faster. We do this by taking these experiences and using them to build a repository of assets that provides a source from which architects can reuse this experience to build future solutions, using proven assets. This reuse saves time, money and effort and in the process helps ensure delivery of a solid, properly architected solution.

The IBM Patterns for e-business helps facilitate this reuse of assets. Their purpose is to capture and publish e-business artifacts that have been used, tested, and proven. The information captured by them is assumed to fit the majority, or 80/20, situation.

The IBM Patterns for e-business are further augmented with guidelines and related links for their better use.

The layers of patterns plus their associated links and guidelines allow the architect to start with a problem and a vision for the solution, and then find a pattern that fits that vision. Then, by drilling down using the patterns process, the architect can further define the additional functional pieces that the application will need to succeed. Finally, he can build the application using coding techniques outlined in the associated guidelines.

1.1 The Patterns for e-business layered asset model

The Patterns for e-business approach enables architects to implement successful e-business solutions through the re-use of components and solution elements from proven successful experiences. The patterns approach is based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in a way that each level of detail builds on the last. These assets include the following.

- ▶ Business patterns that identify the interaction between users, businesses, and data.
- ▶ Integration patterns that tie multiple Business patterns together when a solution cannot be provided based on a single Business pattern.
- ▶ Composite patterns that represent commonly occurring combinations of Business patterns and Integration patterns.
- ▶ Application patterns that provide a conceptual layout describing how the application components and data within a Business pattern or Integration pattern interact.
- ▶ Runtime patterns that define the logical middleware structure supporting an Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.
- ▶ Product mappings that identify proven and tested software implementations for each Runtime pattern.
- ▶ Best-practice guidelines for design, development, deployment, and management of e-business applications.

These assets and their relation to each other are shown in Figure 1-1 on page 6.

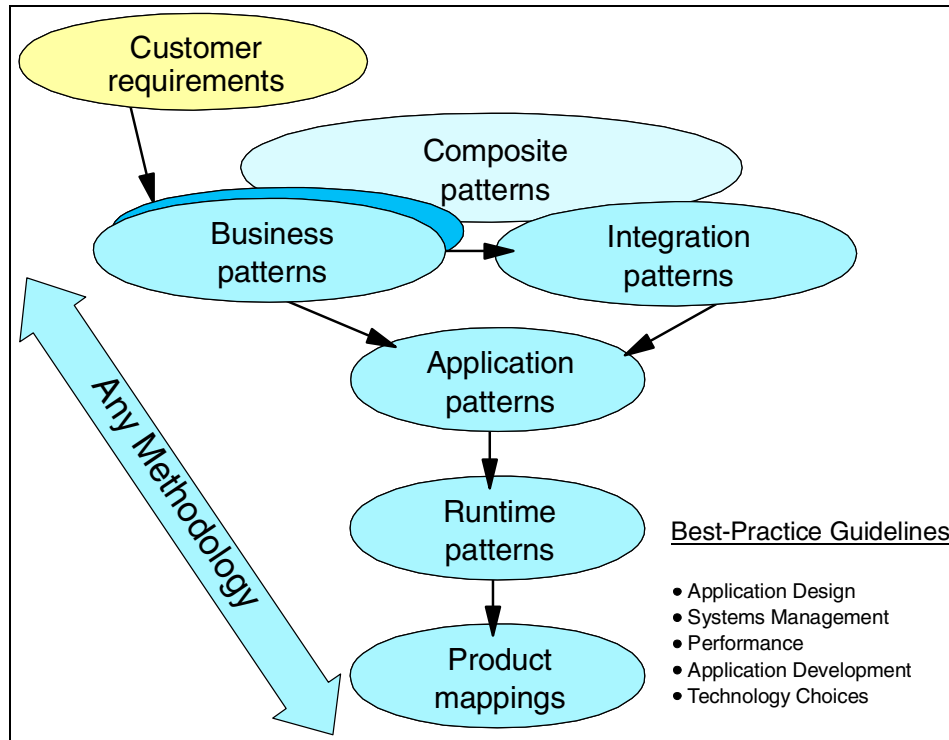


Figure 1-1 The Patterns for e-business layered asset model

Patterns for e-business Web site

The Patterns Web site provides an easy way of navigating top down through the layered Patterns' assets in order to determine the preferred reusable assets for an engagement.

For easy reference to Patterns for e-business refer to the Patterns for e-business Web site at:

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

1.2 How to use the Patterns for e-business

As described in the last section, the Patterns for e-business are a layered structure where each layer builds detail on the last. At the highest layer are Business patterns. These describe the entities involved in the e-business solution.

Composite patterns appear in the hierarchy shown in Figure 1-1 on page 6 above the Business patterns. However, Composite patterns are made up of a number of individual Business patterns, and at least one Integration pattern. In this section, we discuss how to use the layered structure of Patterns for e-business assets.

1.2.1 Selecting a Business, Integration, or Composite pattern, or a Custom design

When faced with the challenge of designing a solution for a business problem, the first step is to take a high-level view of the goals you are trying to achieve. A proposed business scenario should be described and each element should be matched to an appropriate IBM Pattern for e-business. You may find, for example, that the total solution requires multiple Business and Integration patterns, or that it fits into a Composite pattern or Custom design.

For example, suppose an insurance company wants to reduce the amount of time and money spent on call centers that handle customer inquiries. If customers are allowed to view their policy information and to request changes online, they will be able to cut back significantly on the resources spent handling these things over the phone. The objective is to allow policyholders to view their policy information stored in legacy databases.

The Self-Service business pattern fits this scenario perfectly. It is meant to be used in situations where users need direct access to business applications and data. Let's take a look at the available Business patterns.

Business patterns

A Business pattern describes the relationship between the users, the business organizations or applications, and the data to be accessed.

There are four primary Business patterns, explained in Figure 1-2:

Business Patterns	Description	Examples
Self-Service (User-to-Business)	Applications where users interact with a business via the Internet or intranet	Simple Web site applications
Information Aggregation (User-to-Data)	Applications where users can extract useful information from large volumes of data, text, images, etc.	Business intelligence, knowledge management, Web crawlers
Collaboration (User-to-User)	Applications where the Internet supports collaborative work between users	E-mail, community, chat, video conferencing, etc.
Extended Enterprise (Business-to-Business)	Applications that link two or more business processes across separate enterprises	EDI, supply chain management, etc.

Figure 1-2 The four primary Business patterns

It would be very convenient if all problems fit nicely into these four slots, but reality says that things will often be more complicated. The patterns assume that most problems, when broken down into their most basic components, will fit more than one of these patterns. When a problem requires multiple Business patterns, the Patterns for e-business provide additional patterns in the form of Integration patterns.

Integration patterns

Integration patterns allow us to tie together multiple Business patterns to solve a business problem. The Integration patterns are outlined in Figure 1-3 on page 9.

Integration Patterns	Description	Examples
Access Integration	Integration of a number of services through a common entry point	Portals
Application Integration	Integration of multiple applications and data sources without the user directly invoking them	Message brokers, workflow managers

Figure 1-3 Integration patterns

These Business and Integration patterns can be combined to implement installation-specific business solutions. We call this a Custom design.

Custom design

We can represent the use of a Custom design to address a business problem through an iconic representation, as shown in Figure 1-4.

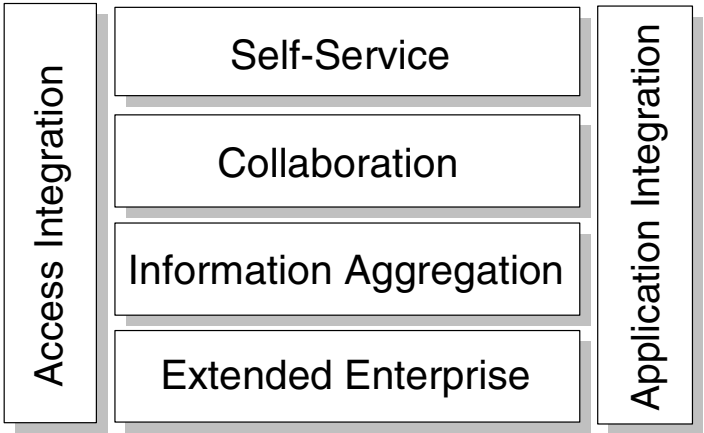


Figure 1-4 Patterns representing a Custom design

If any of the Business or Integration patterns are not used in a Custom design, we can show that using the blocks which are lighter than the others. For example, Figure 1-5 on page 10 shows a Custom design that does not have a Collaboration business pattern or an Extended Enterprise business pattern for a business problem.

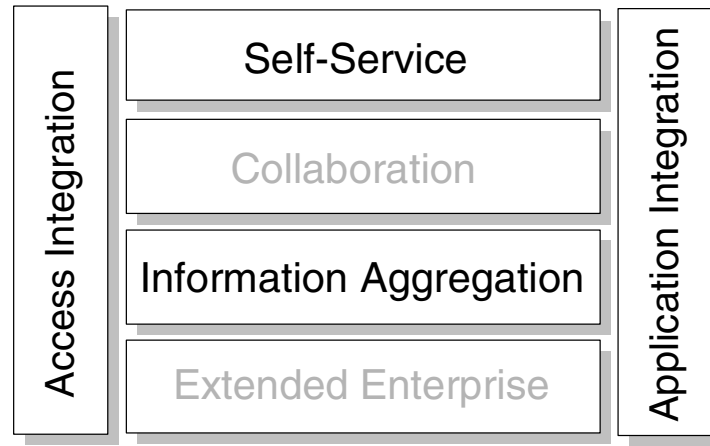


Figure 1-5 Custom design with Self-Service, Information Aggregation, Access Integration and Application Integration

A Custom design may also be a Composite pattern if it recurs many times across domains with similar business problems. For example, the iconic view of a Custom design in Figure 1-5 can also describe a Sell-Side Hub composite pattern.

Composite patterns

Several common uses of Business and Integration patterns have been identified and formalized into Composite patterns. The identified Composite patterns are shown in Figure 1-6 on page 11.

Composite Patterns	Description	Examples
Electronic Commerce	User-to-Online-Buying	<ul style="list-style-type: none"> • www.macys.com • www.amazon.com
Portal	Typically designed to aggregate multiple information sources and applications to provide uniform, seamless, and personalized access for its users.	<ul style="list-style-type: none"> • Enterprise Intranet portal providing self-service functions such as payroll, benefits, and travel expenses. • Collaboration providers who provide services such as e-mail or instant messaging.
Account Access	Provide customers with around-the-clock account access to their account information.	<ul style="list-style-type: none"> • Online brokerage trading apps. • Telephone company account manager functions. • Bank, credit card and insurance company online apps.
Trading Exchange	Allows buyers and sellers to trade goods and services on a public site.	<ul style="list-style-type: none"> • Buyer's side - interaction between buyer's procurement system and commerce functions of e-Marketplace. • Seller's side - interaction between the procurement functions of the e-Marketplace and its suppliers.
Sell-Side Hub (Supplier)	The seller owns the e-Marketplace and uses it as a vehicle to sell goods and services on the Web.	www.carmax.com (car purchase)
Buy-Side Hub (Purchaser)	The buyer of the goods owns the e-Marketplace and uses it as a vehicle to leverage the buying or procurement budget in soliciting the best deals for goods and services from prospective sellers across the Web.	www.wre.org (WorldWide Retail Exchange)

Figure 1-6 Composite patterns

The makeup of these patterns is variable in that there will be basic patterns present for each type, but the Composite can easily be extended to meet additional criteria. For more information on Composite patterns, refer to *Patterns for e-business: A Strategy for Reuse* by Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos.

1.2.2 Selecting Application patterns

Once the Business pattern is identified, the next step is to define the high-level logical components that make up the solution and how these components interact. This is known as the Application pattern. A Business pattern will usually have multiple possible Application patterns. An Application pattern may have logical components that describe a presentation tier for interacting with users, an application tier, and a back-end application tier.

Application patterns break the application down into the most basic conceptual components, identifying the goal of the application. In our example, the application falls into the Self-Service business pattern and the goal is to build a simple application that allows users to access back-end information. The Application pattern shown in Figure 1-7 fulfills this requirement.

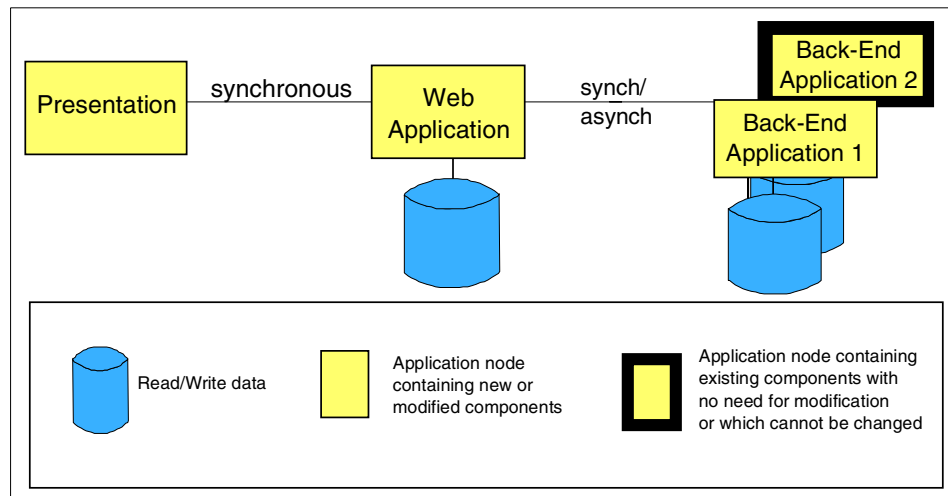


Figure 1-7 Self-Service::Directly Integrated Single Channel

The Application pattern shown consists of a presentation tier that handles the request/response to the user. The application tier represents the component that handles access to the back-end applications and data. The multiple application boxes on the right represent the back-end applications that contain the business data. The type of communication is specified as synchronous (one request/one response, then next request/response) or asynchronous (multiple requests and responses intermixed).

Suppose that the situation is a little more complicated than that. Let's say that the automobile policies and the homeowner policies are kept in two separate and dissimilar databases. The user request would actually need data from multiple, disparate back-end systems. In this case, there is a need to break the request

down into multiple requests (decompose the request) to be sent to the two different back-end databases, then to gather the information sent back from the requests, and then put this information into the form of a response (recompose). In this case, the Application pattern shown in Figure 1-8 would be more appropriate.

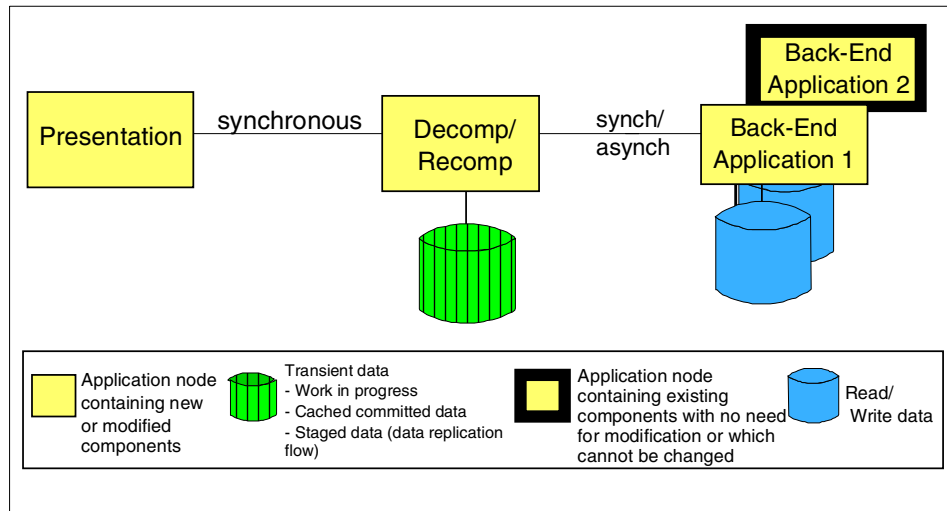


Figure 1-8 Self-Service::Decomposition

This Application pattern extends the idea of the application tier that accesses the back-end data by adding decomposition and recomposition capabilities.

1.2.3 Review Runtime patterns

The Application pattern can be further refined with more explicit functions to be performed. Each function is associated with a runtime node. In reality, these functions, or nodes, can exist on separate physical machines or may co-exist on the same machine. In the Runtime pattern, this is not relevant. The focus is on the logical nodes required and their placement in the overall network structure.

As an example, let's assume that our customer has determined that his solution fits into the Self-Service business pattern and that the Directly Integrated Single Channel pattern is the most descriptive of the situation. The next step is to determine the Runtime pattern that is most appropriate for his situation.

The customer knows that he will have users on the Internet accessing his business data and will therefore require a measure of security. Security can be implemented at various layers of the application, but the first line of defense is almost always one or more firewalls that define who and what can cross the physical network boundaries into his company network.

The customer also needs to determine the functional nodes required to implement the application and security measures. The Runtime pattern shown in Figure 1-9 is one of his options.

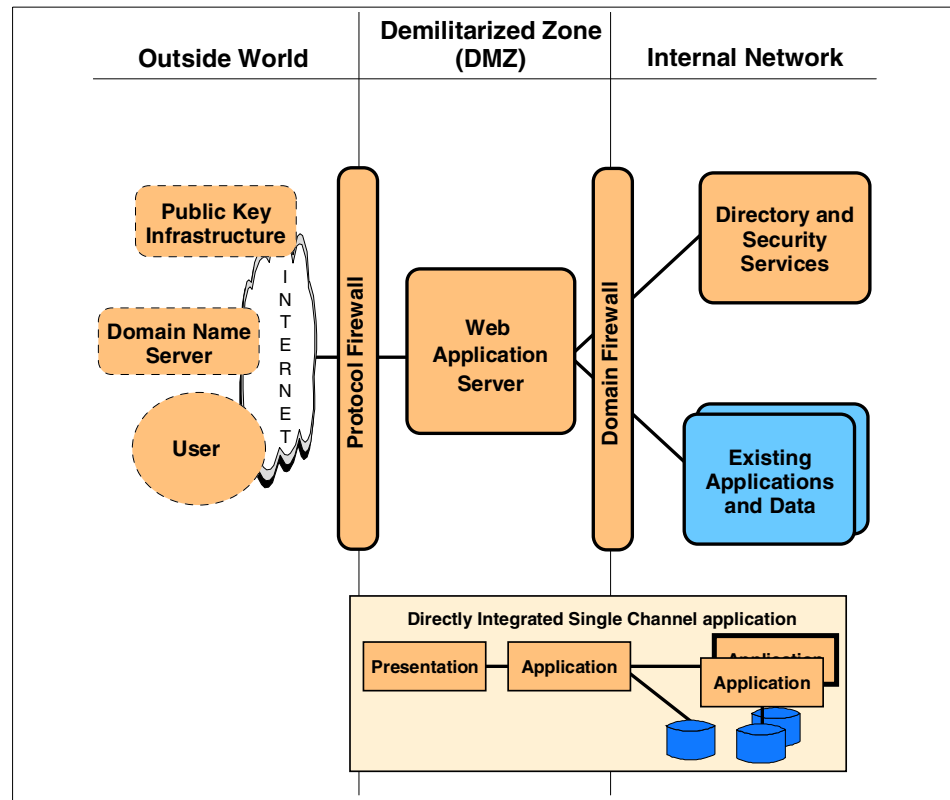


Figure 1-9 Directly Integrated Single Channel application pattern::Runtime pattern

By overlaying the Application pattern on the Runtime pattern, you can see the roles that each functional node will fulfill in the application. The presentation and application tiers will be implemented with a Web application server, which combines the functions of an HTTP server and an application server. It handles both static and dynamic Web pages.

Application security is handled by the Web application server through the use of a common central directory and security services node.

A characteristic that makes this Runtime pattern different from others is the placement of the Web application server between the two firewalls. The Runtime pattern shown in Figure 1-10 is a variation on this. It splits the Web application server into two functional nodes by separating the HTTP server function from the application server. The HTTP server (Web server redirector) will serve static Web pages and redirect other requests to the application server. It moves the application server function behind the second firewall, adding further security.

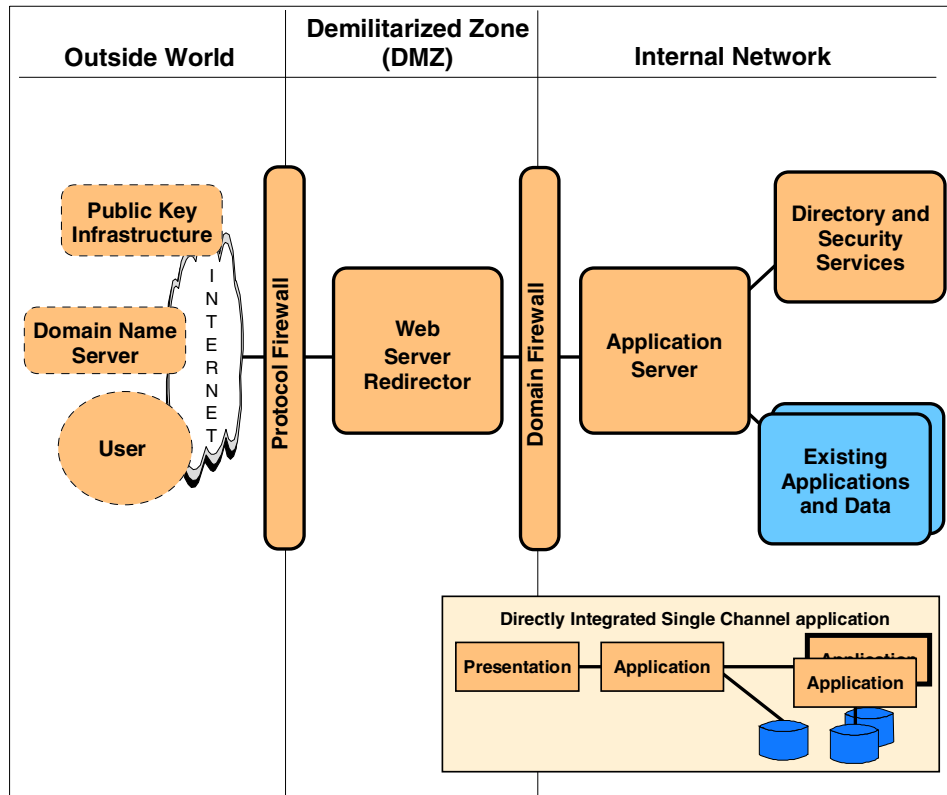


Figure 1-10 Directly Integrated Single Channel application pattern::Runtime pattern: Variation 1

These are just two examples of the possible Runtime patterns available. Each Application pattern will have one or more Runtime patterns defined. These can be modified to suit the customer's needs. For example, the customer may want to add a load-balancing function and multiple application servers.

1.2.4 Review Product mappings

The last step in defining the network structure for the application is to correlate real products with one or more runtime nodes. The Patterns Web site shows each Runtime pattern with products that have been tested in that capacity. The Product mappings are oriented toward a particular platform, though it is more likely that the customer will have a variety of platforms involved in the network. In this case, it is simply a matter of mix and match.

For example, the runtime variation in Figure 1-10 on page 15 could be implemented using the product set depicted in Figure 1-11.

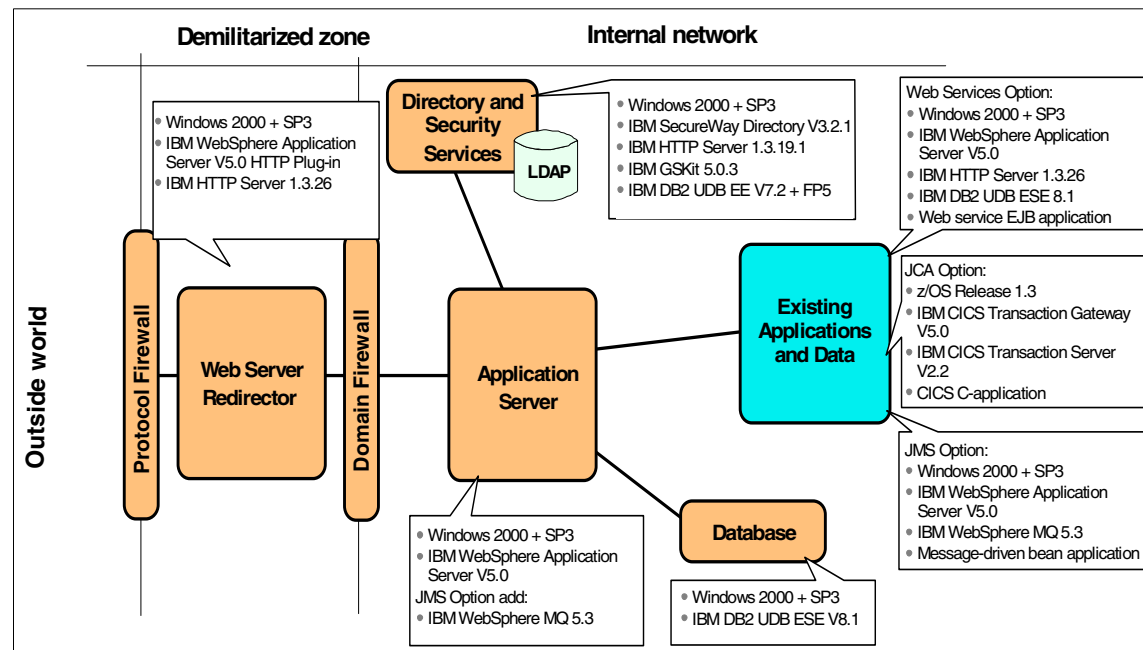


Figure 1-11 Directly Integrated Single Channel application pattern: Windows 2000 product mapping

1.2.5 Review guidelines and related links

The Application patterns, Runtime patterns, and Product mappings are intended to guide you in defining the application requirements and the network layout. The actual application development has not been addressed yet. The Patterns Web site provides guidelines for each Application pattern, including techniques for developing, implementing, and managing the application based on the following:

- ▶ Design guidelines instruct you on tips and techniques for designing the applications.
- ▶ Development guidelines take you through the process of building the application, from the requirements phase all the way through the testing and rollout phases.
- ▶ System management guidelines address the day-to-day operational concerns, including security, backup and recovery, application management, etc.
- ▶ Performance guidelines give information on how to improve the application and system performance.

1.3 Summary

The IBM Patterns for e-business are a collective set of proven architectures. This repository of assets can be used by companies to facilitate the development of Web-based applications. They help an organization understand and analyze complex business problems and break them down into smaller, more manageable functions that can then be implemented.



The Self-Service business pattern

Businesses have traditionally invested a lot of resources into making information available to customers, vendors, and employees. These resources have taken the form of call centers, mailings, etc. They have also maintained information about their customers in the form of customer profiles. Updates to these profiles were usually handled over the phone or by mail.

The concept of self-service puts this information at the fingertips of the customers through a user interface, whether that interface is a Web site, a personal digital assistant (PDA), or some other client interface. An e-business application makes the information accessible to the right audience in an easy-to-access manner, thus reducing the need for human interaction and increasing user satisfaction.

2.1 Self-Service applications

Key elements of an application that provides self-service for a customer would include clear navigational directions, extended search capabilities, and useful links. A popular aspect is a direct link to the online representatives who can answer questions and offer a human interface if needed.

The following are examples of self-service applications:

- ▶ An insurance company makes policy information available to users and allows them to apply for a policy online.
- ▶ A mortgage company publishes information about its loan policies and load rates online. Customers can view their current mortgage information, change their payment options, or apply for a mortgage online.
- ▶ A scientific organization makes research papers available to interested users by putting the papers online.
- ▶ A bank allows customers to access their accounts and pay bills online.
- ▶ A well-known and respected group of technical writers makes its work available online. The group recruits technical participants for its projects by listing the upcoming projects online and allowing possible participants to apply online.
- ▶ A company allows its employees to view current human resource policies online. Employees can change their medical plan, tax withholding information, stock purchase plan, etc., online without having to call the human resources office.

2.2 Self-Service application patterns

As you can see in Figure 2-1 on page 21, the Self-Service business pattern covers a wide range of uses. Applications of this pattern can range from the very simple function of allowing users to view data built explicitly for one purpose, to taking requests from users, decomposing them into multiple requests to be sent to multiple, disparate data sources, personalizing the information, and recomposing it into a response for the user. For this reason, there are currently seven defined Application patterns that fit this range of function. We summarize these for you here. More detailed information can be found in *Patterns for e-business: A Strategy for Reuse*, by Jonathan Adams, Srinivas Koushik, Guru Vasudeva, and George Galambos.

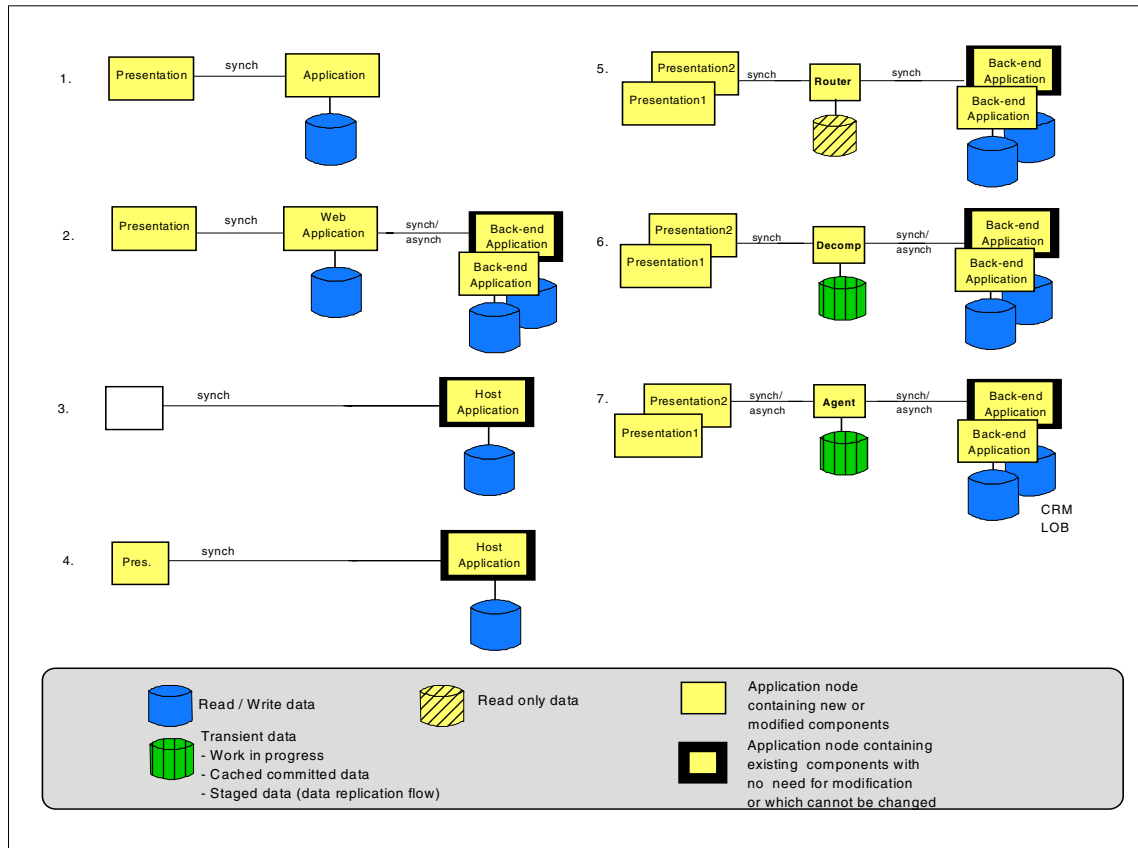


Figure 2-1 Self-Service application patterns

1. **Stand-alone Single Channel application pattern:** Provides for stand-alone applications that have no need for integration with existing applications or data. It assumes one delivery channel, most likely a Web client, although it could be something else. It consists of a presentation tier that handles all aspects of the user interface, and an application tier that contains the business logic to access data from a local database. The communication between the two tiers is synchronous. The presentation tier passes a request from the user to the business logic in the Web application tier. The request is handled and a response sent back to the presentation tier for delivery to the user.
2. **Directly Integrated Single Channel application pattern:** Provides point-to-point connectivity between the user and existing back-end applications. As with the Stand-alone Single Channel application pattern, it assumes one delivery channel and the user interface is handled by the presentation tier. The business logic can reside in the Web application tier

and in the back-end application. The Web application tier has access to local data that exists primarily as a result of this application, for example, customer profile information or cached data. It is also responsible for accessing one or more back-end applications. The back-end applications contain business logic and are responsible for accessing the existing back-end data. The communication between the presentation tier and Web application tier is synchronous. The communication between the Web application tier and the back-end can be either synchronous or asynchronous, depending on the characteristics and capabilities of the back-end application.

3. **As-is Host application pattern:** Provides simple direct access to existing host applications. The application is unchanged, but the user access is translated from green-screen type access to Web browser-based access. This is very quickly implemented but does nothing to change the appearance of the application to the user. The business logic and presentation are both handled by the back-end host. Because the interface is still host driven, this is more suited to an intranet solution where employees are familiar with the application.
4. **Customized Presentation to Host application pattern:** This is one step up from the As-is Host pattern. The back-end host application remains unchanged, but a Web application now translates the presentation from the back-end host application into a more user-friendly, graphical view. The back-end host application is not aware of this translation.
5. **Router application pattern:** The Router application pattern provides intelligent routing from multiple channels to multiple back-end applications using a hub-and-spoke architecture. The interaction between the user and the back-end application is a one-to-one relation, meaning the user interacts with applications one at a time. The router maintains the connections to the back-end applications and pools connections when appropriate, but there is no true integration of the applications themselves. The router can use a read-only database, most probably to look up routing information. The primary business logic still resides in the back-end application tier.

This pattern assumes that the users are accessing the applications from a variety of client types such as Web browsers, voice response units (VRUs), or kiosks. The Router application pattern provides a common interface for accessing multiple back-end applications and acts as an intermediary between them and the delivery channels. In doing this, the Router application pattern may use elements of the Integration patterns.

6. **Decomposition application pattern:** The Decomposition application pattern expands on the Router application pattern, providing all the features and functions of that pattern and adding recomposition/decomposition capability. It provides the ability to take a user request and decompose it into multiple requests to be routed to multiple back-end applications. The responses are recomposed into a single response for the user. This moves some of the

business logic into the decomposition tier, but the primary business logic still resides in the back-end application tier.

7. **Agent application pattern:** The Agent pattern includes the functions of the decomposition tier, plus it incorporates personalization into the application to provide a customer-centric view. The agent tier collects information about the user, either from monitoring their habits or from information stored in a CRM. It uses this information to customize the view presented to the user and can perform cross-selling functions by pushing offers to the user.

2.3 Application patterns used in this book

In this book we used the following two application patterns:

- ▶ Router application pattern
- ▶ Decomposition application pattern

2.3.1 Router pattern

The Router application pattern provides a structure for applications that require the intelligent routing of requests from multiple delivery channels to one of multiple back-end applications.

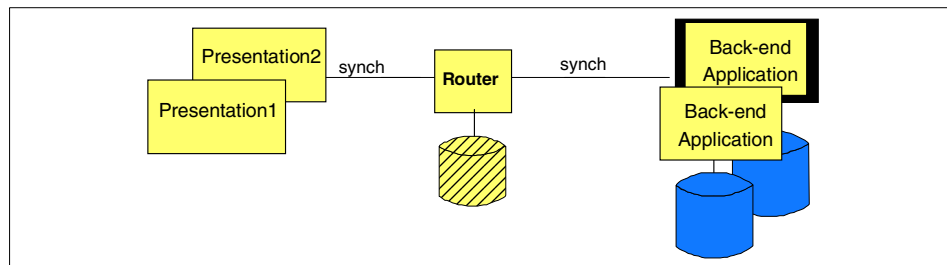


Figure 2-2 Self-Service Router application pattern

Business and IT drivers

- ▶ Reduce the latency of business events
- ▶ Adapt easily during mergers and acquisitions
- ▶ Integrate across multiple delivery channels
- ▶ Minimize total cost of ownership (TCO)
- ▶ Leverage existing skills
- ▶ Leverage legacy investment
- ▶ Integrate back-end applications
- ▶ Minimize enterprise complexity
- ▶ Be maintainable

► Be scalable

The primary business driver for choosing the Router application pattern is to support seamless integration across multiple delivery channels. In this digital economy, users demand universal access to information. To satisfy this demand, many corporations support multiple delivery channels including the Internet, voice recognition units, kiosks, call center applications, and so on. Users expect to retrieve the same information irrespective of the delivery channel they use to access the information. For example, it is important for a discount brokerage firm to ensure that users can retrieve trade execution status consistently either through a Web site or through a voice recognition unit. At the same time, these organizations have multiple back-end applications. For example, due to different tax requirements, discount brokerage firms often maintain IRA (individual retirement account) and regular investment accounts on different back-end applications. Because of this, many channels have a need to access the information from multiple back-end applications.

Applications, over a period of time, evolve either to take advantage of new technological breakthroughs or to accommodate a changing business environment. Ideally such changes to one application should be isolated from another. In other words, if a back-end application is replaced with a new system to take advantage of a new technology, that replacement should not result in significant changes to all the delivery channels that access that back-end applications. At the same time a business decision to support a new delivery channel such as wireless PDAs should not require major changes to back-end applications. Such extensibility is especially important for organizations that are in a highly volatile business environment and have plans for mergers and acquisitions. The Router application is ideally suited for such organizations.

The primary IT driver for choosing this Application pattern is to minimize enterprise wide complexity and reduce the total cost of ownership by using a hub-and-spoke architecture instead of a point-to-point architecture between delivery channels and back-end applications

Solution

As shown in Figure 2-2 on page 23, the Router application pattern is divided into at least three logical tiers: presentation, router, and back-end application.

- The presentation tiers of this Application pattern can support many different presentation styles, including the Internet, call centers, kiosks, and voice recognition units.
- The router tier receives requests from multiple presentation components and intelligently routes them to the appropriate back-end transactions. In doing so, this tier may use a read-only database to look up routing rules. In addition, the router may be responsible for message transformation, protocol conversion,

the management of different levels of security, and session concentration. In most cases the router tier implements minimal business logic. This routing capability can also be used for routing requests from one of the back-end systems to the other.

- The majority of the business logic for this Application pattern is concentrated in the back-end application tier.

A router providing protocol conversion can isolate back-end transactions from the details of delivery channel-specific protocols. For example, the Internet application would typically use HTTP for communication between the browser and the server-side portion of the presentation tier. This tier, in turn, might use RMI or IIOP to communicate with the router. In contrast, call center applications might use RMI or IIOP to send requests directly to the router. The router tier converts these protocol-specific requests into protocol-independent requests before invoking the back-end transactions. This approach enables future support of new types of delivery channels, such as wireless PDAs, without making changes to the back-end applications.

Different delivery channels require different levels of security. Call center application users are usually allowed to see certain details about customer accounts that are not shown to customers when they access their accounts through, for instance, the self-service Web channel. These different levels of security are tightly coupled with the requirements of the delivery channel. Because of this, the back-end application should be isolated from these details. The router tier is ideally suited for making such security decisions.

In order to achieve high scalability and superior performance, session management and session concentration can be done on the router tier. The router tier is thus responsible for establishing a few back-end connections and pooling them for thousands of users originating from multiple delivery channels, when the back-end systems' security capabilities permit it.

The requester interaction between the presentation and router tier is synchronous, as is the requester interaction between the router tier and the back-end application. As a result, if the back-end system is not available, the business function supported by that back-end application would be unavailable across all delivery channels.

Bear in mind that a synchronous (or blocking) request at the Application pattern level can be carried over either a synchronous or an asynchronous communications protocol at the Runtime pattern level.

As described later in the Product mapping portion of this solution design, Web application servers such as WebSphere Application Server and message

brokers such as WebSphere MQ Integrator are often used to implement this Application pattern.

Guidelines for use

The availability, scalability, and performance of the delivery channels in this Application pattern are heavily dependent on the availability, scalability, and performance of the back-end applications. Because of this, robust transaction processing systems should be used to implement the back-end applications. This Application pattern can be used to Web-enable either an existing or a new transaction processing system.

While developing new back-end applications or changing existing ones, transactions should be defined to be channel-independent and reusable. This will make it easier to quickly extend such transactions to newer channels such as wireless PDA devices.

Benefits

- ▶ By using routers to manage session concentration and robust transaction processing systems for implementing business logic, this Application pattern delivers high scalability and superior performance. This is often the target architecture for most of the current high-volume e-business sites.
- ▶ This Application pattern isolates back-end implementation details from the delivery channels and the delivery channel implementation details from back-end applications. This loosely coupled architecture makes it easy to change, replace, or add back-end applications and delivery channels without heavily affecting other applications in the architecture.
- ▶ This loosely coupled architecture also increases maintainability and reduces the total cost of ownership.
- ▶ It uses the same back-end transactions across multiple delivery channels, which avoids duplication of the same business logic on multiple systems. As a result, changes to business logic can be made in one system. This increases the maintainability of the overall system.

Limitations

- ▶ The availability of certain business functions is heavily dependent on the availability of the back-end applications. Currently many organizations have transaction processing systems that are only available for a limited amount of time every day. During the rest of the time, they are brought down for batch processing, backup, and maintenance activities. Such systems will have to be enhanced for 24x7 availability to support delivery channels offering Self-Service capabilities on the Web. These enhancements to existing systems can be expensive and time consuming.

- The focus is primarily on providing access to back-end applications from multiple delivery channels. Most back-end applications are product-specific and the Router application pattern does not address how to move beyond a product-specific view and into a holistic customer-centric view. Under such circumstances, one should consider more advanced Application patterns such as the Decomposition (discussed below) or Agent application pattern.

Putting the Application pattern to use

With presence in four continents, the ITSO Vendor Corporation (IV Corp) can provide a wide range of products and services related to painting products. From manufacturing, packaging, warehousing and shipping, it offers all of these to its customers. Obviously these lines of business activities are very different in nature and are supported by different back-end applications. Delivery channels operated by the company include Internet self-service, kiosks, voice recognition units, and 24-hour call centers.

Currently, different channels use point-to-point connections with back-end applications. Some of them also duplicate the data, which results in inconsistency of information across different channels at a given point in time. Fortunately, most of the product-related back-end applications are robust transaction processing systems that can be made available on a 24x7 basis.

The company wants to provide consistent access to all product information through all channels. In addition, the target architecture must be highly scalable, highly available, and support superior response time. To achieve these goals, the company chooses the Router application pattern.

2.3.2 Decomposition pattern

The Decomposition application pattern extends the hub-and-spoke architecture provided by the Router application pattern. It decomposes a single, compound request from a client into several, simpler requests and intelligently routes them to multiple back-end applications. Typically the responses from these multiple back-end applications are recomposed into a single response and sent back to the client.

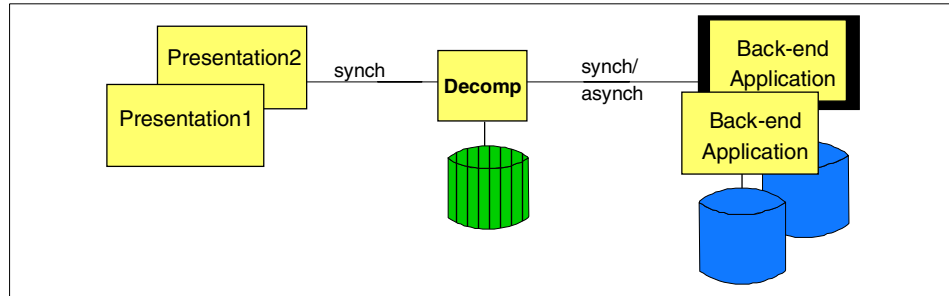


Figure 2-3 Self-Service Decomposition application pattern

Business and IT drivers

- ▶ Improve organizational efficiency
- ▶ Reduce the latency of business events
- ▶ Adapt easily during mergers and acquisitions
- ▶ Integrate across multiple delivery channels
- ▶ Unify customer view across lines of businesses (LOB)
- ▶ Minimize total cost of ownership (TCO)
- ▶ Leverage existing skills
- ▶ Leverage legacy investment
- ▶ Integrate back-end applications
- ▶ Minimize enterprise complexity
- ▶ Be maintainable
- ▶ Be scalable

All business and IT drivers listed under the Router application pattern apply here as well. Additional drivers relate to the fact that many organizations have back-end applications that are focused on certain product lines. For example, insurance companies use different systems for supporting health insurance policies and life insurance policies. Such product-specific silos evolved out of necessity, since the business logic and data requirements of different products were vastly different. For the same reason, many companies plan to continue to use separate systems for separate product lines.

These same companies, however, want to provide a unified customer view when customers visit the self-service Web sites or contact the call centers, rather than providing a fragmented, product-specific view. Similarly when changes are made to customer information in one system, they should be automatically reflected in other systems. In the above example of an insurance company that sells health insurance and life insurance policies, changes to address information should be automatically reflected in both the systems. Such features are increasingly important since customers often ask for a consolidated view of their multiple accounts.

Solution

As shown in Figure 2-3 on page 28, this Application pattern is divided into three logical tiers: presentation, decomposition, and back-end application.

- ▶ Please refer to the Router application pattern for a description of the presentation tier.
- ▶ The decomposition tier supports most of the services provided by the router tier in the Router application pattern, including intelligent routing of requests, protocol conversion, security, and session concentration. In addition, it implements the intelligence to break down a single request received from a presentation client into several, simpler requests that it routes to multiple back-end applications. In doing so, it typically uses a local Work In Progress (WIP) database to store routing, decomposition, and re-composition rules and to cache the results from multiple back-end applications until a re-composition of the desired response has been generated. The decomposition tier implements significantly more business logic than a router tier. Such business logic focuses on providing a unified customer-centric view.
- ▶ The majority of the product and function-specific business logic is still concentrated in the back-end application tier. Some of these back-end applications are highly available and scalable online transaction processing systems and others are batch applications.

The requester interaction between the presentation and decomposition tier is synchronous. The requester interaction between the decomposition tier and the back-end application tier can be either synchronous or asynchronous.

A synchronous requester interaction is required when the presentation client expects an immediate answer, as in the case with the insurance company and its clients. In the insurance company example, a customer logs on to the self-service Web site and asks to view a consolidated bill. This request is decomposed into multiple synchronous requests that are targeted towards multiple product-specific billing systems. The decomposition tier waits for responses from these systems and combines the results and displays a consolidated billing view to the customer.

An asynchronous requester interaction between the decomposition tier and back-end applications is appropriate when the presentation client does not expect an immediate response. For example, consider the customer who initiates an electronic transfer of funds to pay for his or her monthly bills using a self-service Web site. This request can be decomposed into two separate requests. The first request is targeted towards a confirmation engine that synchronously provides a confirmation number to the customer for tracking purposes. At the same time, an asynchronous request can be sent to a batch system that transmits an electronic funds transfer request to a local bank using EDI technology.

A variation on this pattern includes caching on the second logical tier to avoid a high volume of accesses to the back-end application. Another variation is to use fast asynchronous communications so that multiple parallel requests can be sent to the third tier in order to improve response times over serial requests.

As discussed in later Product mappings for this Application pattern, advanced Web application servers such as WebSphere Advanced Edition V5 integrate with the full range of Message Oriented Middleware (MOM) to enable this Application pattern.

Considerations

There are a number of possible approaches to doing business request decomposition into multiple back-end transactions and recombining the responses into a single business response. These include:

- ▶ RYO (roll-your-own) programming that issues multiple asynchronous requests to back-end systems and combines the responses
- ▶ Using a message broker plus a rules engine, possibly with a two-phase commit (2-PC) or compensations mechanism
- ▶ Using a component broker with 2-PC protocols

Guidelines for use

All the guidelines documented under the Router application pattern apply here as well.

Benefits

This Application pattern provides all the benefits of the Router pattern, and diagonally:

- ▶ It provides a holistic customer-centric view rather than a fragmented product-centric view of information.
- ▶ Executing transactions in batch mode, when appropriate, provides several benefits including the ability to free up systems resources for more important tasks at hand. The Decomposition application pattern allows one to distinguish those transactions and use asynchronous mode for communication under such circumstances. This is particularly true for updates that need not be reflected into the appropriate data stores immediately.

Limitations

The focus of this Application pattern is providing a consolidated customer-centric view of information. However, such information is gathered only when required and is discarded at the end of the transaction. Because of this, this Application pattern does not accumulate all the necessary information in an Operational

Data Store (ODS) that can be used for mass customization of services and for cross-selling purposes.

Putting the Application pattern to use

The ITSO Vendor (IV) retailer company that was considered under the Router application pattern wants to extend beyond providing consistent access to all product information through all delivery channels.

The company wants to provide a consolidated customer-centric view through all its channels rather than providing fragmented product-specific views. For example, the company wants its customers to see a consolidated view across order submission and fulfillment. It wants to do so through all delivery channels, including Internet self-service, kiosks, voice recognition units, and the 24-hour call centers. To achieve these goals, the company chooses the Decomposition application pattern.



Runtime patterns

The Router application pattern represents a starting point for delivering a sophisticated e-business application that delivers information from back-end systems to multiple delivery channels, while exploiting the data integrity and performance of legacy applications.

The Decomposition application pattern extends the Router application pattern by decomposing a single, compound request from a client into several, simpler requests and intelligently routes them to multiple back-end applications.

The next step is to choose Runtime patterns that most closely match the requirements of the application. A Runtime pattern uses nodes to group functional and operational components. The nodes are interconnected to solve a business problem. Each Application pattern leads to one or more underpinning Runtime patterns.

3.1 Nodes

A runtime topology will consist of several nodes representing specific functions. 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.

User node

The user node is most frequently a personal computing device (PC) supporting a commercial browser, for example, Netscape Navigator and Internet Explorer. The browser is expected to support SSL and some level of DHTML. Increasingly, designers need to also consider that this node might be a pervasive computing device, such as a personal digital assistant (PDA).

Public Key Infrastructure (PKI)

PKI is a system for verifying the authenticity of each party involved in an Internet transaction, protecting against fraud or sabotage, and for non-repudiation purposes to help consumers and retailers protect themselves against denial of transactions. Trusted third-party organizations called Certificate Authorities (CA) issue digital certificates, which are attachments to electronic messages that specify key components of the user's identity.

During an Internet transaction using signed, encrypted messages, the parties can verify that the other's certificate is signed by a trusted Certificate Authority, before proceeding with the transaction. PKI can be embedded in software applications, or offered as a service or a product. PKI is critical for transaction security and integrity, and the software industry is moving to adopt open standards for their use.

Domain Name System (DNS) node

The DNS node assists in determining the physical network address associated with the symbolic address (URL) of the requested information. The Domain Name System node provides the technology platform to provide host-to-IP address mapping, that is, to allow for the translation of names (URLs) into IP addresses and vice versa.

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.

Load balancer node

The load balancer provides horizontal scalability by dispatching HTTP connections among several, identically configured Web servers. The load balancer component distributes interactive traffic across a number of hosts using dynamically updated rules for load balancing, while providing a single access point to the client system. It is used to achieve scalability through use of multiple servers, and high availability through being able to dynamically vary the algorithms by which a host is selected if one host fails or becomes overloaded.

The load balancer may be required to concurrently provide local or remote load balancing function for:

- ▶ Web servers
- ▶ Firewall nodes
- ▶ Directory and security servers
- ▶ Mail servers

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.

Web presentation node

The Web presentation server node provides services to enable a unified user interface. It is responsible for all presentation-related activity. In its simplest form, it serves HTML pages and runs servlets and JSPs. For more advanced patterns, it acts as a portal and provides the access integration services (Single Sign-On, for example). It interacts with the personalization server node to customize the presentation based on the individual user preferences or on the user role. The Web presentation server allows organizations and their users to standardize and configure the presentation of applications and data in the most efficient way, while enabling fine-grained access control.

Web application server node

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 HTTP client access 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 human resources (HR) systems.

This node would be provided by the company on company premises, or hosted in the enterprise network 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.

The node can contain these data types:

- ▶ HTML text pages, images, multimedia content to be downloaded to the client browser
- ▶ Servlets, JavaServer Pages
- ▶ Enterprise beans
- ▶ Application program libraries, such as Java applets for dynamic download to client workstations

Application server node

The application server node provides the infrastructure for application logic and can 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 can run both presentation and business logic. In other situations, it can be used for business logic only. See also “Web server redirector node” on page 35.

Integration server node

An integration server hosts application logic that can access and use information from existing databases, transaction functions from transaction monitor systems, and application capabilities from application packages. The integration server

can access back-end applications individually or combine this information and function in new ways. At a minimum, the integration server acts as an integration point for multiple presentation tiers (for example, call centers, branch offices, and Web browsers) so that they can share the infrastructure and applications on tiers 2 and 3.

Directory and security services node

The directory and security services node supplies information on the location, capabilities, and attributes (including user ID/password pairs and certificates) of resources and users known to this Web application system. This node can supply information for various security services (authentication and authorization) and can 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 this node also authenticates for access to the database server.

Existing applications and data node

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

3.2 Basic Runtime pattern for the Router pattern

In the Router application pattern, the router tier serves as an integration point for delivery channels in the presentation tier, allowing access to individual back-end applications. In the basic Runtime pattern, the functions of the router tier are performed by an integration server. The functions of the presentation tier are performed jointly by an Web server redirector and the application server. Placing a Web server redirector in the DMZ provides an extra layer of security by putting all application logic behind the firewall. Only a portion of the presentation function is left in the DMZ.

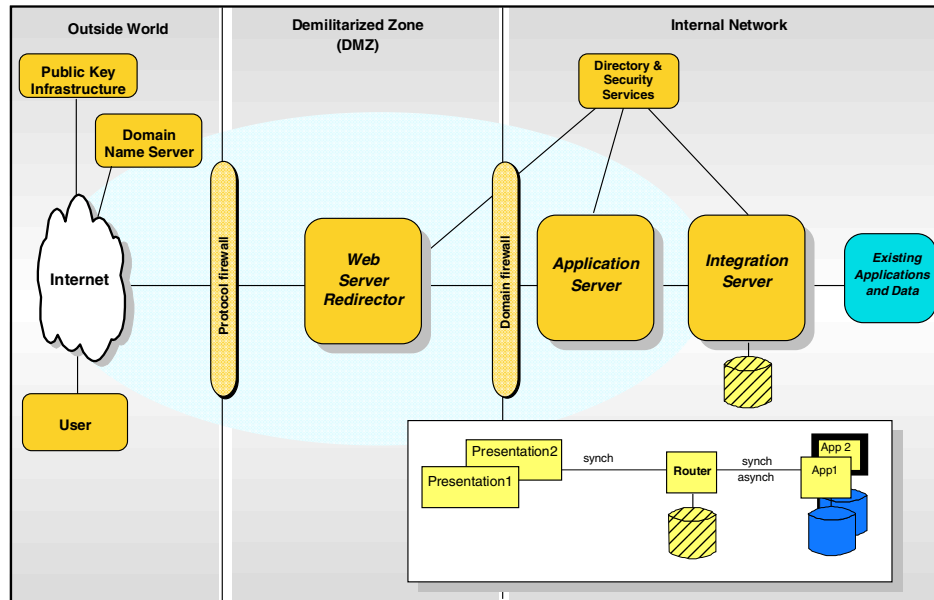


Figure 3-1 Basic Runtime pattern for the Router application pattern

The protocol firewall prevents unauthorized access from the Internet to the demilitarized zone. The role of this node is to allow the Internet traffic access only on certain ports and block other ports. The domain firewall prevents unauthorized access from the demilitarized zone to the internal network. The role of this firewall is to allow the network traffic that originated only from the demilitarized zone and not from the Internet.

A good security design does not permit any business logic or sensitive data in the DMZ. Using a Web server redirector helps to meet that goal. The Web server redirector serves static HTTP pages, while forwarding dynamic servlet and JSP requests to the application server. The presentation logic, therefore, spans both nodes. Together, these two provide the presentation tier, capable of handling multiple, diverse presentation styles. Using a redirector allows you to place the bulk of the business logic behind the protection of both the protocol and domain firewalls.

In addition to presentation logic (for example, JSPs), the application server contains some business logic. This is primarily in the form of the controlling servlets required to access the back-end applications. The application server builds a request based on user input and passes it to the integration server. The primary business logic resides in the back-end applications.

The integration server examines the request, determines the appropriate destination, and forwards it to the chosen back-end application. This may involve activities such as message transformation, protocol conversion, security management, and session concentration. The integration server may use a database to look up routing information as a caching device.

Access to the application server resources is protected by the application server's security features, while access to the integration server's resources is protected by the integration server's security features. User information that is needed for authentication and authorization by both servers is stored in the directory and security services node behind the domain firewall in the internal network. The information may contain user IDs, passwords, certificates, access groups, and so on.

Benefits and limitations

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

The basic pattern as shown is limited in availability, failover capability, and scalability. However, the pattern can be extended to use horizontal and vertical scaling and to take advantage of workload management techniques. For more information on how to do this, see *Patterns for the Edge of Network*, SG24-6822.

Since the requests to the application server need to be forwarded, you could see a performance degradation, depending on the redirector solution chosen.

In our retail company example of the IV Corp, data for each business function (registration, order process, and order fulfillment) is accessible through back-end applications. As a first step to integrating their operations, the company uses a Router application pattern solution that provides access to each back-end application. An IV Corp employee processing a registration application can access the customer's credit history before approving the registration. Read-only data at the router level provides a mapping between the user and the accounts held.

3.2.1 Variation 1

In this variation, the presentation logic has been split from the application logic and placed on a Web presentation server.

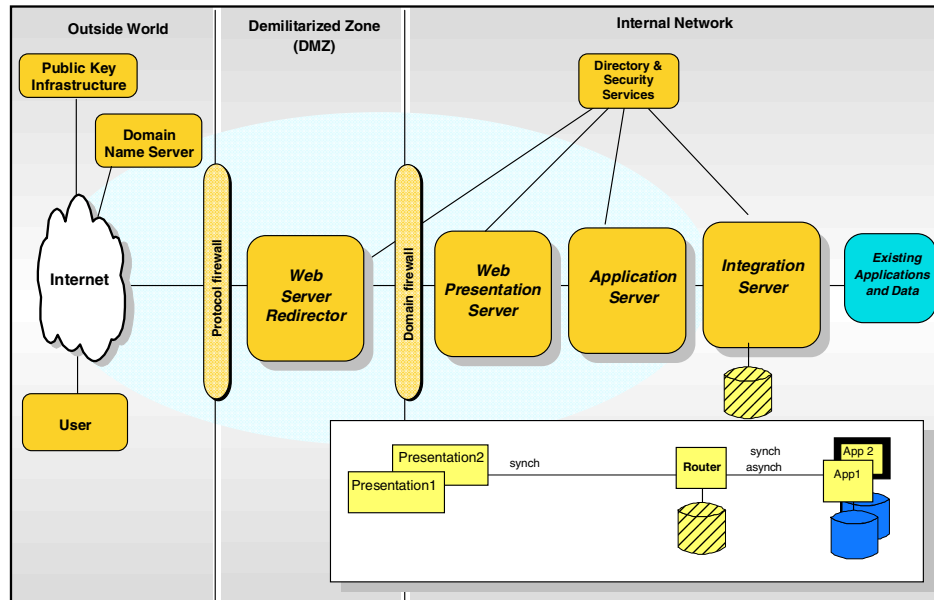


Figure 3-2 Basic Runtime pattern for the Router application pattern

The Web presentation server runs JSPs and servlets to provide the presentation logic for the application. The application server runs EJB logic and sends requests to the integration server. Requests are forwarded from the Web presentation server to the application server using IIOP.

Using a Web presentation server further delineates the line between presentation and application logic. It lends itself to scalability by allowing system resources to be spread across multiple machines and although not shown, can be extended to implement load balancing among application servers.

3.3 Basic Runtime pattern for Decomposition

On the surface, the Runtime pattern for the Decomposition application pattern will appear to be the same as that for the Router application pattern. The only visible difference in the Runtime pattern depiction is in the Application pattern overlay. However, this one small detail in the graphics represents a much larger change in the function provided by the integration server.

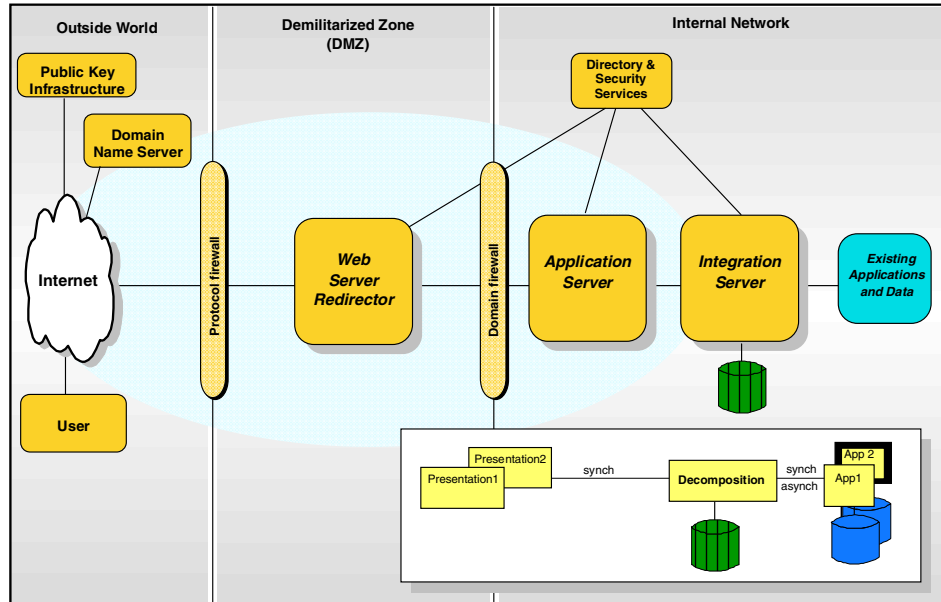


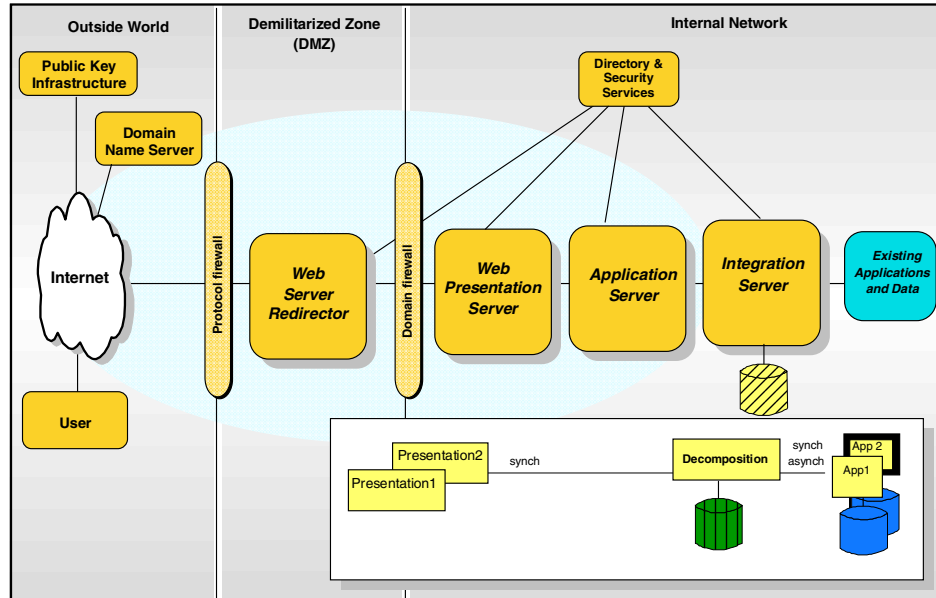
Figure 3-3 Basic Runtime pattern for the Decomposition application pattern

The integration server still examines messages and routes them to the appropriate back-end applications. But now, it can go a step further by taking a single complex message, decomposing it into multiple messages, and routing those messages to the appropriate back-end applications. It is also capable of managing these messages such that it can wait for responses and recompose them into a single response to be sent back to the user. This effectively takes multiple, diverse back-end applications and unifies them into one interface for the user.

The integration server can use a local database for the previously described routing functions, and as a work-in-progress database to store information required for message decomposition and recomposition.

3.3.1 Variation 1

Variation 1 is the same as Variation 1 in the Router pattern. In this variation, the presentation logic has been split from the application logic and placed on a Web presentation server.



The Web presentation server runs JSPs and servlets to provide the presentation logic for the application. The application server runs EJB logic and sends requests to the integration server. Requests are forwarded from the Web presentation server to the application server using IIOP.

Using a Web presentation server further delineates the line between presentation and application logic. It lends itself to scalability in by allowing system resources to be spread across multiple machines and although not shown, can be extended to implement load balancing among application servers.

3.4 For more information

For more information, please consult:

- *Patterns for the Edge of Network*, SG24-6822
- *IBM WebSphere V5.0 Performance, Scalability and High Availability*, SG24-6891



Product mapping

The next step after choosing a Runtime pattern is to determine the actual products and platforms to be used. The open standards and practices of the IBM e-business strategy let you develop and test an e-business application on your development runtime platform and easily deploy the application on any other supported platform. Further, it is common for a company to have a mixture of platforms within an integrated e-business solution. With their support for these multiple platforms, the Patterns for e-business solution designs are an appealing choice when faced with the requirement for integration with a mixed platform environment.

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 selected 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.

4.1 Runtime product mappings

The implementation of the Self-Service::Router application pattern and Decomposition application pattern in this book are done with three primary IBM products.

- ▶ The integration server that performs the Router and Decomposition functions of the application is implemented using IBM WebSphere MQ Integrator.
- ▶ The application server is WebSphere Application Server V5.0. The primary back-end application is also running on IBM WebSphere Application Server V5.0.
- ▶ The transport between the various servers is using IBM WebSphere MQ.

The product mappings for both the Router and Decomposition patterns are the same. The difference in patterns is determined by the functions implemented within IBM WebSphere MQ Integrator, which was used to provide both routing and decomposition.

IBM WebSphere MQ Integrator has the ability to retrieve a message from a message queue, parse it according to type, and make logical decisions regarding the message. In addition, the WebSphere MQ Integrator broker can alter the message, transform it to a different format, create multiple new messages from it, store information in a database, and perform many other actions that might be needed.

As a router, WebSphere MQ Integrator can take a message and route it based on the message content or input from other sources, such as a local database, to the appropriate back-end application to handle the request.

As a decomposition node, WebSphere MQ Integrator can take a single input message and split it (decompose it) into multiple back-end requests. The WebSphere MQ Integrator waits for the replies and recomposes them into one response, then sends it back to the user. For example, in our IV Corp sample, an order that is entered by a customer can have more than one component. This order is decomposed into sub-orders and each sub-order is routed to a specific supplier. When the sub-responses from each supplier return, WebSphere MQ Integrator recomposes them in a single response and returns it to the customer.

WebSphere MQ provides the transport mechanism for the messages. In our implementation, we use an WebSphere MQ queue manager on each server to transport the messages. The Java application uses JMS to place messages on queues. WebSphere MQ is then responsible for ensured delivery of the messages to the proper destination, in our case, the WebSphere MQ Integrator broker.

Using a Web server redirector node, we can place the majority of the business logic in the internal network, placing it behind two firewalls. The redirector is implemented using the WebSphere Application Server HTTP transport plug-in. The redirector serves static HTML pages and forwards requests for dynamic content to a WebSphere Application Server via HTTP/HTTPS protocol.

The back-end application in this mapping is represented by a Java application running on WebSphere Application Server; however, keep in mind that both IBM WebSphere Application Server and WebSphere MQ provide support for a broad range of back-end applications.

What's new? In the product mappings for these patterns based on WebSphere Application Server Version 4.0 (see *Self-Service Applications using IBM WebSphere V4.0 and IBM MQSeries Integrator*, SG24-6160-01), the Enterprise Edition was required on the Existing Applications and Data node to provide support for asynchronous message processing. WebSphere Application Server Version 5.0 provides support for message-driven beans to handle asynchronous messages.

Windows 2000

Figure 4-1 on page 46 shows a product mapping based on the Windows 2000 operating system platform.

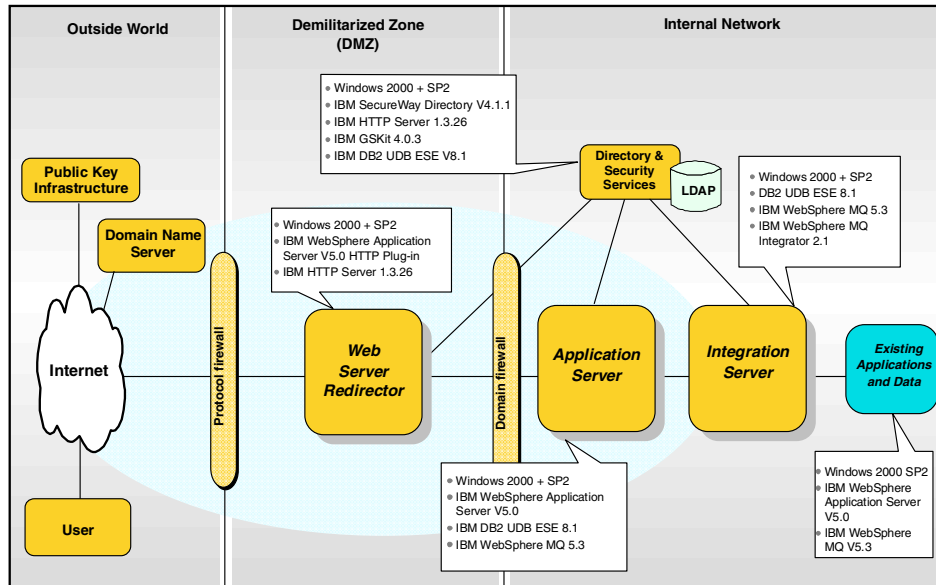


Figure 4-1 Windows 2000 product mapping

Variation 1

According to the Runtime pattern Variation 1, the following diagram depicts the Windows 2000 product mapping.

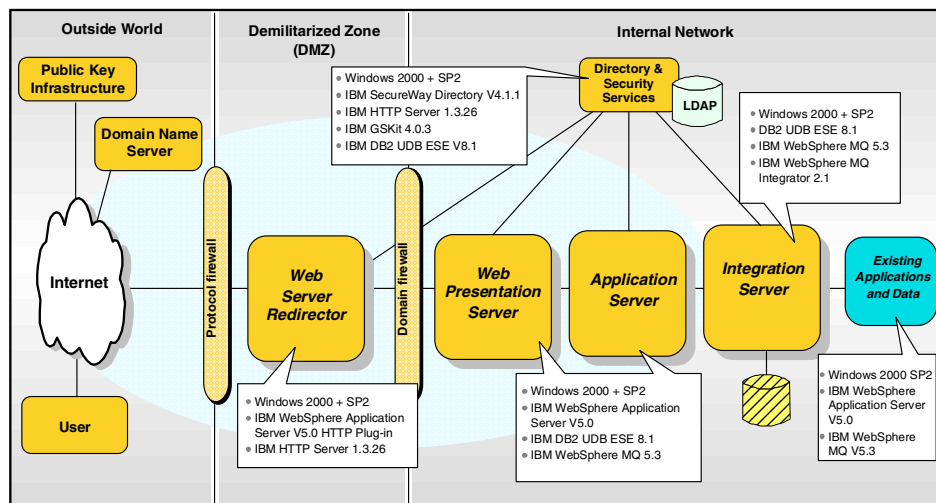


Figure 4-2 Windows 2000 product mapping for Variation 1

AIX®

Figure 4-3 shows a product mapping based on the AIX operating system platform.

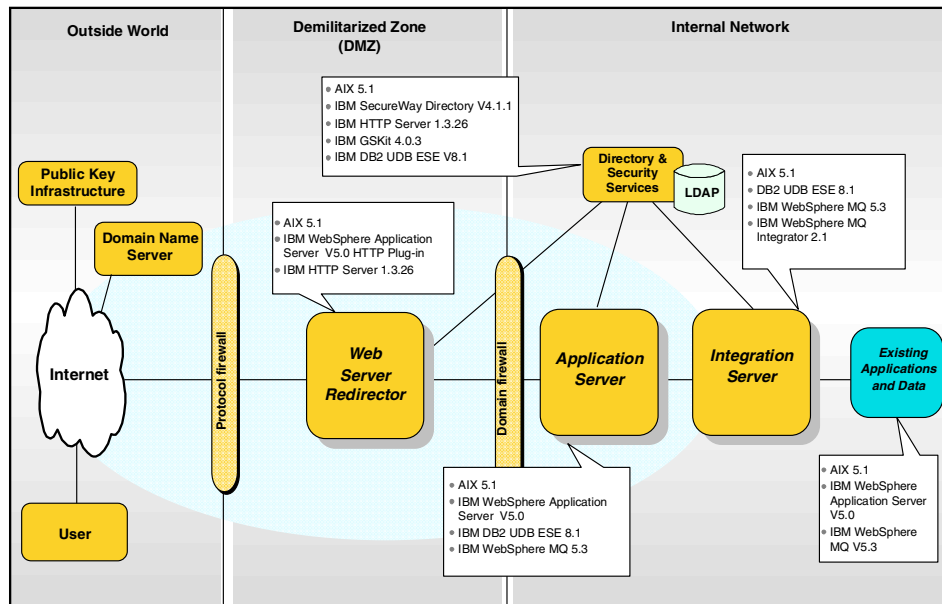


Figure 4-3 AIX product mapping

Variation 1

According to the Runtime pattern Variation 1, the following diagram depicts the AIX product mapping.

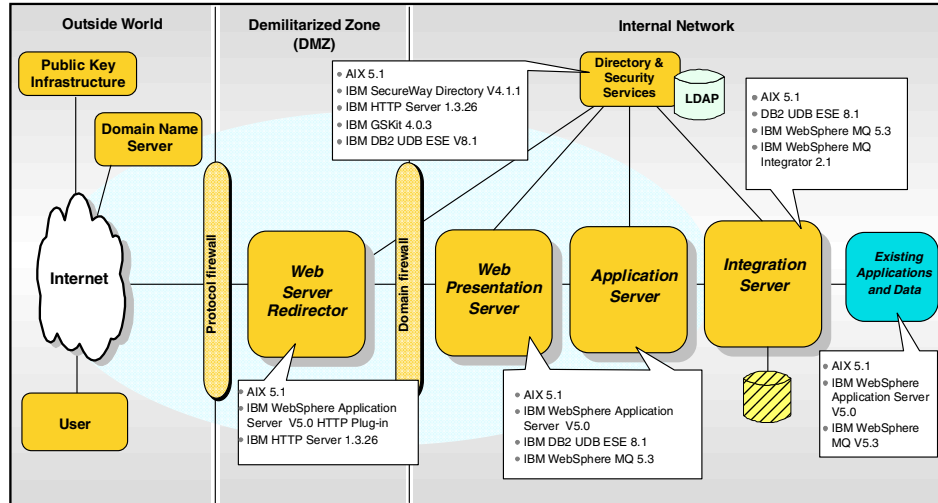


Figure 4-4 AIX product mapping for Variation 1

Linux

Figure 4-5 shows a product mapping based on the Linux operating system platform.

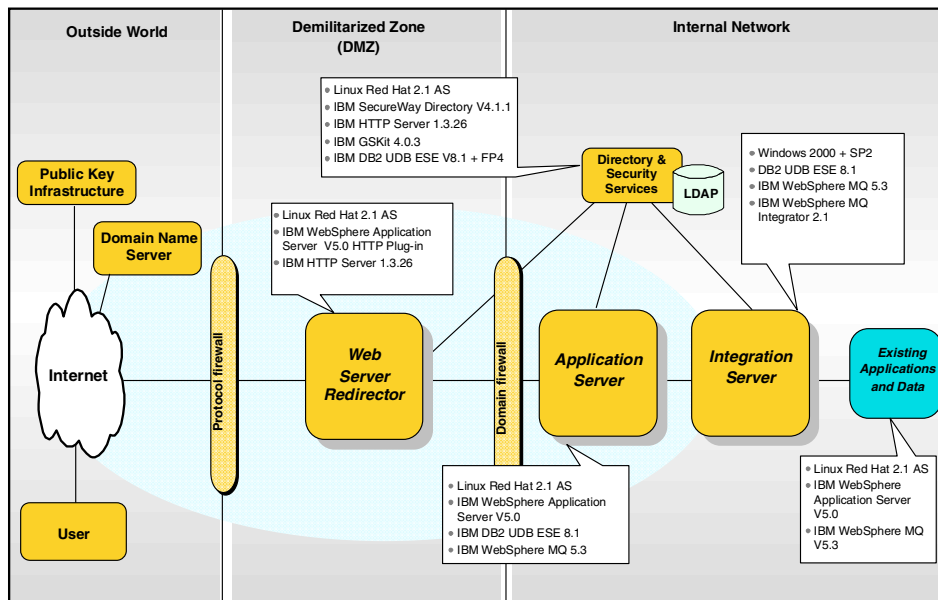


Figure 4-5 Linux product mapping

Variation 1

According to the Runtime pattern Variation 1, the following diagram depicts the Linux product mapping.

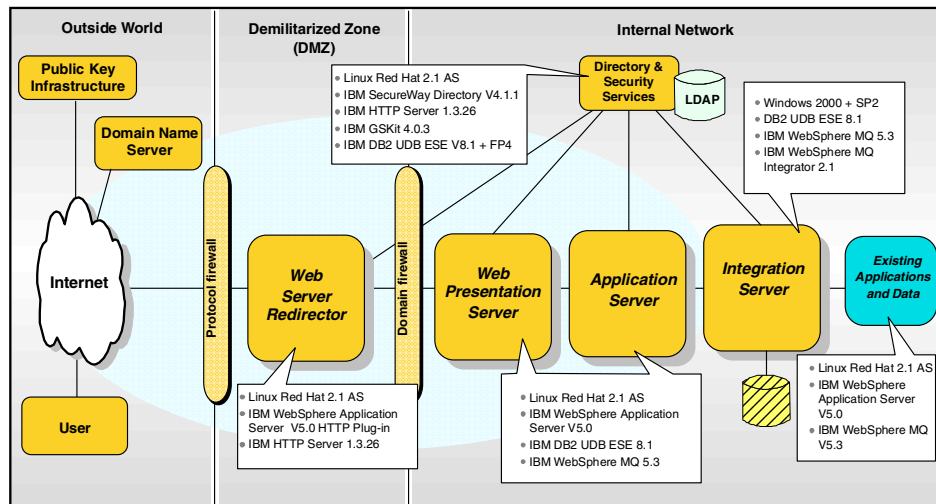


Figure 4-6 Linux product mapping for Variation 1

Note that WebSphere MQ Integrator V2.1 does not exist on Linux, therefore the product mapping for Linux includes the Windows 2000 version of WebSphere MQ Integrator V2.1.

4.2 Product summary

The Router and Decomposition application patterns assume a need for Web-enablement of back-end enterprise applications. Back-end integration requires a mechanism for applications to be able to talk to each other so information can be requested and transactions executed from Web-enabled front-ends to enterprise back-end applications. The need for intermediate action on these messages, such as determining the appropriate back-end system to handle the request or the transformation of data from one format to another, often arises in this situation.

Fulfilling this requirement, the WebSphere MQ family of products provides a highly scalable Message Oriented Middleware (MOM) infrastructure for application messaging, brokering, routing, aggregating, and transforming of business data. It provides the tools needed to apply business rules that act upon the data to suit the requirements and provides the flexibility to extend and reuse enterprise applications across a wide variety of platforms.

This book also shows the complementary products for a whole business integration solution but that are not part of the implementation in our sample scenario.

WebSphere Application Server V5

As the foundation of the WebSphere software platform, the IBM WebSphere Application Server Version 5 reinforces its reputation as the premier Java-based application platform, integrating enterprise data and transactions for the dynamic e-business world. WebSphere Application Server V5 provides a rich e-business application deployment environment with a set of application services that include enhanced capabilities for transaction management in a heterogeneous comprehensive Web services support, increased security, performance, availability, connectivity, and scalability. The new version manages and integrates enterprise-wide applications while leveraging open technologies and application programming interfaces (APIs).

The new capabilities of WebSphere Application Server V5 translate into four distinct benefits:

1. **Maximized ROI through integration support for Web services**

With Web services, you can reduce costs by finding the least expensive trading partners and sharing applications electronically with other organizations. SOAP, UDDI Registry, Web Services Invocation Framework (WSIF), Web Services Gateway, and preview versions of Apache SOAP and JSR 109 technologies make WebSphere Application Server V5 a production-ready Web application server for the development and deployment of enterprise Web services solutions for e-business. In addition, interoperability between Web services and J2EE enables key solution offerings for collaboration, B2B, portal serving, content management, commerce, and pervasive computing.

2. **Reduced costs by leveraging existing software assets**

WebSphere Application Server V5 lets companies grow while reducing costs along the way, by allowing them to use the systems in which they have already invested. J2EE Connector Architecture (JCA) and advanced XML and data transformations let companies reuse and integrate disparate systems and applications while allowing for cross-platform support and connectivity and integration with a variety of brand-name back-end systems. Native, high-performance JMS and J2EE 1.3 message beans enable dynamic application interactions, while a comprehensive and extensible integrated development and deployment platform optimizes development resources by facilitating reuse of CORBA, Microsoft, Java, and legacy assets.

3. Increased productivity

Recognizing that every deployment environment is unique, IBM has developed a single application server code-base with multiple configuration options - supporting a wide range of scenarios from simple administration of a single server to a clustered, highly available, high-volume environment with edge-of-network services. WebSphere support for JMX allows third-party products to read and manage WebSphere resources in a standard way. Additional capabilities are available through the simple, XML-based administration interface, including the ability to create and manage clusters, and quickly deploy new components, applications, and services.

4. Industry leading to handle the demands of dynamic e-business

WebSphere Application Server V5 offers expanded database support and enhanced security. In addition, integrated edge-of-network technology provides scalability, availability and performance through features such as:

- Dynamic caching
- Content distribution
- Transactional Quality of Service
- Dynamic workload management
- Enhanced multi-domain failover function
- High-availability LDAP support
- ESI-based dynamic fragment caching and invalidation capabilities

Improved system management and integration with best-of-breed performance management tools let you easily and dynamically manage and maintain your e-business.

IBM WebSphere Application Server V5 offers the following capabilities:

- ▶ Full J2EE (Java 2 Platform, Enterprise Edition) V1.3 compatibility including an enterprise-ready JMS provider based on technology in IBM WebSphere MQ.
- ▶ New packaging based on a single code base offers multiple deployment options from single server to clustered highly available, high-volume configurations.
- ▶ Improved system management and administration via browser-based administration, JMX, UI, and XML configuration.
- ▶ Tight integration with IBM WebSphere Studio, a highly productive development environment built on Eclipse, the premiere open systems development environment.
- ▶ Improvements over the performance offered by WebSphere Application Server V4 via additional distributed workload and enhanced caching capabilities and the inclusion of Dynamic Work Load Management among WebSphere Application Servers.

- ▶ Additional automation of distributed system management via centralized deployment manager and dynamic clustering capability for simplified administration of all clustered environments.
- ▶ Enhanced application availability and elimination of single points of failure between dispersed data centers with Advanced Multi-Domain Availability/Failover.
- ▶ Content Distribution Framework improves user response times and reduces bandwidth costs by enabling you to deploy published Web site content, including Web pages, fragments and application components, to caches and rehosting servers throughout the network.
- ▶ Enhanced user experiences through expedited, personalized page composition.
- ▶ Extended security capabilities for strong authentication security for client/server applications, using secret-key cryptography.
- ▶ Integration with Tivoli® Access Manager (formerly known as Tivoli Policy Director) software for centralized, site-wide authentication and access control and open security programming interfaces to support third-party security solutions.
- ▶ Option for in-memory replication of cached information for improved performance.
- ▶ Simplified problem determination with first-failure data capture and automated collection of system information to assist in remote problem resolution. Security enhancements include JAAS, CSiv2, and JCE.
- ▶ Broad cross-platform support.

For more information, please refer to:

<http://www-3.ibm.com/software/webservers/appserv/>

WebSphere MQ V5.3

IBM WebSphere MQ connects business software to form one efficient enterprise by providing an open, scalable, industrial-strength messaging backbone.

WebSphere MQ minimizes time taken to integrate key resources and applications held in different systems, so companies can respond to the changing demands of e-business. By connecting business information with people and other applications, more value can be extracted from existing investments, and new systems can be quickly integrated to support new market strategies.

WebSphere MQ does the following:

- ▶ Connects any commercial systems in business today (over 35 platforms supported)
- ▶ Ignores network disruptions – important data is always delivered
- ▶ Uses less time and resources to become an e-business
- ▶ Allows business to integrate disparate islands of automation
- ▶ Provides for time-independent communication
- ▶ Assures one-time delivery
- ▶ Supports a high-volume throughput customer experience in excess of 250 million messages a day

For more information, please refer to:

<http://www-3.ibm.com/software/ts/mqseries/>
<http://www-3.ibm.com/software/ts/mqseries/messaging/>

WebSphere MQ Integrator V2.1

WebSphere MQ Integrator Broker for Multiplatforms allows businesses to shape information flows according to their specific business needs, as follows:

- ▶ Transforms, augments, and applies rules to message-based data, and routes and distributes it between high performance systems.
- ▶ Integrates both existing and new applications with business data using dynamic content and topic-based publish/subscribe functions.
- ▶ Visualizes the application flow through a graphical development environment.
- ▶ Allows message formats to be defined through a variety of dictionaries, either those supplied with the product or from a third party.
- ▶ Simplifies support for multiple environments with a variety of application adapters, templates and tools.
- ▶ Provides a fully scalable architecture to meet growing business needs.
- ▶ Architected with an open framework that allows the use of built-in components together with third-party offerings.

For more information, please refer to:

<http://www-3.ibm.com/software/ts/mqseries/integrator/broker/>.

IBM DB2 V8.1

DB2 Universal Database™ Version 8.1 offers many database and data management enhancements.

IBM DB2 V8 for Linux, UNIX, and Windows marks the next stage in the evolution of the relational database. DB2 is the database of choice for the development and deployment of critical solutions such as:

- ▶ e-business
- ▶ Business intelligence
- ▶ Content management
- ▶ Enterprise Resource Planning
- ▶ Customer Relationship Management

DB2 UDB Version 8.1 enhancements include:

- ▶ Innovative manageability
 - Configuration Advisor enhances and makes DBA tasks easier
 - Health Center/Monitor keeps your database functioning
 - Memory Visualizer lets you dynamically see and control DB2's memory usage
 - Advisors to deliver expert advice on index and materialized query tables (MQTs)
 - Simplified management of large-scale partitioned databases
- ▶ New levels of integrated information
 - Federated Web Services allows combination of data from Web service providers
 - XML productivity tools simplify integrating XML
- ▶ Robust e-business foundation
 - Connection Concentrator for more user scalability
 - Dynamic configuration
 - In-place online reorganization
 - Online load
 - Online storage management
 - Null and default compression
 - Replication enhancements
 - New client architecture
- ▶ Integrated Business Intelligence
 - Multidimensional data clustering improves performance of complex queries
 - Real-time and bulk scoring of data
- ▶ Enhanced application development productivity
 - Development Center
 - WebSphere integration

- Microsoft integration
- Manageability

DB2 greatly reduces the complexity of data management by eliminating, simplifying, and automating many tasks traditionally associated with maintaining an enterprise-class database. Some of these advances are the first implementation of the Self-Managing and Resource Tuning (SMART) project and the first steps towards making full autonomic computing a reality for database implementations.

Performance and scalability enhancements

The Connection Concentrator reduces memory consumption on the database server by allowing transactions from remote clients to be concentrated or multiplexed across a small number of persistent database connections.

- ▶ Improved performance of databases with multiple partitions
- ▶ Multidimensional data clustering
- ▶ Prefetching enhancement
- ▶ Faster page cleaners
- ▶ User-maintained MQTs (or ASTs)
- ▶ Support for 64-bit Windows and 64-bit Linux
- ▶ New streamlined application drivers (ODBC, OLE DB, JDBC, and SQLj)
- ▶ New client architecture
- ▶ InfiniBand support

DB2 is the first database to support InfiniBand fabric connectivity for high-speed interconnections. DB2 V8 includes improved support for the InfiniBand architecture to deploy partitioned databases on multiple servers while offering greater scalability and better performance.

For more information, refer to:

<http://www-3.ibm.com/software/data/db2/udb/v8/>



Part 2

Guidelines



Technology options

In this chapter, we take a look at the Web application technology options you should consider. The recommendations are guided by the demands of reuse, flexibility, and interoperability, and subsequently are based on the open industry standards outlined by Java 2 Platform, Enterprise Edition (J2EE). Many of the choices continue to evolve and expand as the J2EE specification matures to include a broader view of the enterprise architecture. These recommendations are based on the J2EE1.3 specification and parts of the J2EE1.4 specification.

Our discussion of technology options is organized along the enterprise application tiers shown in Figure 5-1:

- ▶ Web client technologies for providing client-side presentation.
- ▶ Web application server technologies for providing server-side presentation and business logic.
- ▶ Integration technologies for providing access to the enterprise tier.

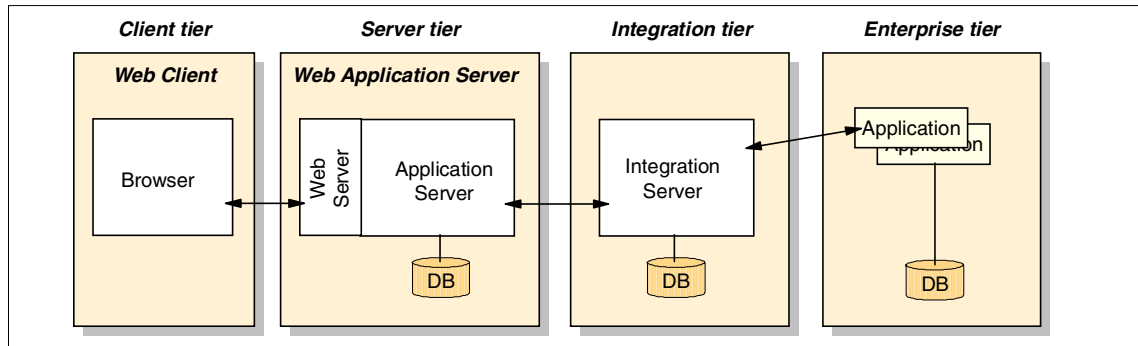


Figure 5-1 Self-Service application tiers

5.1 Web client

Figure 5-2 shows the recommended technologies for Web clients.

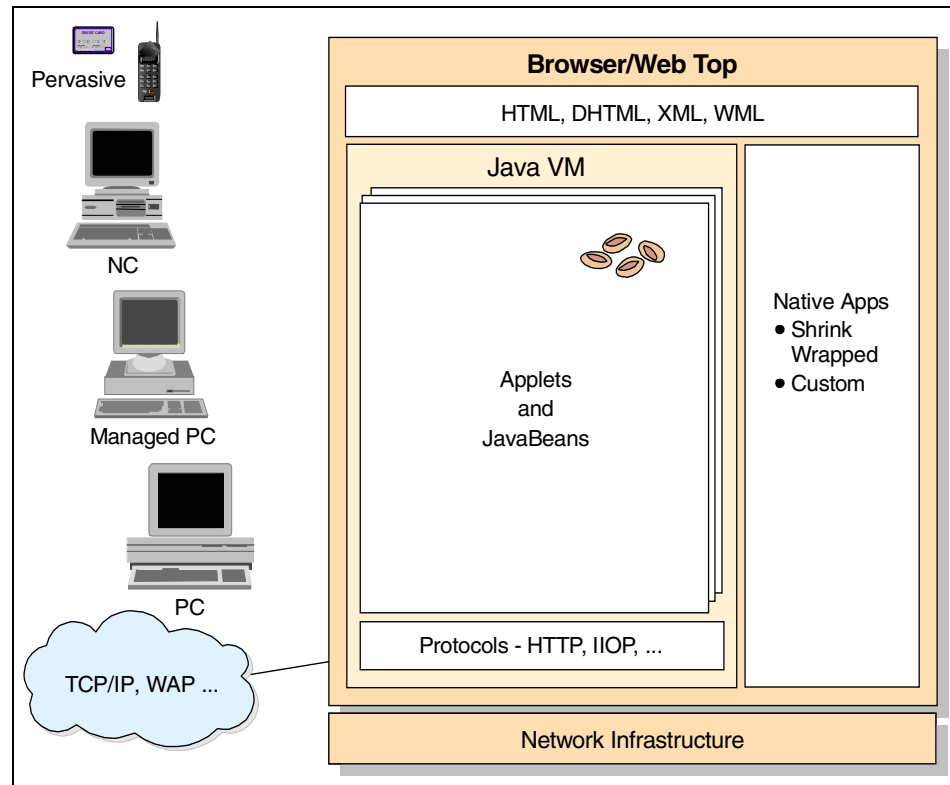


Figure 5-2 Web client technology model

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 selection of client-side technologies used in your design will require consideration for the server side, such as whether to store, or dynamically create, elements for the client side.

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.

We also touch on some of the current technology choices in the wireless area.

5.1.1 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 that 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. For Internet users, this is especially true. With intranet users, you can assume support for a standard browser. Regarding plug-ins, you need to consider what portion of your intended user community will have that capability.

Cross-browser strategies are required to ensure robust application development. Although many of these technology choices are maturing, they continue to be inconsistently supported by the full range of browser vendors. Developers must know browser compatibility for all features being exploited by the application. In general, developers will need to code to a lowest denominator or at least be able to distinguish among browser types using programmatic techniques. The key decision here is to determine the application requirements and behavior when handled by old browsers, other platforms such as Linux and Mac, and even the latest browsers.

In the J2EE model, the Web browser plays the role of client container. The model requires that the container provide a Java Runtime Environment as defined by the Java 2 Platform, Standard Edition (J2SE). However, 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).

5.1.2 HTML

HTML (HyperText Markup Language) 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, checkboxes, and 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. To ensure compatibility, developers should be unit testing pages against a validator tool. Free tools are available, such as the W3C HTML Validation Service, available at:

<http://validator.w3.org/>

5.1.3 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. See 5.1.4, “CSS” on page 64 for details on Cascading Style Sheets.

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 window, 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 be manipulated (moved, hidden, shown) using JavaScript only.

Due to varying levels of browser support, cross-browser design strategies must be used to ensure appropriate presentation and behavior of DHTML elements. In general this technology is not recommended unless its features are needed to meet usability requirements.

5.1.4 CSS

Cascading Style Sheets (CSS) allow you to define a common look and feel for HTML documents. This specification describes how Web documents are to be presented in print and online.

CSS is defined as a set of rules that are identified by selectors. When processed by the client browser, the selectors are matched to specific HTML tags and then are applied against the properties of the tag. This allows for global control over colors, fonts, margins, and borders. More advanced commands allow for control over pixel coordinates. Related stylesheet commands can be grouped and then externalized as a separate template file to be referenced by a multitude of Web pages.

CSS is defined as level1 and level 2 specifications. Level 1 was written with HTML in mind, while level 2 was expanded to include general markup styles for XML documents. Developers using CSS should unit test against a validator tool, such as the W3C CSS Validation Service at:

<http://jigsaw.w3.org/css-validator/>.

Due to varying levels of browser support, cross-browser design strategies must be used to ensure appropriate presentation and behavior of CSS elements. In general, this technology should be used with great attention to support of specification elements.

5.1.5 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) that 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.

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 that uses JavaScript 1.2 extensions must detect the browser being used, and select the correct statements to run.

One caveat is that users can disable JavaScript on the client browser, but this can be programmatically detected.

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.

5.1.6 Java applets

The most flexibility 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, and so on.

You can use either the awt or the Swing classes to build a Java applet. But while designing your applet, you should keep in mind that Swing is supported only by later browser versions.

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.

JavaScript can be used to invoke methods on an applet using the SCRIPT tag in the applet's HTML page.

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.

Using Java plug-ins, you can extend the functionality of your browser to support a particular version of Java. Java plug-ins are part of the Java Runtime Environment (JRE) and they are installed when the JRE gets installed on the computer. You can specify certain tags in your Web page, to use a particular JRE. This will download the particular JRE if it is not found on the local computer. This can be done in HTML either through the:

- ▶ Conventional APPLET tag, or
- ▶ OBJECT tag instead of APPLET tag for Internet Explorer or the EMBED tag with the APPLET tag for Netscape.

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.

Using HTTP tunneling, an applet can call back on the server without reloading the HTML page. For users who are behind a restrictive firewall, HTTP tunneling offers a bidirectional data connection to connect to a system outside the firewall.

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.

5.1.7 XML (client side)

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

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, 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 5.2.4, “XML” on page 75 for additional discussion of the server-side use of XML.

5.1.8 XHTML 1.1 (HTML 4.01)

XHTML (Extended HyperText Markup Language) is an extension to HTML 4, which supports document types that are XML based. It is intended to be used as a language for XML-conforming content as well as for HTML 4-conforming user agents.

The advantages of XHTML are:

- ▶ Since XHTML documents are XML conforming, they can be viewed, edited, and validated with standard XML tools.
- ▶ XHTML documents can be used to traverse either the HTML Document Object Model or the XML Document Object Model.

Some issues with XHTML are:

- ▶ XHTML documents are not as easy to create as HTML documents because XHTML is validated more strictly than HTML.
- ▶ HTML already used so widely that it is difficult for XHTML to attract the attention of most Web developers.
- ▶ Browser support is not usually an issue as documents can be created using HTML-compatible XHTML that is understood by most browsers. There are also utilities that can be used to convert HTML documents to HTML-compatible XHTML.
- ▶ Development tool support for XHTML is also improving. The Page Designer tool in IBM WebSphere Studio Application Developer V5.0, for example, allows visual authoring of XHTML pages.

XHTML Basic is designed for Web clients that do not support the full set of XHTML features. It is meant to serve as a common language and share basic content across mobile phones, pagers, car navigation systems, vending machines, etc.

Some of the common features found in Wireless Markup Language (WML) and other subsets of HTML have been used as the basis for developing XHTML Basic:

- ▶ Basic text
- ▶ Basic forms and tables
- ▶ Hyperlinks

Some HTML 4 features have been found inappropriate for non-desktop devices, so extending and building on XHTML Basic will help to bridge that gap.

5.1.9 VoiceXML

VoiceXML is a dialog markup language that leverages the other specifications for creating dialogs that feature synthesized speech, digitized audio, speech recognition, DTMF (touch tone) input, etc.

5.1.10 XForms

XForms is W3C's specification for Web forms that can be used with desktop computers, hand-held devices, etc. The disadvantage of the HTML Web forms is that there is no separation of purpose from presentation. XForms separates the data and logic of a form from its presentation. Also, XForms are device independent.

XForms uses XML for transporting the data that is displayed on the form and the data that is submitted from the form. HTML is used for the data display.

Currently, the main issue with XForms is it is still an emerging technology so browser and server support is not yet standard.

5.1.11 XSLT

Extensible Stylesheet Language Transformations (XSLT) is a W3C specification for transforming XML documents into other XML documents. The XSLT is built on top of the Extensible Stylesheet Language (XSL) which, is a stylesheet language for XML. Unlike CSS2, XSL is also a transformation language.

A transformation expressed in the XSLT language defines a set of rules for transforming a source tree to a result tree and it is expressed in the form of a stylesheet.

An XSLT processor is used for transforming a source document to a result document. There are currently a number of XSLT processors available on the market. DataPower has introduced a XSL just-in-time (JIT) compiler, which speeds up the time taken for the XSL transformation.

The XSLT processor has a performance overhead, so online processing of larger documents can be slow.

5.1.12 Mobile clients

Mobile clients include wireless devices such as phones, pagers, and PDAs. The challenges these devices bring as Web clients are based primarily on the very limited computer resources of the supporting platform. The goal, however, is to overcome these limitations to provide access to information and application services from anywhere by leveraging and extending the existing Web server architectures.

Devices

Mobile devices include wireless desktop PCs, WAP devices, i-mode devices, PDAs, and Phone w/Voice. PDA devices cannot run the major operating systems that run on desktop PCs and consequently there are various mobile device-specific platforms. Palm devices use Palm-OS. WinCE/PocketPC devices use a version of Microsoft Windows called Windows CE.

Voice

Voice-enabled applications allow for a hands-free user experience unencumbered by the limitations of computer interface controls.

Voice technology fall into two categories: those that recognize speech and those that generate speech. The ability to recognize human voice by computers is called Automatic Speech Recognition (ASR). The ability to generate speech from written text is called speech synthesis or Text-to-Speech (TTS).

Architecture

Support for mobile clients impacts the runtime topology and therefore must be designed and implemented using best practices for system architecture. The good news is that any past investment in Web architecture to support Internet-based applications can be extended to support mobile clients.

A Wireless Application Protocol (WAP) gateway is used between the mobile client device and the Web server. The gateway translates requests from the wireless protocol into HTTP requests and, conversely, converts HTTP requests into the appropriate device format.

WAP

WAP is the Wireless Application Protocol. This is the standard for presentation and delivery of information to wireless devices, which are platform, device and network neutral. The goal of this protocol is to provide a platform for global, secure access through mobile phones, pagers, and other wireless devices.

Microbrowser

WAP microbrowsers run on mobile clients. They are responsible for the display of Web pages written in WML and can execute WMLScripts. These play the same role as HTML browsers that run on a PC.

WML

The Wireless Markup Language (WML) is based on XML and HTML 4.0 to fit small hand-held devices. It is a tag-based language that handles formatting static text and images, can accept data input, and can follow hyperlinks.

WMLScript

This is the companion language to WML, in the same way as JavaScript is a companion language to HTML. WMLScript allows for procedural programming such as loops, conditional and event handling. It has been optimized for a small memory footprint and small devices. This language is derived from JavaScript.

Bluetooth

Bluetooth is a set of technical specifications for low-range (up to 30 feet) wireless devices that define standards such as power use and radio frequency. The goal of this technology is to connect a wide range of computing and telecommunication devices easily and simply in a peer-to-peer manner.

5.2 Web application server

Figure 5-3 on page 72 shows the recommended technology model for a Web application server.

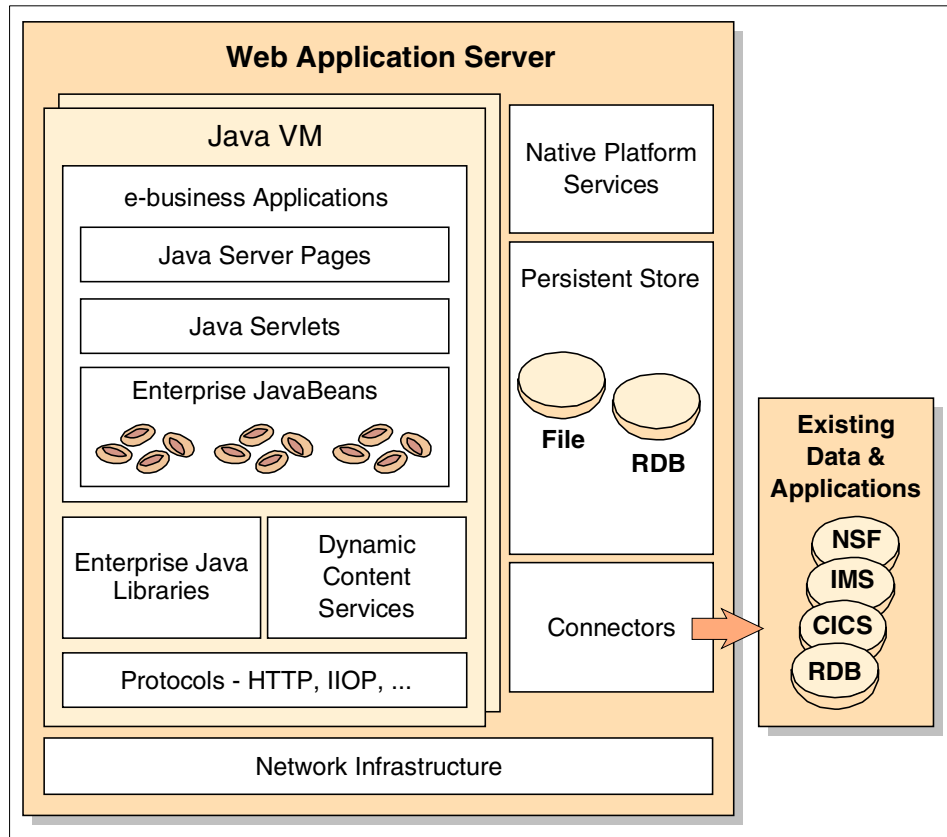


Figure 5-3 Web application server technology model

We assume in this book that you are 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 that is experiencing widespread industry adoption.

Before looking at the technologies and APIs available in the Web application programming environment, first let's have a word about two fundamental operational components on this node, the HTTP server and the application server. For production applications, they should be chosen for their operational characteristics in areas such as robustness, performance, and availability.

We follow the well-known Model-View-Controller (MVC) design structure so often used in user interfaces. For the Web application programming model:

- The Model represents the data of the application, and the business rules and logic that govern the processing of the data. In a J2EE application, the model

is usually represented to the View and the Controller via a set of JavaBeans components.

- ▶ The View is a visual representation of the model. Multiple views can exist simultaneously for the same model and each view is responsible for making sure that it is presenting the most current data by either subscribing to state change events or by making periodic queries to the model. With J2EE, the view is generally implemented using JavaServer Pages (JSP).
- ▶ The Interaction Controller decouples the visual presentation from the underlying business data and logic by handling user interactions and controlling access to the model. It processes the incoming HTTP requests and invokes the appropriate business or UI logic. Using J2EE, the controller is often implemented as a servlet.

5.2.1 Java servlets

Servlets are Java-based software components that can respond to HTTP requests with dynamically generated HTML. Servlets are more efficient than CGI for Web request processing, since they do not create a new process for each request.

Servlets run within a Web container as defined by the J2EE model and therefore have access to the rich set of Java-based APIs and services. In this model, the HTTP request is invoked by a client such as a Web browser using the servlet URL. Parameters associated with the request are passed into the servlet via the `HttpServletRequest`, which maintains the data in the form of name/value pairs. Servlets maintain state across multiple requests by accessing the current `HttpSession` object, which is unique per client and remains available throughout the life of the client session.

Acting as an MVC Controller component, a servlet delegates the requested tasks to beans that coordinate the execution of business logic. The results of the tasks are then forwarded to a View component, such as a JSP to produce formatted output.

One of the attractions of using servlets is that the API is a very accessible one for a Java programmer to master. The specification of J2EE 1.3 platform requires Servlet API 2.3 for support of packaging and installation of Web applications.

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.

5.2.2 JavaServer Pages (JSPs)

JSPs were designed to simplify the process of creating Web 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 JSP acts as the View component in the MVC model.

The chief advantage JSPs have over standard Java servlets is that they are closer to the presentation medium. A JavaServer Page is developed as an HTML page. Once compiled it runs as a servlet. JSPs can contain all the HTML tags that Web authors are familiar with. A JSP may contain fragments of Java code that encapsulate the logic that generates the content for the page. These code fragments may call out to beans to access reusable components and enterprise data.

JSP technology uses XML-like tags and scriptlets written in Java programming language to encapsulate the conditional logic that generates dynamic content for an HTML page. In the runtime environment, JSPs are compiled into servlets before being executed on the Web application. Output is not limited to HTML but also includes WML, XML, cHTML, DHTML, and VoiceXML. The JSP API for J2EE 1.3 is JSP 1.2.

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 is 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.

5.2.3 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 JavaBeans or just beans. One of the design criteria for the JavaBeans 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 JavaServer Pages specification includes a set of tags for accessing JavaBeans properties.

5.2.4 XML

XML (Extensible Markup Language.) and XSL stylesheets 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 stylesheets that use CSS.

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 or Schema. 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.

IBM's XML4J XML parser was made available through the Apache open source organization under the Xerces name. For open source XML frameworks, see:

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

Defining XML documents

XML documents are defined using DTDs or XML Schemas.

DTDs are a basic XML definition language, inherited from the SGML specification. The DTD specifies what markup tags can be used in the document along with their structure.

DTDs have two major problems:

- ▶ Poor data typing: In DTDs elements can only be specified as EMPTY, ANY, element content, or mixed element-and-text content, and there is no standard way to specify null values for elements.

Data typing like date formats, numbers, or other common data types cannot be specified in the DTD, so a XML document may comply with the DTD but still have data type errors that can only be detected by the application.

- ▶ Not defined in XML: DTD uses its own language to define XML syntax, that is not compliant to the XML specification. This makes it difficult to manipulate a DTD.

To solve this problems, the World Wide Web Consortium (W3C) defined a new standard to define XML documents called XML Schema. XML Schema provides the following advantages over DTDs:

- ▶ Strong typing for elements and attributes
- ▶ Standardized way to represent null values for elements
- ▶ Key mechanism that is directly analogous to relational database foreign keys
- ▶ Defined as XML documents, making them programmatically accessible

Even though XML Schema is a more powerful technology to define XML documents, it is also a lot harder to work with, so DTDs are still widely used to define XML documents. Additionally, simple, not hard-typified documents can be easily defined using DTDs with similar results to using XML Schema.

Whether to use one or the other will depend on the complexity of the messages and the validation requirements of the application. Actually in many cases both (a DTD and a XML Schema) are provided, so they can be used by the application depending on its requirements.

Note: We have to remember that the validation process of a XML document using XML Schemas is an *expensive process*. Validation should be performed only when it is necessary.

XSLT

Extensible Stylesheet Language Transformations (XSLT) is a W3C specification for transforming XML documents into other XML documents. The XSLT is built on top of the Extensible Stylesheet Language (XSL) which, like CSS2 seen in 5.1.4, “CSS” on page 64, is a stylesheet language for XML. Unlike CSS2, XSL is also a transformation language.

A transformation expressed in the XSLT language defines a set of rules for transforming a source tree to a result tree and it is expressed in the form of a stylesheet.

An XSLT processor is used for transforming a source document to a result document. There are currently a number of XSLT processors available on the market. DataPower has introduced a XSL just-in-time (JIT) compiler, which speeds up the time taken for the XSL transformation.

The XSLT processor has a performance overhead, so online processing of larger documents can be slow.

XML Security

XML security is an important issue, particularly where XML is being used to by organizations to interchange data across the Internet. Several new XML security specifications are working their way through three standards bodies - the W3C (World Wide Web Consortium), IETF (Internet Engineering Task Force), and OASIS (Organization for the Advancement of Structured Information Standards). We highlight a few of them here:

- ▶ XML Signature Syntax and Processing is a specification for digitally signing electronic documents using XML syntax. According to the W3C, "XML Signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere."

A key feature of the protocol is the ability to sign parts of an XML document rather than the document in its entirety. This is necessary because an XML document might contain elements that will change as the document is passed along or various elements that will be signed by different parties.

WebSphere Studio provides you with the ability to create (using a wizard) and verify XML digital signatures.

- ▶ XML encryption will allow encryption of digital content, such as Graphical Interchange Format (GIF) images or XML fragments. XML Encryption allows the parts of an XML document to be encrypted while leaving other parts open, encryption of the XML itself, or the super-encryption of data (that is, encrypting an XML document when some elements have already been encrypted).
- ▶ XKMS (XML Key Management Specification) establishes a standard for XML-based applications to use Public Key Infrastructure (PKI) when handling digitally signed or encrypted XML documents. XML signature addresses message and user integrity, but not issues of trust that key cryptography ensures.
- ▶ SAML (Security Assertion Markup Language) is the first industry standard for secure e-commerce transactions using XML. It aims to standardize the exchange of user identities and authorizations by defining how this information is to be presented in XML documents, regardless of the underlying security systems in place.

For further discussion, see Sun ONE article *Riddle Me This: Is Your XML Data Safe?* by Brett Mendel:

<http://dcb.sun.com/practices/websecurity/overviews/xmldata.jsp>

Advantages of XML

There are many advantages of XML in a broad range of areas. Some of the factors that influenced the wide acceptance of XML are:

- ▶ Acceptability of use for data transfer
XML is a standard way of putting information in a format that can be processed and exchanged across different hardware devices, operating systems, software applications and the Web.
- ▶ Uniformity and conformity
XML gives you an common format that could be developed upon and is accepted industry-wide.
- ▶ Simplicity and openness
Information coded in XML is human readable.
- ▶ Separation of data and display
The representation of the data is separated from the presentation and formatting of the data for display in a browser or other device.
- ▶ Industry acceptance
XML has been accepted widely by the information technology and computing industry. Numerous tools and utilities are available, along with new products for parsing and transforming XML data to other data, or for display.

Disadvantages of XML

Some XML issues to consider are:

- ▶ Complexity
While XML tags can allow software to recognize meaningful content within documents, this is only useful to the extent that the software reading the document knows what the tagged content means in human terms, and knows what to do with it.
- ▶ Standardization
When multiple applications use XML to communicate with each other they need to agree on the tag names they are using. While industry-specific standard tag definitions often do exist, you can still declare your own non-standard tags.
- ▶ Large size

XML documents tend to be larger in size than other forms of data representation.

5.2.5 Enterprise JavaBeans

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

Enterprise beans are distinguished from JavaBeans 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 three types of Enterprise JavaBeans:

- ▶ Session beans
- ▶ Entity beans
- ▶ Message-driven beans

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.

A typical Message-Driven Bean has the following characteristics:

- ▶ Consumes messages sent to a specific queue.
- ▶ Is asynchronously invoked.
- ▶ Is stateless.
- ▶ Can be transaction aware.
- ▶ May update shared data in underlying client message.
- ▶ Executes upon receipt of single client message.
- ▶ Has no component or home interface.
- ▶ Is removed when the EJB container crashes. The container has to re-establish a new message-driven object to continue computation.

Typically, an entity bean is used for information that has to survive system restarts. In session beans, on the other hand, 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.

An important change to the specification in EJB 2.0 is the addition of a new enterprise bean type, the message-driven bean (MDB). The message-driven bean is designed specifically to handle incoming JMS messages. The EJB container uses message properties and bean deployment descriptor to select the bean to invoke when a message arrives, so your application logic only needs to process the message contents.

The J2EE 1.3 platform requires support for EJB 2.0. As a tool provider the WebSphere Application Server V5.0 supports J2EE 1.3 and therefore supports EJB 2.0. EJBs are packaged into EJB modules (JAR files) and then combined with Web modules (WAR files) to form an enterprise application (EAR file). EJB deployment requires generating EJB deployment code specific to the target application server.

5.2.6 Additional enterprise Java APIs

The J2EE specification defines a set of related APIs that work together. Here are the remainder not discussed so far:

- ▶ JNDI: Java Naming and Directory Interface. This package provides a common API to a directory service independent of any directory access protocol. This allows for easy migration to new directory services. Through this interface, component providers can store and retrieve Java object instances by name. Service provider implementations include those for JDBC data sources, 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
 - Locating a driver-specific data source
- ▶ RMI-IIOP: Remote Method Invocation (RMI) and RMI over IIOP are part of the EJB specification as the access method for clients to access EJB services. From the component provider point of view, these calls are local. The EJB container takes care of calling the remote methods and receiving the response. To use this API, component providers create an IDL description of the EJB interface and then compile it to generate the client-side and server-side stubs. The stubs connect the object implementations with the Object Request Broker (ORB). ORBs communicate with each other through the Internet Inter-ORB Protocol (IIOP). RMI can also be used to implement limited-function Java servers.
- ▶ JTA: Java Transaction API. 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.
- ▶ JAF: JavaBeans Activation Framework. This API is not intended for typical application use, but it is required by the JavaMail API.
- ▶ JavaMail: This is a set of classes for supporting e-mail. Functionally it provides APIs for reading, sending, and composing Internet mail. This API models a mail delivery system and requires the SMTP for sending mail and POP3 or IMAP for receiving mail. Special data wrapper classes are provided to view and edit data in the mail content. Support for MIME data is delegated to the JAF-aware beans.

- ▶ JAXP: API for parsing and transforming XML documents.
- ▶ JAAS: Java Authentication and Authorization Service.

5.3 Integration technologies

With the continuous progress of enterprise computing, more and more enterprises are finding the need to quickly adopt new technologies and integrate with existing applications. Furthermore, it is often not feasible for enterprises to completely discard their existing infrastructure, due to limitations in cost and human resources.

Enterprise application integration (EAI) allows disparate applications to communicate with each other. Some points you should consider while deciding on the connector technology between your application and the enterprise tier applications are as follows:

- ▶ The current infrastructure

Do you already have a messaging system on the enterprise tier? Then it makes sense to go for JMS. Or if you have a legacy-system, such as CICS® or IMS™, J2EE Connectors might be the better choice.
- ▶ Time to market

Web service enabling an application is relatively fast with the Web services development tools available.
- ▶ Future expansion plans

If you plan to expand your enterprise systems in the future, you need to keep in mind the integration with your current infrastructure and your planned infrastructure. Web services may provide the most cost-effective migration path in such a case.
- ▶ Reliability

JMS with WebSphere MQ, for example, can be used to provide assured transfer of data, even when the enterprise application is unavailable.
- ▶ Transaction support

Web services currently do not offer support for transactions. If your application needs transactional management, it might be worthwhile considering either JMS or J2EE Connectors.

5.3.1 Web services

The W3C's Web Services Architecture Working Group has jointly come to agreement on the following working definition of a Web service:

“A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML based messages exchanged via Internet-based protocols.”

Basic Web services combine the power of two ubiquitous technologies: XML, the universal data description language, and the HTTP transport protocol widely supported by browser and Web servers.

Web services = XML + transport protocol (such as HTTP)

Let's take a closer look:

- ▶ Web services are self-contained.

On the client side, no additional software is required. A programming language with XML and HTTP client support is enough to get you started. On the server side, merely a Web server and a servlet engine are required. It is possible to Web service enable an existing application without writing a single line of code.

- ▶ Web services are self-describing.

Neither the client nor the server knows or cares about anything besides the format and content of request and response messages (loosely coupled application integration).

The definition of the message format travels with the message. No external metadata repositories or code generation tools are required.

- ▶ Web services are modular.

Web services are a technology for deploying and providing access to business functions over the Web; J2EE, CORBA and other standards are technologies for implementing these Web services.

- ▶ Web services can be published, located, and invoked across the Web.

The standards required to do so are:

- Simple Object Access Protocol (SOAP) also known as service-oriented architecture protocol, an XML-based RPC and messaging protocol.
- Web Service Description Language (WSDL) is a descriptive interface and protocol binding language.
- Universal Description, Discovery, and Integration (UDDI), a registry mechanism that can be used to look up Web service descriptions.

- ▶ Web services are language independent and interoperable.

The interaction between a service provider and a service requester is designed to be completely platform and language independent. This interaction requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP). Because the service provider and the service requester have no idea what platforms or languages each other are using, interoperability is a given.

- ▶ Web services are inherently open and standards based.

XML and HTTP are the technical foundation for Web services. A large part of the Web service technology has been built using open source projects. Therefore, vendor independence and interoperability are realistic goals this time.

- ▶ Web services are dynamic.

Dynamic e-business can become a reality using Web services because, with UDDI and WSDL, the Web service description and discovery can be automated.

- ▶ Web services are composable.

Simple Web services can be aggregated to more complex ones, either using workflow techniques or by calling lower-layer Web services from a Web service implementation.

- ▶ Web services are built on proven, mature technology.

There are a lot of commonalities, as well as a few fundamental differences between Web services and other distributed computing frameworks. For example, the transport protocol is text based and not binary.

WebSphere V5.0 provides support for Web services. WebSphere applications can send and receive SOAP messages and also communicate with UDDI registries to publish and find services.

For detailed information on Web services, check out the following:

- ▶ IBM Redbooks:
 - *WebSphere Version 5 Web Services Handbook*, SG24-6891
 - *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292
- ▶ The World Wide Web Consortium (W3C) Web site at <http://www.w3.org/>.

Static and dynamic Web services

There are two ways of binding to Web services: *static* and *dynamic*:

- ▶ In the static process, the binding is done at design time. The service requester obtains service interface and implementation description through a proprietary channel from the service provider (by e-mail, for example), and stores it into a local configuration file. No private, public, or shared UDDI registry is involved.
- ▶ The dynamic binding occurs at runtime. While the client application is running, it dynamically locates the service using a UDDI registry and then dynamically binds to it using WSDL and SOAP.

This requires that the contents of the UDDI registry be trusted. Currently, only private UDDI networks can provide such control over the contents.

Web services and the service-oriented architecture

Service-oriented architectures (SOA) support a programming model that allows service components residing on a network to be published, discovered, and invoked by each other in a platform, network protocol and language independent manner.

The origin of SOA can be traced back to Remote Procedure Calls (RPC), distributed object protocols such as CORBA and Java RMI, and component based architecture such as J2EE/EJBs (Sun) and (D)COM/COM+/.Net (Microsoft).

Using XML over HTTP, Web services extend the SOA programming model into the global Internet allowing the publication, deployment, and discovery of service applications over the Internet.

For more information on SOA and Web services, refer to:

<http://www.ibm.com/software/solutions/webservices/resources.html>

This Web site provides a collection of IBM resources on this topic. For example, you can find an introduction to the SOA in a white paper titled Web Services Conceptual Architecture (WSCA 1.0).

Advantages of Web services

Web services technology enables businesses to:

- ▶ Deliver new IT solutions faster and at lower cost by focusing their code development on core business, and using Web services applications for non-core business programming.
- ▶ Protect their investment in IT legacy systems by using Web services to wrap legacy software systems for integration with modern IT systems.

- ▶ Integrate their business processes with customers and partners at less cost. Web services make this integration feasible by allowing businesses to share processes without sharing technology. With lower costs, even small business will be able to participate in B2B integration.
- ▶ Enter new markets and widen their customer base. Web services listed in UDDI registries can be “discovered” and thus are “visible” to the entire Web community.

Disadvantages of Web services

Some Web services issues to consider are:

- ▶ Binding to Web services dynamically requires that the contents of the UDDI registry be trusted. Currently, only private UDDI networks can provide such control over the contents.
- ▶ The SOAP server footprint is significant and the technology is relatively new, so adding the Web service provider stack to existing enterprise systems can be a problem.

5.3.2 J2EE Connector Architecture

The J2EE Connector Architecture is aimed at providing a standard way of accessing enterprise applications from a J2EE-based Java application. It defines a set of Java interfaces, through which application developers can access heterogeneous EIS systems, for example legacy systems such as CICS, and Enterprise Resource Planning (ERP) applications.

J2EE Connector Architecture 1.0 support is a requirement of the J2EE 1.3 specification. It provides access to a range of systems through a common client interface API (CCI). Application programmers code to the single API rather than having unique interfaces for each proprietary system. The link from the API to the enterprise system is called a resource adapter and is provided by a third-party vendor. This is somewhat analogous to the model for JDBC drivers. Resource adapters are packaged as resource adapter archive (RAR) files.

IBM WebSphere Application Server V5.0 supports the J2EE Connector Architecture 1.0, as required by the J2EE 1.3 specification. The administrative console supports J2EE Connector resource adapter configuration. The administrative console allows the association of connection factories for the resource adapter that encapsulate the pooling attributes. Component Providers request a connection for an enterprise information system (EIS) from the connection factory through the JNDI lookup mechanism. IBM supplies resource adapters for enterprise systems such as CICS, HOD, IMS, SAP, and Crossworlds as separate products.

IBM WebSphere Studio Application Developer V5.0 supports application development using J2EE Connectors, and development of custom J2EE Connectors.

CICS resource adapter

The CICS Transaction Gateway (CTG) V5 is a set of client and server software components that allow a Java application to invoke services in a CICS region.

The CTG offers three basic interfaces for Java clients:

- ▶ ECI (External Call Interface) for COMMAREA-based CICS applications.
- ▶ EPI (External Presentation Interface) for 3270-based transactions.
- ▶ ESI (External Security Interface) for password management in order to verify and change user IDs and passwords.

The CICS resource adapter is covered in detail in the following chapters.

IMS resource adapter

The IMS Connector for Java provides a way to create Java applications that can access IMS transactions. The IMS Connector for Java uses IMS Connect to access IMS. IMS Connect is a facility that runs on the host IMS machine and supports TCP/IP and Local Option communication to IMS. A Java application or servlet accesses IMS Open Transaction Manager Access (OTMA) through IMS Connect. IMS Connect accepts messages from its TCP/IP clients and routes them to IMS OTMA using the Cross-System Coupling Facility (XCF).

The runtime component of IMS Connector for Java is provided as a component of IMS Connect Version 1 Release 2 (Program Number 5655-E51). The J2EE connector implementation of this runtime component is also referred to as the IBM WebSphere Adapter for IMS. It is packaged as a RAR file, `imsico.rar`, for deployment into WebSphere Application Server. The RAR file is installed to a target directory from the IBM IMS Connect, Version 1 Release 2.

Advantages of J2EE Connectors

Some reasons to use J2EE Connectors are:

- ▶ The common client interface simplifies application integration with diverse EISs. This common interface makes it easy to plug third-party or home-grown resource adapters into your applications.
- ▶ Each EIS requires just one implementation of the resource adapter, since there is no need to custom develop an adapter for every application.
- ▶ J2EE Connectors facilitate scalability and provide quality of service features transparently to the client application.

- ▶ J2EE Connector Architecture-compliant resource adapters are portable across J2EE application servers. If a vendor provides a resource adapter for WebLogic, for example, it should also work with WebSphere Application Server.
- ▶ J2EE Connectors have low intrusion on the enterprise system, because native client interfaces are utilized.

Disadvantages of J2EE Connectors

Some J2EE Connector issues to consider are:

- ▶ J2EE Connector Architecture has support only for synchronous communication. (The CICS adapter does offer support for non-blocking calls.) Support for asynchronous communications is expected in the J2EE Connector Architecture 1.5 specification.
- ▶ The J2EE Connectors standard is still relatively new and performance compared with previous alternatives has not been firmly established. For example, some customers may prefer to continue with the well proven non-J2EE Connector CTG base classes.
- ▶ Though J2EE Connector Architecture promises an abstraction to access any legacy system, with J2EE Connector Architecture 1.0, parts of the client application need to have resource adapter-specific implementation. This means that if you have to change the resource adapter (move to a different enterprise system, which provides a different adapter), the client application will be impacted.

For more information on J2EE Connectors and CICS, refer to the following redbooks:

- ▶ *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401.
- ▶ *Revealed! Architecting Web Access to CICS*, SG24-5466

5.3.3 Java Message Service

Messaging middleware is a popular choice for accessing existing enterprise systems in an asynchronous manner. A standard way for using messaging middleware from a Java application is using the Java Message Service (JMS) interface. JMS offers Java programmers a common way to create, send, receive and read enterprise messages. The JMS specification was developed by Sun Microsystems with the active involvement of IBM, other enterprise messaging vendors, transaction processing vendors, and RDBMS vendors.

In IBM WebSphere Application Server V5.0, the J2EE 1.3 specification is implemented, which includes JMS 1.0 and EJB 2.0.

According to the JMS 1.0 specification a message provider is integrated in application server. As shown in Figure 5-4, the integrated message provider makes it possible to communicate asynchronously with other WebSphere applications, without installing separate messaging software like IBM WebSphere MQ. WebSphere's integrated JMS server is based on IBM WebSphere MQ.

WebSphere Application Server V5 provides support to external messaging providers too. One specific external provider can be WebSphere MQ, others can be configured as Generic messaging providers in WebSphere.

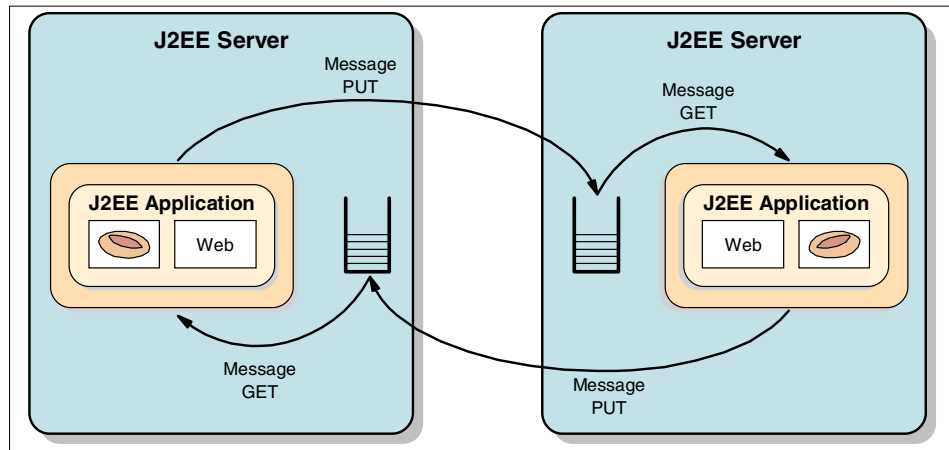


Figure 5-4 Integrated JMS Provider

An important new feature of EJB 2.0 is message-driven beans (MDB). As described in 5.2.5, “Enterprise JavaBeans” on page 79, Message-driven beans are designed specifically to handle incoming JMS messages.

What is messaging?

Messaging is a form of communication between two or more software applications or components. One of the strength of messaging is application integration. Messaging communication is loosely coupled as compared to tightly coupled technologies such as Remote Method Invocation (RMI) or remote procedure calls (RPC). The sender does not need to know anything about the receiver for communication. The message to be delivered is sent to a destination (queue) by a sender component and the recipient picks it up from there. Moreover, the sender and receiver do not both have to be available at the same time to communicate.

JMS has two messaging styles, or in other words two domains:

- One-to-one, or point-to-point model

- ▶ Publish/subscribe model

JMS and IBM WebSphere MQ

When you want to integrate with an application not based on WebSphere Application Server V5.0 an external JMS provider is needed. IBM WebSphere MQ V5.3 includes built-in JMS provider support with enhanced performance features for integrating JMS applications with other applications.

WebSphere MQ enables application integration by allowing business applications to exchange information across different platforms, sending and receiving data as messages. WebSphere MQ takes care of network interfaces, assures once and once only delivery of messages, deal with communications protocols, dynamically distribute workload across available resources, and handle recovery after system problems.

Advantages of JMS

The JMS standard is important because:

- ▶ It is the first enterprise messaging API that has achieved wide cross-industry support.
- ▶ It simplifies the development of enterprise applications by providing standard messaging concepts and conventions that apply across a wide range of enterprise messaging systems.
- ▶ It leverages existing, enterprise-proven messaging systems.
- ▶ It allows you to extend existing message-based applications by adding new JMS clients that are integrated fully with their existing non-JMS clients.
- ▶ Developers have to learn only one common interface for accessing diverse messaging systems.

Disadvantages of JMS

Though JMS provides a common interface for Java applications to interact with messaging systems, it might lose out on some specific functionality offered by the messaging vendor. In that case, you might still have to write vendor-specific code to access such functionality.

JMS only provides asynchronous messaging so the design is more complex when addressing response correlation, error handling and data synchronization.

Further information on JMS can be found in IBM Redbook *MQSeries Programming Patterns*, SG24-6506.

5.3.4 Message Oriented Middleware

The Message Oriented Middleware (MOM) consists of a transport layer that deals with the communication (physical) aspects of moving messages between the origination and the destination points. It takes care of the networking protocol issues, the message encoding and even translation. It is said to encapsulate the business of transporting message. At the same time, it provides APIs to enable application to manipulate (send/receive) messages. It is obviously a very critical and absolutely necessary part of the messaging-based integration layer.

The transport layer per se does not do any message transformation or handle message routing.

From the integration layer perspective such basic messaging systems offer limited value. They were therefore enriched with the additional functionality of handling the transformation and containing dynamic routing capabilities. This in turn evolved even further to support not only point-to-point distribution but also publish/subscribe style scenarios and handling of the complex message flows. The messaging system components providing this functionality are commonly known as the message brokers, for example, WebSphere MQ.

Message broker services

A message broker is a server-side service in a message-oriented system. It manipulates the messages it receives, and performs some or all of the following functions:

- ▶ Routing services
 - Content based
 - Publish/subscribe style
 - Message flow processing
- ▶ Data transformation (mapping message to other formats) services
 - Syntactic (format driven)
 - Semantic (content driven)
- ▶ Security (authentication/authorization, encryption/decryption) services
- ▶ Transactional service (unit of work support)
- ▶ Error handling
- ▶ Service invocation (invoking an external function to process message payload)

A message flow is a sequence of operations on a message, dependent upon the message content and the current message flow state. Message flows can be viewed as *business services*, initiated by receiving a message.

If the broker supports the publish/subscribe style of messaging, subscribing applications can register their topic subscriptions with the broker. Publishing applications can then send messages to the broker, who will map the topic to the registered subscribers and forward the message to all interested parties.

The message-oriented systems providing broker services are “natural” integration layer technologies for our Self-Service patterns.

A very good example of the messaging system providing an extensive set of the message broker services is the IBM WebSphere MQ Integrator product.

5.3.5 Others

In this section we briefly touch on a few other integration technologies, including:

- ▶ RMI/IIOP
- ▶ CORBA

RMI/IIOP

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

The Internet Inter-ORB (Object Request Broker) Protocol (IIOP) is a protocol used for communication between CORBA object request brokers. An object request broker is a library that enables CORBA objects to locate and to communicate with one another.

RMI/IIOP is an implementation of the RMI API over IIOP that allows developers to write remote interfaces in the Java programming language.

CORBA

Common Object Request Broker Architecture (CORBA) is a platform-, language-, and vendor-neutral standard for writing distributed object systems. The CORBA standard was developed by the Object Management Group (OMG), a consortium of companies founded in 1989. CORBA offers a broad range of middleware services, including naming service, relationship service, and so on.

CORBA can be used for integration with legacy applications. This is done by creating a CORBA wrapper for the existing application, which can then be invoked by other applications.

CORBA is just a specification and there are a number of vendors (such as IONA or Borland) that implement it. Each vendor will provide additional value-added services such as persistence, security, and so on, which can be leveraged by CORBA developers.

The disadvantage of CORBA is in the steep learning curve involved. Also, CORBA is slow-moving; it takes a long time for the OMG to adopt a new feature.

5.4 Where to find more information

For more information on topics discussed in this chapter, see:

- ▶ Redbook *Mobile Applications with IBM WebSphere Everyplace Access Design and Development*, SG24-6259
- ▶ Redbook *WebSphere Version 5 Web Services Handbook*, SG24-6891
- ▶ Redbook *Web Services Wizardry with WebSphere Studio Application Developer*, SG24-6292
- ▶ Redbook *Java Connectors for CICS: Featuring the J2EE Connector Architecture*, SG24-6401.
- ▶ Redbook *Revealed! Architecting Web Access to CICS*, SG24-5466
- ▶ Redbook *MQSeries Programming Patterns*, SG24-6506
- ▶ 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
- ▶ IBM CICS
<http://www.ibm.com/software/ts/cics>
- ▶ IBM WebSphere MQ
<http://www.ibm.com/software/ts/mqseries>
- ▶ ECMAScript language specification
<http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM>

- ▶ Java APIs and technology
<http://java.sun.com/products>
- ▶ Validator tools
<http://validator.w3.org/>
<http://jigsaw.w3.org/css-validator/>
- ▶ Bluetooth Web site
<http://www.bluetooth.com>
- ▶ *Bluetooth Applications in Pervasive Computing* white paper at:
<http://www.ibm.com/pvc/tech/bluetoothpvc.shtml>
- ▶ World Wide Web Consortium (W3C) site
<http://www.w3.org/>
- ▶ Open source XML frameworks
<http://xml.apache.org/>
- ▶ Sun ONE article, *Riddle Me This: Is Your XML Data Safe?* by Brett Mendel:
<http://dcb.sun.com/practices/websecurity/overviews/xmldata.jsp>
- ▶ Service-oriented architecture and Web services:
<http://www.ibm.com/software/solutions/webservices/resources.html>



Application design

To illustrate the techniques used in building a Self-Service application that uses router and decomposition, we built a sample application for this project. This chapter discusses Java design techniques used to build this application.

The information presented here is intended to supplement the information in *Patterns: Self-Service Application Solutions using WebSphere V5.0*, SG24-6591-00. We build on the application design information in that book by extending the design to include messaging capability.

In this chapter, we will focus on:

- ▶ Using Struts to create the front end
- ▶ Using JMS to place a message on a queue
- ▶ Using container-managed two-phase commit transaction processing

6.1 Application structure

The Self-Service Web application can be described as a set of interactions between a Web browser and a Web application server. The interaction begins with an initial request from the user's Web browser for the welcome page of the application. All subsequent interactions are initiated by the user by clicking a button or a link, causing a request to be sent to the Web application server. The Web application server processes the request, dynamically generates a results page, and then sends it back to the client along with a set of buttons and links that will generate the next request.

6.1.1 Model-View-Controller design pattern

One problem that is sometimes seen with large application development is that the flow of the application is hard-wired in the presentation layer implementation itself. This makes later application flow modifications and maintenance difficult and expensive.

The Model-View-Controller design pattern divides the presentation layer from the application flow and logic by dividing the application into three layers:

- ▶ **Model:** the Model contains the core application functionality, both business logic and data. The model is fully decoupled from the View or the Controller.
- ▶ **View:** the View provides the presentation of the model, in other words, how the application looks from a user's point of view. The View can access the Model information but should not make any changes to the Model directly. Any change in the Model should be passed on to the View.
- ▶ **Controller:** the Controller, as its name suggests, is responsible for the application flow. It controls how the user's input is translated into Model interactions and how the Model is later presented to the user through the View.

As shown in Figure 6-1 on page 97, the 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 (MVC) design pattern.

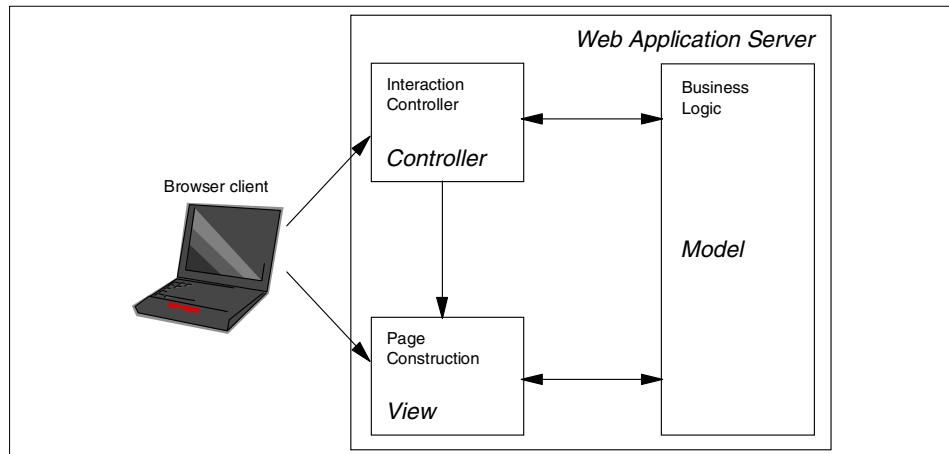


Figure 6-1 Model-View-Controller design

Many 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 Self-Service Web applications. Throughout this chapter, Model is often referred to as business logic, View is referred to as page constructor or display page, and Controller is referred to as interaction controller.

6.1.2 Struts

In a Web environment, changes in the Model cannot be passed on to the View directly. Instead, the View has to re-query the server to discover modifications in the state of the application. This is called MVC Model 2 or MVC 2. In this redbook, we use a well-known implementation of the MVC 2 design pattern called Struts. Struts is an open-source implementation of MVC 2, part of the Jakarta project in the Apache Software Foundation. Struts is a framework that implements the MVC 2 pattern by providing a set of classes, servlets, JSP tag libraries, and a flexible configuration.

The core of the Struts framework is a flexible control layer based on such standard technologies as Java servlets, JavaBeans, ResourceBundles, and Extensible Markup Language (XML).

Struts encourages application architectures based on the Model 2 approach, a variation of the classic Model-View-Controller (MVC) design paradigm. Struts provides its own Controller component and integrates with other technologies to provide the Model and the View. For the Model, Struts can interact with any

standard data access technology, including Enterprise JavaBeans, JDBC, and Object Relational Bridge. For the View, Struts works well with JavaServer Pages, Velocity Templates, XSLT, and other presentation systems.

The Struts framework provides the invisible underpinnings every professional Web application needs to survive. Struts helps you create an extensible development environment for your application, based on published standards and proven design patterns.

Struts application design

True to the MVC design pattern, Struts applications have three major components:

- ▶ Controller, implemented using the Struts ActionServlet and classes extending the Struts Action class
- ▶ View, implemented using JavaServer Pages and Struts form beans
- ▶ Model, implementing the application's business logic

The ActionServlet routes HTTP requests from the user to the appropriate action class. Action classes provide access to the application's business logic and control how the flow should proceed. Form beans are used to collect and validate form data from the user.

Figure 6-2 shows an example Struts form bean, TransferFundsForm, that has been defined in the struts-config.xml file and linked to an action mapping. When a request calls for the TransferFundsAction Struts action, the ActionServlet retrieves the form bean (or creates it if it does not exist), and passes it to the action.

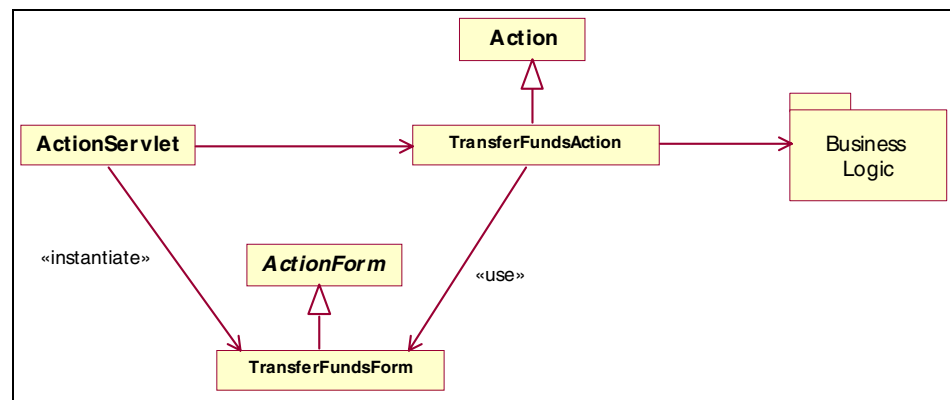


Figure 6-2 Struts action class diagram

The action can then check the contents of the form bean before its input form is displayed, and also queue messages to be handled by the form. When ready, the action can return control with a forward to its output form, usually a JSP. The `ActionServlet` can then respond to the HTTP request and direct the client to the JSP.

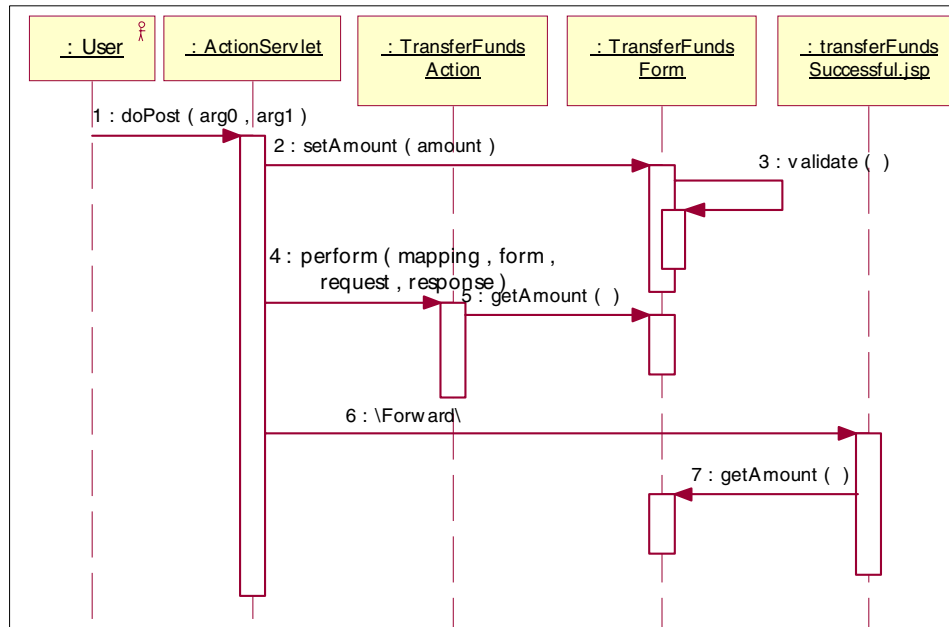


Figure 6-3 Struts action sequence diagram

6.1.3 Sample application

The following diagram depicts the two use cases that are implemented in the sample application.

- Order placement
- Order pickup

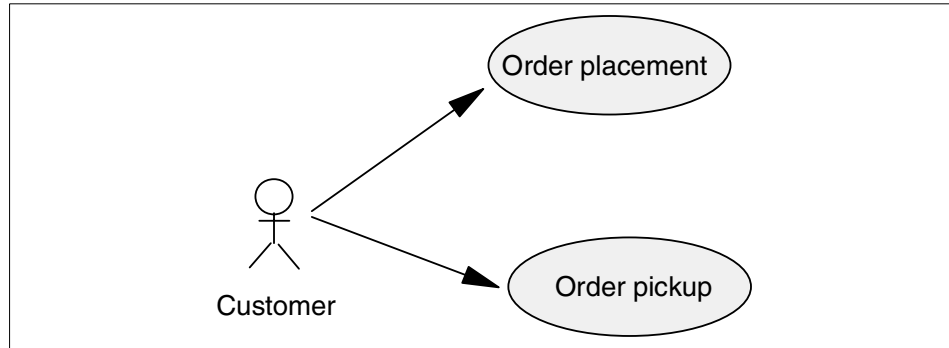


Figure 6-4 Main use case

These two use cases are only a smaller part of a scenario that is not implemented fully. The implementation provided together with the book is intended to show the technologies used in this pattern.

Order placement

In this use case, the customer goes to the IV Corp Web site to place an order. This consists of selecting an item and quantity, then submitting the order.

Figure 6-5 on page 101 is a functional diagram that show the different nodes (front-end, back-end, integration server, and Supplier application), functional items (enterprise applications, integration flows) and resources (data sources, queue connection factories, queues), for the sample application.

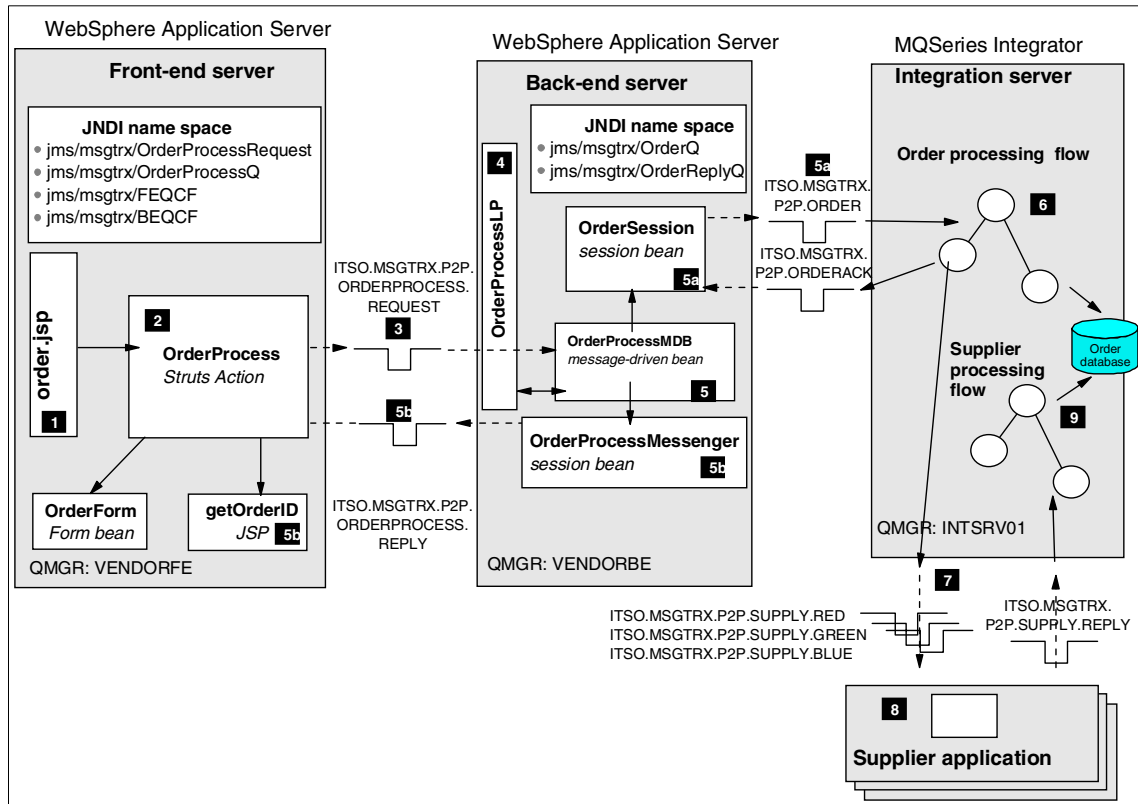


Figure 6-5 Order placement flow

The order placement process flows in the following way.

View

1. The user accesses the order placement page, `order.jsp`, on the front-end server. The requested item and quantity are entered in this page and submitted for processing.

The product catalog is stored in an XML file for the application. The `order.jsp` page generates the catalog page from the XML file using a stylesheet. The result is a list of products on the HTML page where the user can choose an item to order.

Controller

2. The Web application executes the `OrderProcess` Struts action. The parameters from the HTML form are passed to the action in a form bean called `OrderForm`.

3. OrderProcess creates an XML document based on the request, then sends a message to the back-end server over JMS with the order details.

OrderProcess then waits for a reply from the back end to confirm the execution of the order process.

In order to match the reply message with the request message, the JMS message correlation ID is used in the application. The correlation ID stores the request message ID. When the response arrives, the sender can compare the original message ID with the returned correlation ID. If they are the same, the reply arrived properly; if not, the reply belongs to another request.

This type of implementation is a message-facade, where the front-end server sends a message to the back-end server with the requested action and pertinent parameters. On the back-end server, one message-driven bean listens to every action request and starts up the appropriate action on the server. This is very similar to the facade pattern that is commonly used with session EJBs and entity EJBs.

Model

4. On the back-end server, the OrderProcessLP listener port monitors the queue for incoming messages.
5. The OrderProcessMDB message-driven bean picks up the message and instantiates two beans, OrderSession and OrderProcessMessenger. It uses the local interfaces to call the session beans to improve performance.
 - a. The OrderSession session bean performs some pre-processing of the order and then generates a message to be sent to the integration node.

Ordered items can be thought of as a sum of components. The components are available from individual suppliers and then “assembled” for delivery to the customer. The pre-processing code locates the item in the XML-format catalog and determines the components that make up the item. XML transformation is used to take the catalog information and generate a message that contains the original order and component information. The message is sent to the integration node, where it is decomposed into individual orders to be sent to the suppliers.

Further processing would also be possible here, allowing execution of other processes on external systems.

OrderSession implements the Singleton pattern to store the queue connection factory JNDI name space lookup results for later use. The queue connection factory is used to make connections to queue destinations in WebSphere. The name space lookup is a significant overhead for applications, using the Singleton pattern results' better runtime performance.

- b. OrderProcessMessenger sends a reply back to the front-end server, acknowledging that the order has been placed and providing the order ID.

The order ID is generated from the message ID of the outgoing message from the back-end server to the integration server.

The front-end server receives the reply and redirects the user to the getOrderID.jsp, which shows the user the order ID. The ID is needed to pick up the order.

The integration server uses the order ID to store the order details in the database.

Integration (decompose)

6. The order processing flow picks up the message on the integration node. The message is decomposed into multiple orders. In our sample application, the original order is decomposed into three separate orders, labeled “red”, “green”, and “blue”.

The new orders are stored as separate records in a database under the same order ID. As the responses to the orders come back from the suppliers, the responses will be stored in the same database with the orders.

7. After the decomposition, the individual order requests are sent to the appropriate supplier queues where each supplier can pick up his request.

This same flow sends a message to OrderSession, acknowledging that the order has arrived.

Supplier application

8. On the supplier side, the suppliers pick up the message and send back a reply containing the item they can supply and the amount they can ship.

Integration

9. The incoming messages from the suppliers arrive at the integration node, where a processing flow picks them up. The flow stores the incoming messages from the suppliers in the database with the matching order.

An order is complete when all the suppliers have responded. In other words, each order in the database on the integration node has a reply from the suppliers stored with it.

Order pickup

This use case is responsible for the pickup process. The user comes to the corporate Web site to enter the order number (ID) that was returned when the original order was submitted. In return, the system comes back with the results based on the responses provided by the suppliers.

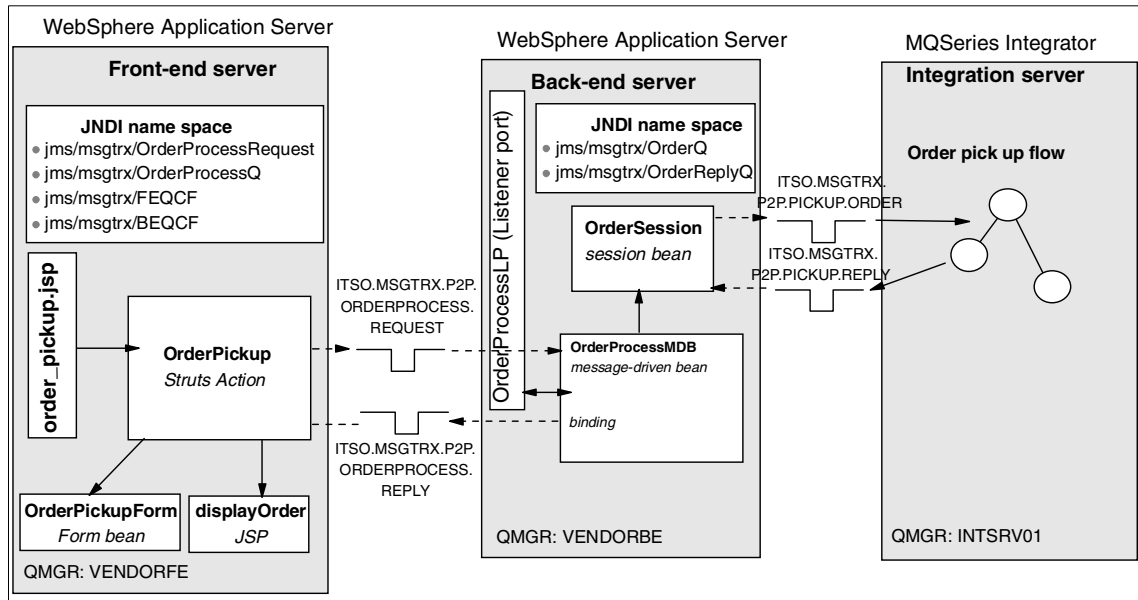


Figure 6-6 Order pickup flow

The order pickup process flows in the following way.

View

1. The customer goes to the IV Corp Web site to pick up an order, provides an order number, and submits the form.

Controller

2. The request invokes the OrderPickup Struts action. The action gets the form information from the OrderPickupForm object.
3. The front-end application sends a message to the back-end, the same way as it was described in the previous use case, using the message facade.

Model

4. The back-end server performs only message forwarding in this scenario, towards the Integration server. In a real scenario, the server would perform further processing or would execute other processes on external systems.

Integration (recompose)

5. The message is picked up by the order pickup flow on the integration server from the ITSO.MSGTRX.P2P.PICKUP.ORDER queue.

The flow collects the order parts from the database and composes the parts into one message, then the response is sent back to the back-end server to the ITSO.MSGTRX.P2P.PICKUP.REPLY queue.

Back-end applications

6. The back-end server forwards the reply to the front-end server with the results of the completed order.

Front-end results

7. The front-end server receives the order and compares it to the original order. The resulting page shows the original color and the color that can be supplied. If the order was fulfilled, the suppliers supplied the requested amount and the two colors are the same. If the order was not fulfilled, the suppliers could not provide the requested amount and the resulting color can be different, depending on the difference between the requested and the supplied substance amounts.

6.2 EJB design guidelines

General EJB design guidelines for self-service applications can be found in *Patterns: Self-Service Application Solutions using WebSphere V5.0*, SG24-6591. The guidelines here are specific to applications that use JMS messaging.

6.2.1 Local and remote home interfaces

In messaging applications, remote method invocations are often replaced using messaging to invoke remote functions. For example, JMS messages can invoke remote EJB methods via a message-driven bean, replacing the synchronous EJB remote method calls. Although remote method calls are not in use in such an application, EJBs in the same container should be able to call each other using an efficient method, making local calls.

The most common use of local EJB interfaces in messaging applications is to invoke session bean methods from message-driven beans. For example, in our sample application, the OrderProcessMDB bean calls the OrderSession bean using local interfaces.

Before the EJB 2.0 Specification, the only way to invoke an EJB, be it entity or session bean, was through its home and remote interfaces. Both interfaces assumed that the client was a remote client, so any call from the client to the EJB had to go through the RMI-IIOP invocation stack. While this provides great flexibility and portability in a multi-tier/distributed environment, when both the

caller and the EJB reside in the same JVM, this advantage became a significant performance overhead.

For performance reasons, EJB 2.0 introduced the local home interface and component local interface for entity and session beans. The local home interface and component local interface do not use the RMI-IIOP invocation stack but instead invoke the EJB directly from the client.

A session or entity bean can have remote interfaces, local interfaces or both, depending on the use of that bean.

The home interfaces let the user create, remove, and find (in the case of entity beans) EJBs. The result of these operations will be an instance or instances of component local interfaces to the EJBs.

Local home interfaces differ from remote home interfaces in such a manner that they can only be found in the local context. Only the object within the same JVM can locate this interface in the JNDI context.

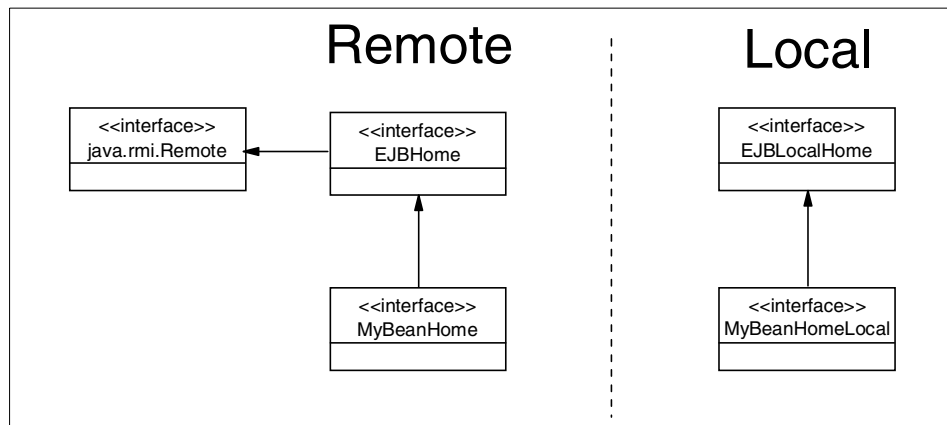


Figure 6-7 Local and remote component interfaces

Component local and remote interfaces

Clients never directly access instances of the bean's class. A client always uses the bean component interfaces to access the instance. When both the client and the bean instance exist in the same JVM, the client can use the component local interface instead of the remote interface to interact with the bean instance.

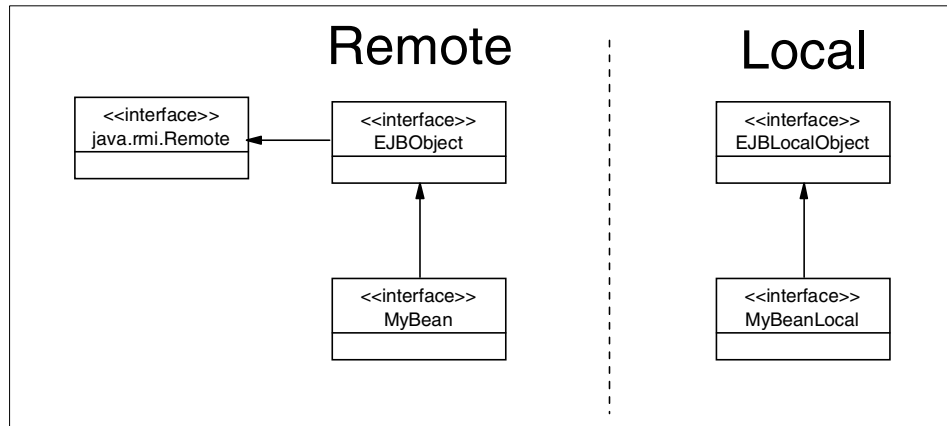


Figure 6-8 Local and remote objects

The difference between the two is that the remote interface implements the EJBObject interfaces, which is an extension of the java.rmi.Remote interface.

Implementing component and home interfaces

The local home interface extends the javax.ejb.EJBLocalHome class, and the component local interface extends the javax.ejb.EJBLocalObject class.

The container makes local and remote home interfaces available to the client through JNDI, as shown in the following example.

Example 6-1 Locating local and remote home interfaces

```

// Locating local home
InitialContext ic = new InitialContext();
myBeanLocalHome = (MyBeanLocalHome) ic.lookup("java:comp/env/ejb/MyBean");
MyBeanLocal myBeanLocal = (MyBeanLocal) myBeanLocalHome.create();

// Locating remote home
Properties env=new Properties();
env.put(Context.PROVIDER_URL,"iiop://ejbserver:2809");
env.put(Context.INITIAL_CONTEXT_FACTORY,
"com.ibm.websphere.naming.WsnInitialContextFactory");
InitialContext ic = new InitialContext(env);
Object o = ic.lookup("ejb/MyRemoteBean");
myRemoteBeanHome = (MyRemoteBeanHome) PortableRemoteObject.narrow(o,
MyRemoteBeanHome.class);
MyRemoteBean myRemoteBean=(MyRemoteBean) myRemoteBeanHome.create();
  
```

In our sample scenario, we use both local and remote interfaces for the Enterprise Beans:

- ▶ Entity beans use only local home and component interfaces, since they will only be called from a session bean (implementing a Session Facade pattern). This also eliminates the possibility of having unexpected clients interacting directly with the entity bean, since no remote interface for the entity bean is available.
- ▶ Most session beans implement only the remote home and component interfaces. This way, remote clients can interact with the session bean, using it as a facade to the business logic.
- ▶ If local objects want to interact with the session bean, they can use the remote interface, or we can implement the local interfaces for the bean. When both interfaces are implemented and are meant to look exactly the same, we can use a business model interface.

The business model interface will allow us to detect inconsistencies between the local and remote interfaces at build time. The bean, component local and component remote interfaces will implement this interface, allowing us to check the consistency at build time. Figure 6-9 shows how this can be accomplished.

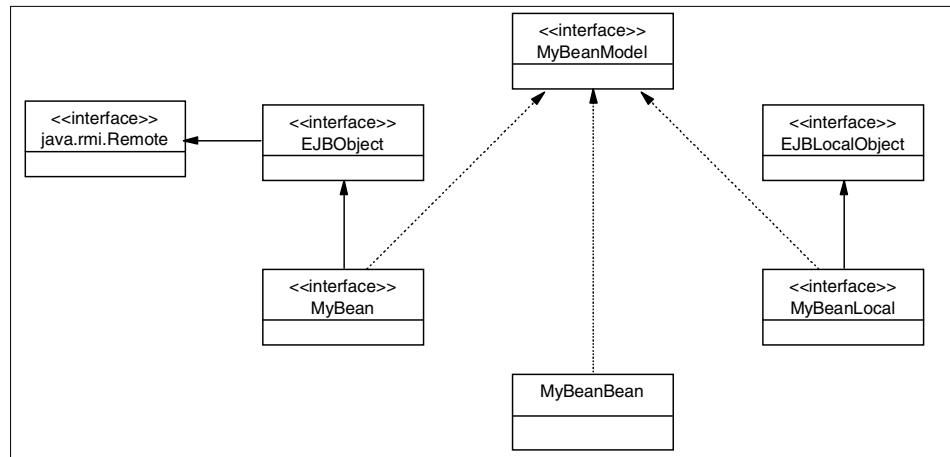


Figure 6-9 Class hierarchy for EJBs

In other cases, the local and remote interfaces do not have to be the same. The remote interface exposes only the method that can be used by a remote client, while the local interface does the same for the local clients, as shown in Figure 6-10 on page 109.

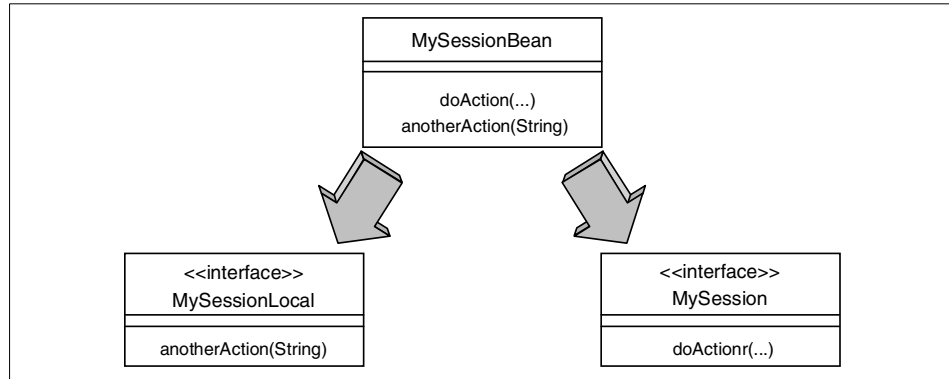


Figure 6-10 Local and Remote interfaces

6.2.2 Using the Singleton pattern

Sometimes it is important for a class to have one and only once instance. This is the case, for example, of a printer spooler or a caching helper class. This Singleton pattern ensures that a class has only one instance, and provides a global point of access to that instance.

The Singleton pattern implementation generally complies with the following characteristics:

- ▶ A public point of access for the class instance
Since we need to provide a global point of access to this unique class instance, the Singleton class usually provides a `getInstance()` method that gives access to the instance.
- ▶ No public class constructor
Since the Singleton class can only have one instance, we do not want any other object to be able to create an instance of this class directly. Instead, a private class constructor is provided that will be used by the class static method `getInstance()`.

The following code fragment shows a base Singleton class implementation.

Example 6-2 Singleton patter implementation in Java

```

class Singleton {
    private static instance = null;
    /**
     * We can also initialize the instance variable here
     * private static instance = new Singleton();
     */
}
  
```

```

private Singleton() {
    // perform some initialization process here
}

public Singleton getInstance() {
    if (instance == null) {
        instance = new Singleton();
    }
    return instance;
}
}

```

As shown in the previous example, the class instance can be created on demand, or it can be initialized during the class load process.

One of the most common use of the Singleton pattern in the J2EE platform is as a cache for JNDI references to resources.

For example, when an EJB is accessed by another object (that object can be another EJB, a servlet or any client class), the object has to locate the EJB Home (either local or remote) using JNDI to instantiate then invoke the given EJB. This lookup process can introduce a significant overhead in the EJB invocation process.

This can be avoided by using a helper class that implements the Singleton pattern. The Singleton object will look up the EJB Home and add a reference to the home interface.

The Singleton pattern can also be used to handle the JMS queue/topic connection process. This process is usually expensive because it requires a JNDI lookup for the ConnectionFactory and then the connection creation that actually establishes a connection to the underlying JMS Provider.

The sample scenario uses the Singleton pattern to store the connection factories for the session beans that are sending JMS messages, for example, the OrderProcess session bean.

6.2.3 The Facade pattern

The Facade design pattern provides an interface between two layers of an application. It has two major purposes:

- ▶ Hide the complexity of a subsystem by providing an easy-to-use interface.
- ▶ Reduce the number of interactions required with the subsystem in a distributed environment.

For example, some business operations may require multiple interactions with the application subsystem, such as creating records in a database, logging some information, and returning the results to the client.

A facade provides a single, simple method for the client, so the whole business operation will be executed with a single method invocation.

Using the Facade pattern

In a multi-tier J2EE application, the Model and the View layers might be distributed on two or more different application servers, and even in different machines across the network to provide better security and performance.

With the Facade pattern, we can minimize the number of interactions between the different layers of an application and, at the same time, provide better decoupling between these layers.

The Message Facade is a Facade pattern implementation using messaging for asynchronous interactions. In our scenario, we have used the Message Facade pattern to invoke remote EJB methods asynchronously via messaging. The caller component can simply send a message to the receiver component with the requested command to invoke together with the parameters for the call.

Using the Session Facade pattern

One of the most widely used implementation of the Facade pattern is the Session Facade pattern. The Session Facade pattern creates a facade for the Web container that hides the interaction with all the other beans in the EJB container.

Generally, a session bean will contain multiple business methods related to each other and each business method will represent a given use case.

Sometimes there is a need to decouple the client event from the business layer, so the client does not have to know the interface of the session bean explicitly to invoke it. There are several possible patterns that can help to deal with this situation, such as the Business Delegate pattern or the Command pattern.

The Business Delegate pattern provides a wrapper on top of the EJB interface, so any change in the EJB interface will not be reflected to the client. The EJB home lookup and EJB creation process are handled by the Business Delegate class, and any call to a method in the EJB is made through corresponding methods in the business delegate.

For further information about the Command pattern, go to:

<http://www-106.ibm.com/developerworks/patterns/index.html>

Using the Message Facade pattern

The Message Facade pattern helps to decouple the client completely from the EJB component that implements the business logic. The only contract required between the client and the business logic layer is the message format and semantics.

With this pattern, the client only needs to know the queue connection factory and queue to send the message to. On the business logic layer, a message-driven bean is listening to requests from that destination and will process them.

Generally, it is recommended that the message-driven bean not actually perform any business logic itself but instead delegate this process to a session bean, which acts as a facade again.

The sample application uses the message facade between the OrderProcess and OrderProcessMDB to invoke calls in the OrderSession bean. The same message facade is implemented between the OrderPickup and OrderProcessMDB beans.

6.3 JMS design guidelines

In this section, we focus on the roles of the J2EE Java Message Service (JMS) and WebSphere MQ in enterprise messaging applications. JMS applications are composed of the following parts:

- ▶ JMS clients are Java programs that send and receive messages.
- ▶ Messages are defined for each application and used for communication.
- ▶ A JMS Provider is a message system that implements JMS in addition to the other administrative and control functionality required for a full-featured messaging product.
- ▶ Administrated objects are preconfigured JMS objects created by an administrator for JMS clients. There are two types of administrative objects:
 - ConnectionFactory is used to create a connection with the provider.
 - Destination is used to access a source or destination of messages.

6.3.1 Message models

Each messaging model has a set of interfaces in JMS that define specialized operations for that model. There are two domains defined in the JMS specification for messaging applications:

- ▶ Point-to-point (PTP)
- ▶ Publish/subscribe (pub/sub)

Please see Figure 6-11 for the JMS class diagram.

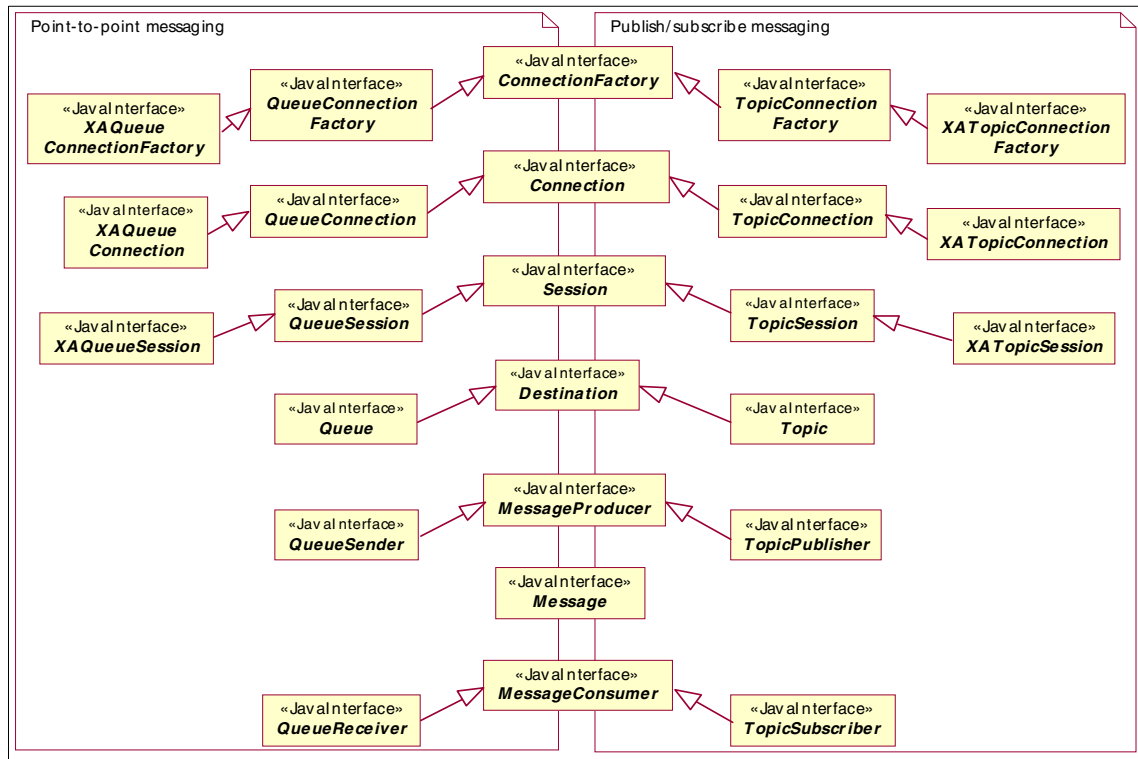


Figure 6-11 JMS classes

JMS is based on some common messaging concepts which are defined in JMS parent classes. Each messaging domain defines a customized set of these classes for its own domain. There are also classes defined that are transaction aware, like XAQueueConnection Factory.

The JMS parent classes define the following basic message concepts:

- ▶ ConnectionFactory is an administrative object used by a client to create a connection.
- ▶ Connection is an active connection to a JMS Provider.
- ▶ Destination is an administrative object encapsulating the identity of a message destination.
- ▶ Session is a single-threaded context for sending and receiving messages.

- ▶ MessageProducer is an object created by a Session for sending messages to a Destination.
- ▶ MessageConsumer is an object created by a Session for receiving messages from a Destination.

Not all JMS objects can be used concurrently. Table 6-1 shows the objects that can be used concurrently and those that cannot.

Table 6-1 Concurrent JMS classes

Object	Concurrent use
ConnectionFactory	Yes
Connection	Yes
Destination	Yes
Session	No
MessageProducer	No
MessageConsumer	No

There are two reasons for restricting concurrent access to Sessions. First, Sessions are the JMS entities that support transactions. It is very difficult to implement transactions that are multi-threaded. Second, Sessions support asynchronous message consumption. If a Session has been set up with multiple, asynchronous consumers, it is important that these separate consumers not execute concurrently.

6.3.2 JMS point-to-point model

Point-to-point (PTP) messaging involves working with queues of messages. The sender sends messages to a specific queue to be consumed normally by a single receiver. In point-to-point communication, a message has at most one recipient. A sending client addresses the message to the queue that holds the messages for the intended (receiving) client. You can think of the queue as a mailbox. Many clients might send messages to the queue, but a message is taken out by only one client. Like a mailbox, messages remain in the queue until they are removed. Thus, the availability of the recipient client does not affect the ability to deliver a message. In a point-to-point system, a client can be a sender (message producer), a receiver (message consumer), or both. In JMS, PTP types are prefixed with “Queue”.

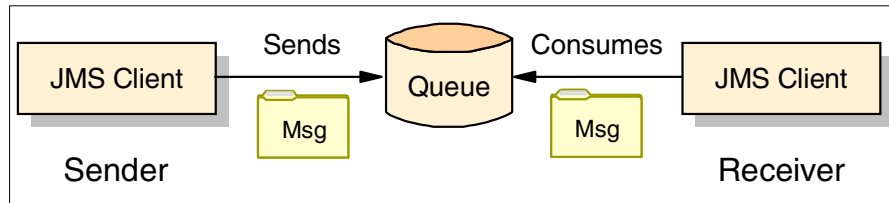


Figure 6-12 JMS point-to-point model

In point-to-point messaging, there are generally three messaging patterns:

- ▶ Request/reply
- ▶ Send-and-forget model or message producer
- ▶ Message consumer

We look at these messaging patterns in more detail in 6.3.5, “Synchronous versus asynchronous design considerations” on page 121.

6.3.3 JMS publish/subscribe model

In contrast to the point-to-point model of communication, the publish/subscribe model, shown in Figure 6-13, enables the delivery of a message to multiple recipients. A sending client addresses, or publishes, the message to a topic to which multiple clients can be subscribed. There can be multiple publishers, as well as subscribers, to a topic. A durable (or persistent) subscription, or interest, exists across client shutdowns and restarts. While a client is down, all objects that are delivered to the topic are stored and then sent to the client when it renews the subscription. A non-durable subscription will deliver messages when the consumer is connected, but discard messages when the consumer is not connected. In a publish/subscribe system, a client can be a publisher (message producer), a subscriber (message consumer), or both. In JMS, pub/sub types are prefixed with “Topic”.

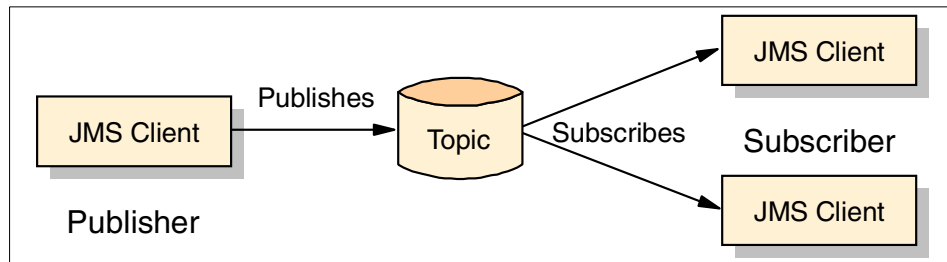


Figure 6-13 Publish/subscribe model

JMS also supports the optional durability of subscribers and “remembers” that they exist while they are inactive. All an application has to do is send information it wants to share to a standard destination managed by IBM WebSphere MQ publish/subscribe, and let IBM WebSphere MQ publish/subscribe deal with the distribution. Similarly, the target application does not have to know anything about the source of the information it receives.

Another important aspect of the pub/sub model is that there is typically some latency in all pub/sub systems. This is because messages observed by subscribers may depend on the underlying JMS Provider’s capability to propagate the existence of new subscribers and how long the messages are retained by the provider.

We look at the pub/sub pattern in more detail in 6.3.5, “Synchronous versus asynchronous design considerations” on page 121.

Integration hub concept

An integration hub provides a single interface for various services and applications. It significantly reduces the maximum number of interfaces needed to couple enterprise applications with each other. Without an integration hub, there may be up to $N \times N$ interfaces necessary to connect N different applications to each other. When using an integration hub, the same connectivity can be achieved with only N interfaces. With a hub in the middle, it is sufficient to connect each application to the hub. There must not be interfaces between the applications.

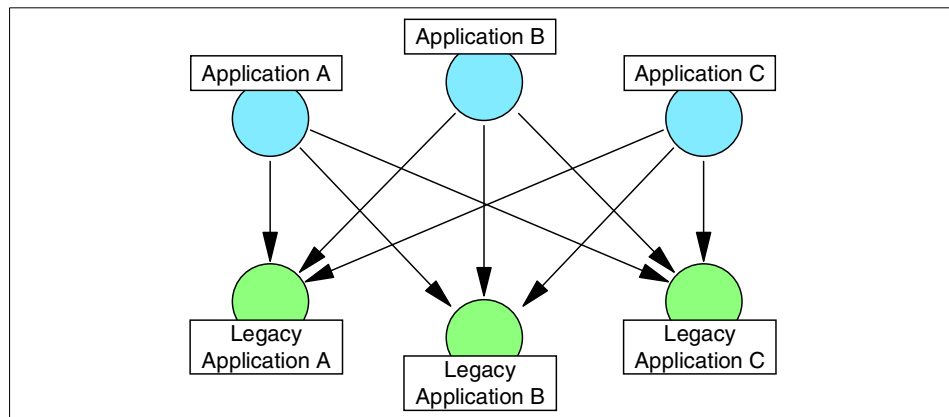


Figure 6-14 Stovepipe problem without dedicated integration layer

The number of interfaces in the enterprise is decreased by mapping the various interfaces to common abstract interfaces. This is particularly useful when the interfaces that are “wrapped” share common semantics:

- ▶ Different “back-end” applications of the same purpose can get a common (user) interface.
 - Integration of national or regional applications into a global self-service Web application.
 - Integration of information technology in a merged company.
 - Adaptation of suppliers’ information technology into the supply chain.
- ▶ Different access points to the same back-end technology.

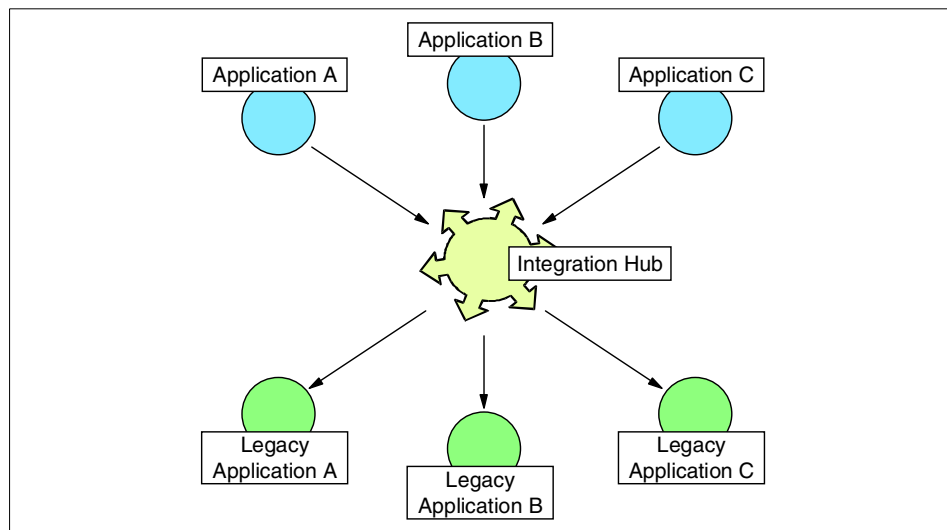


Figure 6-15 Integration hub

Integration bus topology

The integration servers in the previous scenario variations were built around the hub-and-spoke topology. In this way, all unique applications connect through a central hub. A new application only needs to be connected to the integration server to be integrated with other systems connected to the integration server. The integration server acts as a message broker to control the transport, data translation, and process integration among the connected applications.

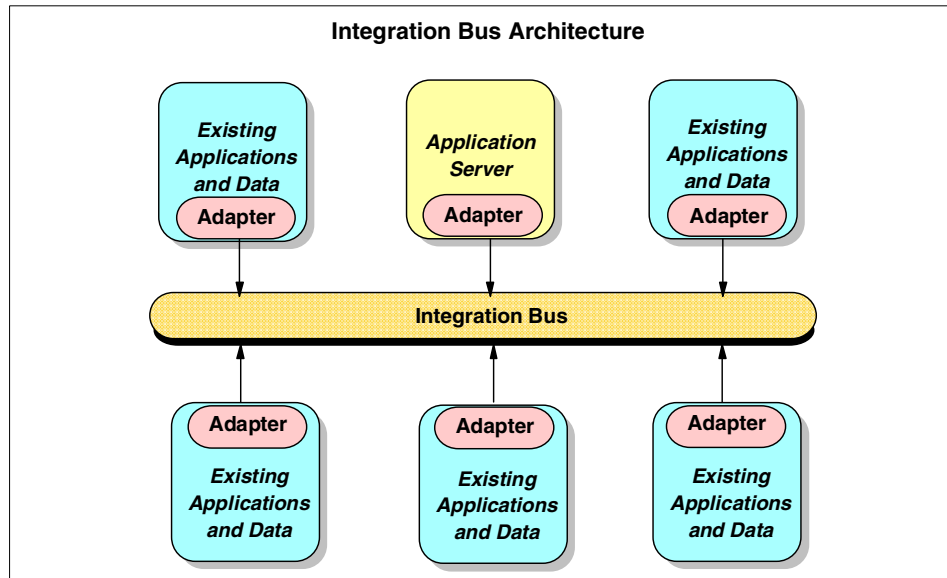


Figure 6-16 Integration bus topology

In addition to reducing the point-to-point integration, a hub-and-spoke integration server can be centrally managed, which simplifies the administration. The disadvantage of the centralized architecture is that it can create performance bottlenecks and also a single point of failure. We have to address these issues by adding multiple message brokers and servers, which will introduce additional architectural and administrative difficulties.

This is not the only model for integration servers. Another approach is to use the integration bus topology.

In an integration bus topology, all nodes are connected in a sequence in the direction of a shared communication backbone. Messages that are sent from interconnected systems go through the bus to the integration server, which manages the data transformation, transportation, translation, and finally routes to the receiving nodes.

The bus offers the means for messages to arrive at their destinations. The physical implementation of this architecture involves housing adapters within each integrated application, which then uses the integration bus communication backbone for interacting with the integration server and all other connected applications.

A comparison of the integration bus with the hub-and-spoke architecture shows that the integration bus architecture scales better and possibly offers better

performance. However, implementation of the integration bus architecture is more complex and more difficult to manage as the environment expands.

The choice of which architecture to use depends on other factors that demand an evaluation of the company. The types of applications, their usage, and the resources available should be considered. Companies that have limited IT resources and few systems with a moderate volume of transactions are better addressed with the hub-and-spoke architecture. On the other hand, if the company has plenty of IT resources and a large number of systems interacting with a high volume of transactions, the integration bus architecture is better suited.

When deciding upon an integration server, it is crucial to examine its architecture and to determine the impact it will have on the applications and the infrastructure of the company. The hub-and-spoke architecture can easily be implemented compared to the integration bus, but the scalability is more restricted. You should also consider the adapters for a particular application that the integration server might support and measure the work required to get those adapters working with the existing systems. In this case, process management support will be one of the most important features a company will implement to automate the business processes across multiple systems.

6.3.4 JMS messages

Another design choice is the JMS message type. As shown in Figure 6-17 on page 120, JMS messages are composed of the following parts:

- ▶ Header: contains information to identify and route messages.
- ▶ Properties: custom values that can optionally be added to messages. Properties can be:
 - Application-specific: properties added to messages, which are used by JMS applications
 - Standard: JMS properties
 - Provider-specific: properties that are specific to a messaging provider
- ▶ Body: the message data.

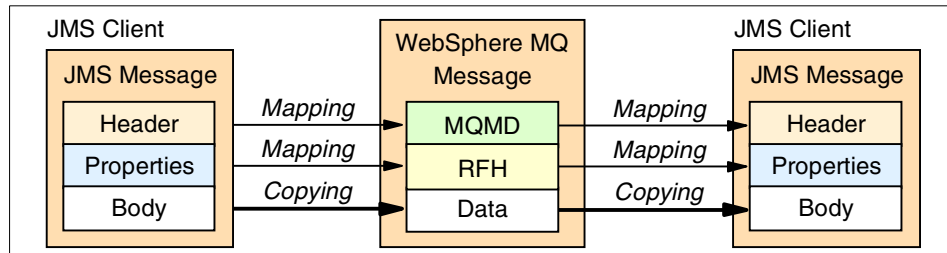


Figure 6-17 Message content

JMS provides different *message types*. Each contains specific interfaces pertaining to its content and allows specific operations on the messages.

The message types that can be used in JMS are:

- ▶ **BytesMessage:** contains operations for storing and accessing a stream of bytes.
- ▶ **StreamMessage:** contains operations for storing and accessing a stream of Java primitive values. It is filled and read sequentially.
- ▶ **ObjectMessage:** contains operations for storing and accessing a serialized Java object. If the application design requires more than one object to be serialized, then use a Collection object.
- ▶ **MapMessage:** contains operations for storing and accessing a set of key-value pairs from the message body. The keys must be strings and the values must be primitive types.
- ▶ **TextMessage:** contains operations for storing and accessing the body of a message as a string. Text messages can be used to store XML-data. This type of message can be used for sending messages to non-Java applications.

A couple of message settings are also important to consider:

- ▶ **Delivery mode:** when delivery needs to be assured by the business requirements, persistent messages are needed. But when this is not needed, performance can be gained by the use of non-persistent messages.
- ▶ **Message expiration:** when using non-persistent messages, message expiration can be used to discard messages that have remained on a queue or topic for longer than required. This prevents unprocessed messages from building up over time.

6.3.5 Synchronous versus asynchronous design considerations

In the Web application environment, choosing an asynchronous or synchronous approach to JMS communication will significantly affect the design of the application. The effects could ripple as far as the user interface interaction (or user experience), or it could affect only the low-level design and behavior of the underlying application.

Note: IBM WebSphere MQ is a fully asynchronous messaging system. To achieve request/reply requires coordination between the request queue and reply queue with the use of correlation IDs.

We look at both the user interaction differences and the system design considerations.

For the purposes of discussion, let us consider an example Web application that provides Web banking and needs to connect to an enterprise application that is hosting the bank account data.

First, it is important to go over some basic Web application principles. The Web is a *stateless* environment; typically, a request is received and the reply sent back immediately within the same client session. A Web server is not normally able to initiate a connection to a Web client out of the blue. Information about the requesting client is retained while the request is being serviced and not lost until a reply is sent back. The Web is a typical request/reply model. Most Web applications are built using this model and this style of user interaction where the user can expect a reply back from the server that will be the result of making a request.

Using our Web banking example, let us assume a Web request requires information from the enterprise application about the bank balance. The JMS interaction between the Web application and the enterprise application can be achieved using:

- ▶ Request/reply pattern
- ▶ Send-and-forget pattern
- ▶ Message consumer pattern
- ▶ Publish/subscribe pattern

Request/reply pattern

Using this approach, we fit the standard Web model by providing a complete round trip for the client request that results in a reply. The user does not have to visit another results page to see the results of his/her request.

As shown in Figure 6-18, the Web application sends a request message, then waits for a reply. The response message needs to be linked to the request message using the request message ID as the correlation ID of the response message.

The overriding factor in a request/reply pattern is the time delay before a reply comes back. You should remember that request/reply is a synchronous communication over an asynchronous transport. For request/reply, two queues are needed, one for the sender to send messages and one for receiving the responses back. The request/reply consists of two units of work:

- ▶ Putting the message on a queue
- ▶ Receiving the response and, for example, inserting the message in a database

These actions can never be one unit of work because the real put of the message only takes place after the commit. No message will be sent without a commit, and when no request message is sent, no reply will arrive.

An example of a request/reply scenario is getting your account balance. A message is first sent with the account ID, then the application waits until a response message is sent back with the balance of the account, with the results logged to an application database.

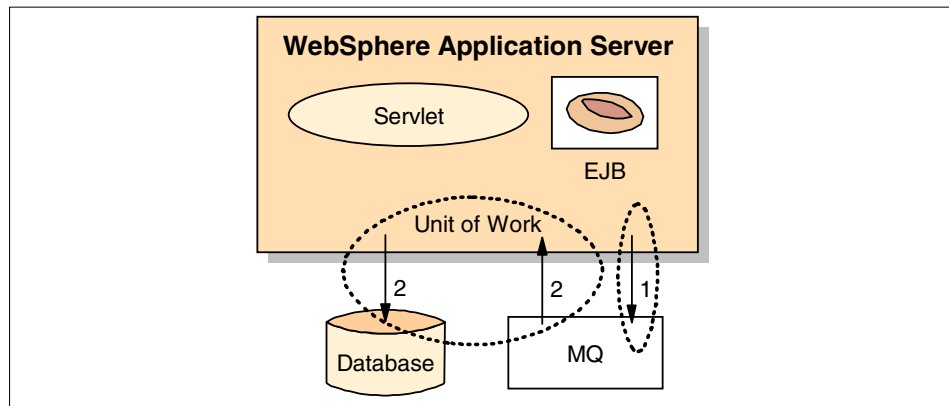


Figure 6-18 Request/reply pattern

Request/reply design considerations

Applications should not be designed without appropriate timeout or retry capability. Waiting indefinitely on a queue for the reply message to arrive will cause at best a poor user experience. At worst, it will result in a consumption of system resources to a point where the entire application will fail or stop responding. The application design should therefore cover delayed replies and

no reply at all. Non-persistent messages can become useful in these situations. If a reply has not been received within a given amount of time, an exception path is taken by the application, potentially resubmitting the request. When the response of the first request arrives, this response can be ignored. It is also possible to include a timeout time in a message. The message will then be destroyed when the timeout is reached. Typically, the use of non-persistent messages will also depend on the type of application and business requirements. For example, if the application is a transfer of monetary funds then it is quite likely that persistent messages will be used. When a failure occurs, a check back later algorithm is needed.

However, implementation of this approach presents a few important issues. Specifically, if the message producer is implemented as a session EJB, then in the request/reply JMS model, the EJB must wait (or block) until the enterprise application has replied before it can continue processing. Blocking in an EJB is not generally recommended because it restricts the EJB container's ability to effectively manage its resources. Care must be taken to ensure that the EJB is not waiting indefinitely and that there is a timeout in place.

Send-and-forget pattern

In Send-and-forget (or fire and forget), shown in Figure 6-19 on page 124, the Web application will initiate the request to the enterprise application using a JMS destination, but it will not wait for the outcome. This design has important repercussions on the user interaction. A message consumer pattern could be used for receiving the reply from the enterprise application. The user must at some point go to a result page to see his or her bank balance when it has been retrieved from the enterprise application.

From an implementation point of view, the blocking EJB dilemma is avoided. However, a new page is required to allow the customer to come back to check his/her last balance request from the local database. This design has alleviated the need for the blocking EJB design. However, the user experience is drastically different from the request/reply model.

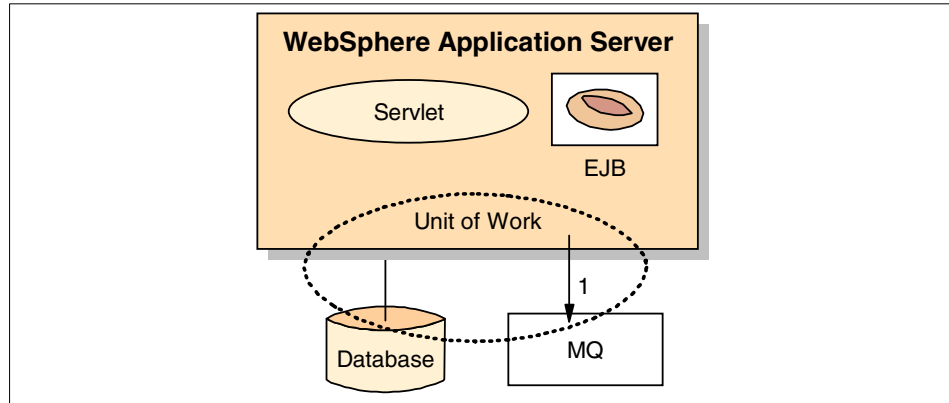


Figure 6-19 Send-and-forget pattern

Send-and-forget design considerations

One of the first things you should consider is whether you make use of persistent or non-persistent messages. Non-persistent messages do not survive process failure, but because their handling does not incur any disk I/O (for persistence), they can be processed much more quickly by the JMS Provider. The decision to use persistent or non-persistent messages will generally be governed by the business requirements. In the case of getting a balance, no funds transfer occurs; as such, a lost message has little impact and may not warrant persistent messages. The application may be sending the message as part of a unit of work (transaction), which implies that at some point either the application or the application server will commit the transaction to affect the send.

Another point to consider is the application component used to implement the message producer. The Model-View-Controller model suggests that access to an enterprise resource (messaging infrastructure) should occur via session EJBs. If you want the message sent as part of a transaction that will be coordinated by the application server (for example, send message and update a local database within a transaction), then EJBs are required. However, it is not uncommon to find the message producer being implemented in servlets; this offers a simpler programming framework (when compared to EJBs), but immediately precludes the use of global units of work, or access to any other features that the EJB container may offer.

Message consumer pattern

Message consumers can be implemented by message-driven beans, which are invoked by the container when a message arrives at a destination. When a message arrives at the destination, the EJB container passes the message to an instance of a user-developed message-driven bean.

This pattern can be used by a catalogue application receiving updates for changes in the online catalogue. In this scenario, a message-driven bean receives an incoming message and updates a database, as shown in Figure 6-20. The pattern is typically useful in a business-to-business situation where no user interaction is needed.

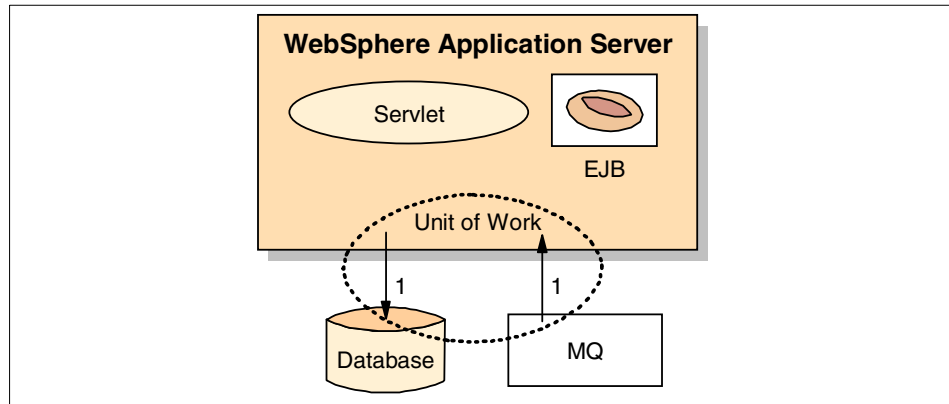


Figure 6-20 Message consumer

Message consumer design considerations

Use message-driven beans only to handle the message. Move the business logic to another bean that will be invoked by the message-driven bean. This way, it is also possible to call the business logic out of another channel like a servlet that has been activated by a user.

Message-driven beans cannot throw exceptions to the user, so exceptions have to be logged in a error report.

Publish/subscribe pattern

Using this approach, we can provide the user with an immediate reply without the user explicitly having to go to a separate Web page to see the results. However, this can only be achieved if a local copy of the data is used.

The Web application will register interest in information from the enterprise application upon startup. Periodically, the enterprise application will publish information to the subscribers (Web application). The message consumer pattern can be implemented at the subscriber site to receive the publications of the enterprise application. The Web application will store this information in a local database and use this information when a Web request is being serviced.

Using this approach, the Web application can operate in its native modes (stateless and request/reply) and the user can see the results of his request

within the same user transaction. However, the information may be slightly outdated.

Publish/subscribe design considerations

Never cache a non-durable subscription; use durable subscriptions instead.

For further information about how IBM WebSphere MQ can be used in the pub/sub model, refer to the publication *MQSeries Publish/Subscribe Applications*, SG24-6282.

Selecting a messaging pattern

None of these options is incorrect if *implemented* correctly. The user's requirements and experience will dictate which decision is the correct one.

A request/reply JMS communication model is ideal in a Web environment. However, if EJBs are to be the implementers of the enterprise access, care needs to be taken during implementation to prevent blocking calls from EJBs. If the user is willing to accept a different user interaction model, then asynchronous fire-and-forget is also an acceptable option. The middle ground could be achieved using full publish and subscribe; however, the accuracy of the information may be at stake.

Note: The request/reply blocking stateless EJB *must* be implemented such that appropriate timeout and retry conditions are applied.

The EJB 2.0 specification does point out that only one client will have access to an instance of a stateless EJB while it is servicing a client-invoked method. If, however, a blocking wait occurs for an indefinite period, the container may run short of available instances of the specific EJB to service other clients and thus slow down the overall performance of the application.

For further information, please refer to:

http://java.sun.com/products/jms/tutorial/1_3_1-fcs/doc/jmsj2ee.html

Client identifiers

JMS Providers must support the definition of a client identifier, which links a client application's connection to a message service with a state maintained by the message service on behalf of the client. By definition, a client identifier can only be used by one user at a time. Client identifiers are used in combination with a durable subscription name to make sure that each durable subscription corresponds to only one user.

The JMS specification allows client identifiers to be set by the client application through an API method call, but recommends setting it administratively using a connection factory administered object.

For deployed applications, the client identifier must either be programmatically set by the client application, using the JMS API, or administratively configured in the `ConnectionFactory` objects used by the client application.

Reliable messaging

JMS defines two delivery modes: persistent messages and non-persistent messages.

- ▶ **Persistent messages** are messages that are guaranteed to be delivered and successfully consumed once and only once. Reliability is the most valuable attribute for such messages.
- ▶ **Non-persistent messages** are messages that are guaranteed to be delivered at most once. Reliability is not a major worry for such messages.

There are two ways of assuring reliability in the case of persistent messages:

- ▶ The first is to assure that their delivery to and from a message service is successful.
- ▶ The second is to assure that the message service does not misplace persistent messages before delivering them to consumers.

Transactions/acknowledgments

Reliable messaging depends on promising that the delivery of persistent messages to and from a destination is achieved. This reliability can be obtained using either of two mechanisms supported by a session: transactions or acknowledgments.

If a session is created as transacted, then the production and/or consumption of one or more messages can be assembled into an atomic unit, a transaction. The client can commit the transaction if delivery of all messages is attained. However, if an anomaly occurs on the performing operations, all the operations in the transaction are undone or rolled back.

The scope of a transaction is in a single session; a transaction cannot extend across more than one client session.

A session can also be created as non-transacted. In that case, it will not support transactions. Instead, the session uses acknowledgments to guarantee that it had a reliable delivery.

From the point of view of a producer, this will imply that the message service will acknowledge the delivery of a persistent message to its destination before the producer's send method returns. From the point of view of a consumer, this will imply that the client will acknowledge the delivery and consumption of a persistent message from a destination before the message is deleted from that destination.

Persistent message storage

Another very important aspect of reliability is the guarantee that once persistent messages are delivered to their destinations, the message service does not delete them before they are delivered to consumers. This means that upon delivery of a persistent message to its destination, the message service must place it in persistent storage. If the message delivery fails for any reason, it can recover the message and deliver it to the appropriate consumers.

The message service should also store durable subscriptions. This is because, to guarantee delivery in the case of topic destinations, it is not sufficient to recover only persistent messages. The message service must also recover information about durable subscriptions for a topic. Otherwise, it would not be able to deliver a message to durable subscribers when they become active.

Messaging applications that are involved in assuring the delivery of persistent messages must use queue destinations or employ durable subscriptions to topic destinations.

6.3.6 Where to implement message producers and consumers

There are a number of options when considering where to implement your JMS message producers and consumers in the J2EE application architecture. We examine some of these options in this section.

Producers

If the Model-View-Controller (MVC) pattern is invoked, then the model is typically where the producer would be implemented. In J2EE application architecture, this is likely to be a session Enterprise JavaBean. However, it is possible to implement the message producer almost anywhere. A simple JavaBean could also implement the message producer and fit in with the MVC pattern.

If the producer is participating in a transaction of some kind, then session EJBs may be a better implementation choice. Transaction creation and management is gained almost freely within EJBs, whereas it would have to be explicitly created and managed within other implementation choices such as JavaBeans. In the same manner, EJBs typically will have access to other facilities or features within an EJB container.

Servlets can also be used as message producers. They offer a simpler programming model than EJBs. Servlets, however, are usually implementers of the controller aspect of the MVC pattern, and no advantages can be made of the container facilities for EJBs. A special case could be made for “utility or helper” servlets that are not being used as controllers.

Consumers

In the same way as producers, there are a number of implementation choices for consumers. When consumers are used in request-reply scenarios, it leaves the choice to implement this in a servlet or an EJB. When implementing the consumer in an EJB, there is the advantage of transaction management and security management of the container. The disadvantage is that an EJB thread will be occupied until a response arrives. Some extra programming is needed to disregard a response when it takes too much time.

Another option for the consumer is a message-driven bean. When using a message-driven bean, the request and reply will be loosely coupled, which makes it more complex. A message-driven bean is a good solution for subscription and message consumer patterns.

6.3.7 Message-driven beans

A message-driven bean (MDB) is an asynchronous message consumer. The `onMessage` method of the message-driven bean is invoked by the container on arrival of a JMS message on a queue. Rather than writing application code to poll for messages on a queue, you can use a message-driven bean instead. The main difference between message-driven beans and other enterprise beans is that message-driven beans have only a bean class. There is no home or remote interface for a message-driven bean. Message-driven beans can only be invoked by the container.

The main components used with message-driven beans are shown in Figure 6-21 on page 130. The deployed message-driven beans (MDBs1 to 4) are invoked by listeners. These listeners listen to the ports that are defined for the different destinations where other applications put their messages. Each listener port defines the association between a connection factory, destination, and a deployed message-driven bean. Another component is the message listener service, which includes a listener for each listener port, all controlled by the same listener manager.

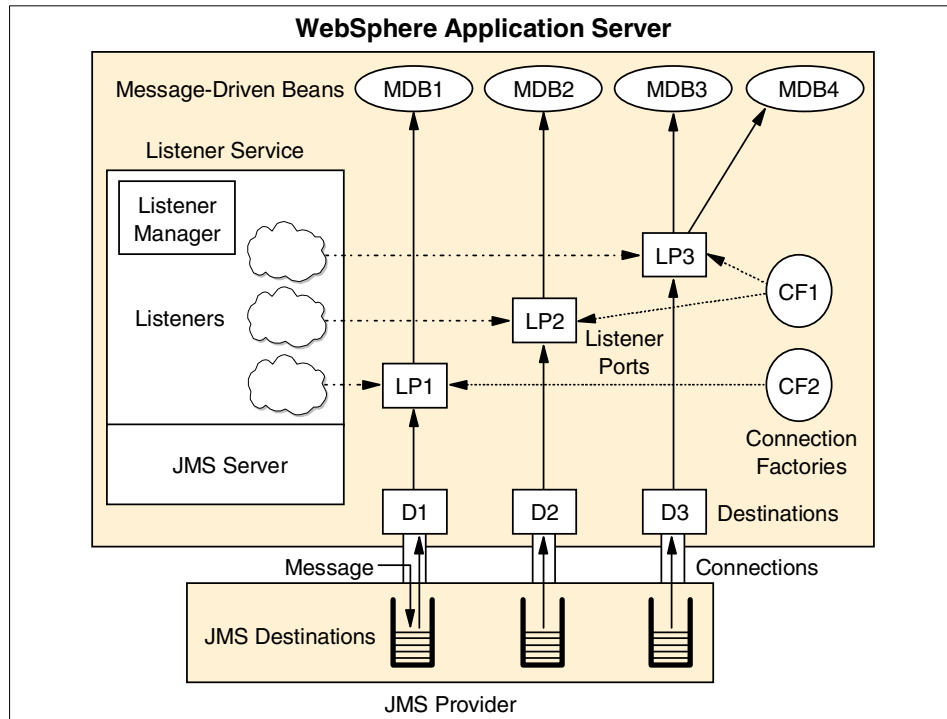


Figure 6-21 Message-driven bean

The message listener service is an extension to the JMS functions of the JMS Provider. It includes a listener manager, which controls and monitors one or more JMS listeners. Each listener monitors either a JMS queue destination (for point-to-point messaging) or a JMS topic destination (for pub/sub messaging).

The connection factory is used to create connections with the JMS Provider for a specific JMS queue or topic destination. Each connection factory encapsulates the configuration parameters needed to create a connection to a JMS destination.

When a deployed message-driven bean is installed, it is associated with a listener port and the listener for a destination. When a message arrives on the destination, the listener passes the message to a new instance of a message-driven bean for processing.

The listener manager is initialized at the start of an application server based on the configuration data. The listener manager creates a dynamic session thread pool for use by listeners, creates and starts listeners, and, during server termination, controls the cleanup of listener message service resources. Each

listener completes several steps for the JMS destination it is to monitor, including:

- ▶ Creating a JMS server session pool and allocating JMS server sessions and session threads for incoming messages
- ▶ Interfacing with JMS ASF (Application Server Facility) to create JMS connection consumers to listen for incoming messages
- ▶ If specified, starting a transaction and requesting that it be committed (or rolled back) when the EJB method has completed
- ▶ Processing incoming messages by invoking the `onMessage()` method of the specified enterprise bean

According to the EJB 2.0 specification, a transactional context may not be carried by a message, so an MDB will never execute within an existing transaction. However, a transaction may be started during the `onMessage` method execution if one of the following applies:

- ▶ The transaction attribute is "required" (container-managed transaction).
- ▶ It is explicitly started within the method (bean-managed transaction).

In the second case, the message receipt will not be part of the transaction. In the first case, the container will start a new transaction before de-queuing the JMS message (the receipt of which will thus be part of the started transaction) and enlist the resource manager associated with the arriving message, as well as all the resource managers accessed by the `onMessage` method. If the `onMessage` method invokes other enterprise beans, the container passes the transaction context with the invocation. The transaction started at the `onMessage` method execution can involve several operations such as accessing a database (via an entity bean or a JDBC data source), or sending messages (using a JMS connection factory). A message-driven bean instance has no state for a specific client. However, the instance variables of the message-driven bean instance can contain a state across the handling of client messages. Examples of such states include an open database connection and an object reference to an EJB object.

Message-driven beans design considerations

A message-driven bean should not contain any business logic. It is better to put the business logic in another enterprise bean, as shown in Figure 6-22 on page 132, which makes it possible to call the business logic from other components.

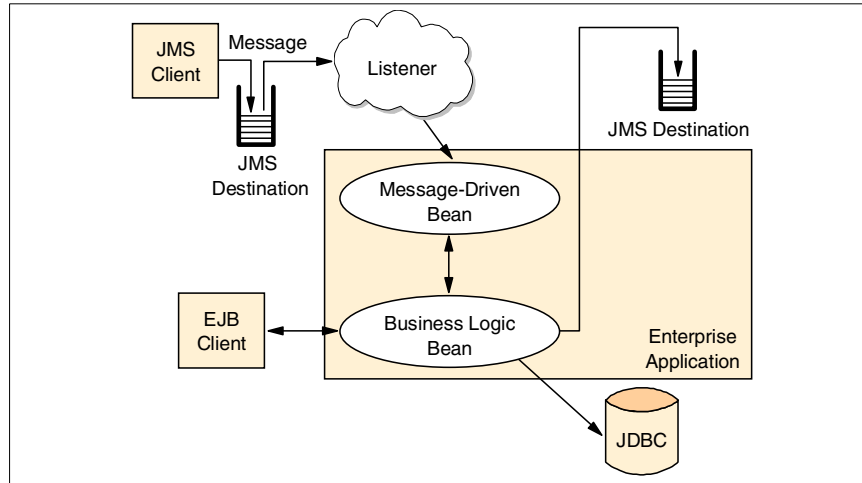


Figure 6-22 Message-driven bean development

Some other design issues are:

- ▶ The server does not know the client's security identity. Messaging does not propagate this identity to the message-driven bean. With this kind of bean, all instances are the same.
- ▶ Performance considerations: messaging becomes a middle layer between the client and the server. Even though message-driven beans are relatively lightweight, an extra layer can add time to your system response.
- ▶ Application complexity: an application that does not need asynchronous processing can be easier to code and debug.

Our sample application is using message-driven beans in the J2EE applications to consume asynchronous incoming messages from other applications, for example from other J2EE applications, or from WebSphere MQ Integrator flows. Message-driven beans were also used to implement the message-facade pattern for asynchronous remote calls, as discussed in “Using the Message Facade pattern” on page 112.

Transactions

Transactions for message-driven beans can be managed in two different ways:

- ▶ Container-managed transactions (CMT)
- ▶ Bean-managed transactions (BMT)

EJB container-managed transaction (CMT) considerations

In container-managed transactions, the EJB container manages the transaction demarcation. Container-managed transactions include all entity beans and any session or message-driven beans with a transaction type set to container.

In our sample scenario, we used container-managed transaction support. It is recommended that you use container-managed transactions for the application whenever it is possible. This not only makes development easier, but it also makes the application more reliable, since it is the container's responsibility to handle the transaction, not the programmer's. There are situations, of course, where the bean-managed transactions are necessary, if the developer cannot get around them and use purely container-managed transactions.

If the EJB-based application requires transactional support, then the container-managed transaction demarcation is the easiest and recommended approach. In addition to being supported by all types of EJB beans, it offers the following advantages:

- ▶ It is controlled declaratively, outside of the bean code, using the transaction attributes and the EJB deployment descriptor.
- ▶ It is set at the bean method level and forces method invocation to be either completely included or excluded from a transaction (unit of work).
- ▶ It simplifies programming, because there is no code to begin and end the transaction. Transactions do not need to be explicitly started and terminated.
- ▶ When used in the message-driven bean, it offers the only way to make the initial message, an MDB triggering message, part of a transaction.

To achieve this, the transaction attribute of Supported must be specified.

- ▶ It is the only choice for the entity beans.
- ▶ It offers a way to suspend an existing transaction and initiate a new one when a method is invoked.

To achieve this, the transaction attribute of RequiresNew must be associated with the invoked method.

Important: Care must be taken not to process the JMS Request/Reply type of scenario within the same transaction scope. The message is not delivered until the transaction is committed. Therefore, the synchronous receipt of a reply to such a message within the same transaction will not be possible.

Transaction attributes determine how transactions are managed and demarcated. The transaction attributes can be specified at the bean or at the method level. If present in both places, the method attribute will take precedence. The transaction attributes (EJB 2.0) are as follows.

- ▶ Required
- ▶ RequiresNew
- ▶ Mandatory
- ▶ NotSupported
- ▶ Supports
- ▶ Never

Note: Message-driven beans can only support the Required or NotSupported transaction attributes. This is a logical consequence of the fact that MDB is asynchronously driven by the container and cannot inherit a transaction from the client nor throw an exception for the client to handle.

Tip: The only way to make the triggering JMS message part of a transaction is to use the container-managed transaction for the MDB and specify the transaction attribute of Required. In this case, the container-initiated transaction includes the receipt of a message and the onMessage method in the same transaction scope. When onMessage successfully completes, the container commits the transaction and the JMS message is permanently removed from the queue. In the case of an onMessage failure or when rollback is requested, the container performs the rollback and the JMS message is re-delivered.

Table 6-2 Container-managed transaction semantics for message-driven beans methods

EJB method	Transaction Context at method invocation time
setMessageDrivenContext	Unspecified
ejbCreate	Unspecified
ejbRemove	Unspecified
onMessage	Determined by the transaction attribute

EJB bean-managed transaction (BMT) considerations

In bean-managed transactions, the code in the session or message-driven bean explicitly controls the transaction demarcation. Bean-managed transactions include any session or message-driven beans with a transaction type set to Bean.

Although the container-managed transactions are usually recommended for the reasons outlined in the previous section, the bean-managed transaction demarcation has its place, too.

In general, it offers an alternative, a programmatic rather than declarative approach to the transaction demarcation control. However, there are also some potential advantages. The main features of this approach are as follows:

- ▶ It is controlled programmatically from within the bean method code by issuing explicit Java `javax.transaction.UserTransaction` interface method calls.
- ▶ It offers “finer” granularity of transaction control than the container-managed options, at a Java statement level versus a method level, which could be beneficial in the following scenarios:
 - It may be desirable to have a method where a transaction is conditionally initiated or where only part of a method requires transactional protection (scope).
 - It may be desirable to initiate a transaction in one method and commit it in another method, which works only with the stateful session beans.
- ▶ It offers the ability to use JDBC type or JTA type transactions.
- ▶ Bean-managed transaction is only supported for the session bean and the message-driven bean. Entity beans can only use container-managed transaction.
- ▶ A message-driven bean instance `onMessage` method that has started a transaction must complete the transaction before returning; this is the same behavior as for the stateless session bean methods. If the transaction is not explicitly committed, it will be rolled back by the container.

Table 6-3 Bean-managed transaction semantics for message-driven beans

EJB Method	Transaction Context at the method invocation time	Can use <code>UserTransaction</code> interface methods?
<code>setMessageDrivenContext</code>	Unspecified	
<code>ejbCreate</code>	Unspecified	
<code>ejbRemove</code>	Unspecified	
<code>onMessage</code>	No	

6.3.8 Managing JMS objects

JMS Connection is the first point of access to JMS objects. JMS Connection is created from JMS ConnectionFactory. Once a connection is created, one or more sessions can be created in the context of the connection. JMS Sessions allow you to create message consumers and producers. When consumers or producers are created, the connection needs to be started to receive or send

messages. JMS Connections can be cached, in a similar way that EJB Home objects are cached, and reused by many clients.

JMS Sessions are designed for synchronous access only. A session can only be used by a single client and not shared among other clients. Similarly, an instance of either MessageConsumer and MessageProducer can only be used by a single client. JMS Sessions are opened for the duration of message sending or receiving; after this the session can be closed.

When a session is opened, the correct session acknowledgment must be selected from a performance perspective. In our sample scenario, we selected **AUTO_ACKNOWLEDGE**. This policy specifies that the message be delivered once and only once. The server must send an acknowledgment back, so the server incurs an overhead to implement this policy. The **DUPS_OK_ACKNOWLEDGE** setting resends the message until an acknowledgment is sent from the server. The server will operate in a lazy acknowledge mode, thereby reducing the overhead on the server but resulting in an increase in network traffic. With the most overhead of the three settings, **CLIENT_ACKNOWLEDGE** will cause the server to wait until a request for acknowledgment is sent from the client. Usually, the client calls the sent message's acknowledge method.

Upon completion of the interaction with the message producer or consumer (sender or receiver), the session needs to be closed. If the connection is closed, the session belonging to this connection is automatically closed as well.

The message producers/consumers must also be closed when you finish sending/receiving messages. Again, if the connection is closed, the producers and consumers are automatically closed.

Garbage collection of Java cannot be relied upon to clean out objects in a timely manner. It is always a good practice to call the close of any resource-bound object.

For further information, please read the JMS specification at:

<http://java.sun.com/products/jms/docs.html>

6.3.9 JMS and JNDI

The Java Naming and Directory Interface (JNDI) API implementation provides directory and naming functionality to programs developed in Java. This allows Java programs to discover and retrieve objects of any type from the JNDI name space.

JMS has two types of administered objects:

- ▶ ConnectionFactory
- ▶ Destination

An administrator can place objects of these types in the JNDI name space to be accessed by messaging applications.

Figure 6-23 shows the role of JMS and JNDI relative to a Java application. These two APIs sit above any specific service providers and encapsulate any vendor-specific information.

As a result, a developer using these technologies in a messaging-enabled application need only be familiar with the APIs, not the specific messaging systems.

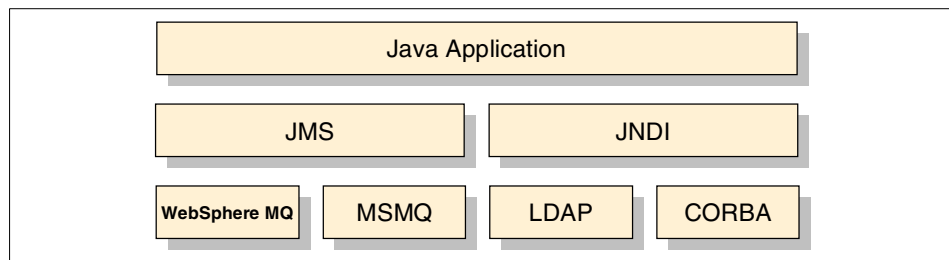


Figure 6-23 The role of JMS and JNDI relative to an application

So, how does an administrator put these objects in the JNDI name space? This step is vendor-specific. If you are using WebSphere MQ V5.3 with WebSphere Application Server V5.0, you can administer these objects right from the WebSphere Administrative Console. If you are using another application server, WebSphere MQ V5.3 provides a tool called JMSAdmin for this purpose.

6.3.10 Embedded JMS Provider versus WebSphere MQ

In line with the J2EE 1.3 specification, WebSphere Application Server V5.0 has an embedded JMS Provider, or messaging service, included in the application server. This internal JMS Provider can be used for asynchronous JMS communications with other WebSphere applications. The internal messaging service cannot be used for messaging with other messaging system, such as WebSphere MQ. If you need to communicate with other systems using WebSphere MQ, then you need to install WebSphere MQ as a JMS Provider on a WebSphere Application Server.

6.3.11 WebSphere to MQ connection options

A message placed on an IBM WebSphere MQ queue from an application server may originate directly from a servlet, or may be sent from a command bean or EJB (we recommend the latter two methods, not so much using servlets). Regardless of the method used, the messages are sent to a queue manager using one of the two available WebSphere MQ Java APIs by IBM WebSphere MQ. Each API has certain characteristics that make it appropriate for a situation, depending on the priorities you have. However, the API chosen can have an effect on the options you have for distributing the application components.

The two APIs that we discuss here are:

- ▶ The IBM WebSphere MQ for Java Message Service package, `com.ibm.mq.jms.jar` and `com.ibm.jms`

IBM WebSphere MQ for JMS classes implements the J2EE Java Message Service (JMS) interface to enable JMS programs to access a subset of IBM WebSphere MQ features from a vendor-neutral point of view, as defined by the JMS specification. The JMS interface is implemented by a set of IBM WebSphere MQ classes for JMS.

- ▶ The IBM WebSphere MQ for Java package, `com.ibm.mq.jar`

IBM WebSphere MQ for Java classes enable Java applets, applications, servlets, and EJBs to issue direct calls and queries to IBM WebSphere MQ using specific calls designed to take advantage of IBM WebSphere MQ features.

A JMS Java application uses the vendor-independent JMS interfaces to access the MQ-specific implementation of the JMS classes.

A key idea in JMS is that it is possible, and strongly recommended, to write application programs that use only references to the interfaces in `javax.jms`. All vendor-specific information is encapsulated in implementations of:

- ▶ `QueueConnectionFactory`
- ▶ `TopicConnectionFactory`
- ▶ `Queue`
- ▶ `Topic`

Coding outside the JMS interface to access WebSphere MQ-specific features will, of course, reduce the portability of the application, since it is now directly referencing WebSphere MQ-specific classes. If application portability, vendor independence, and location transparency are of importance, pure JMS is the obvious choice. JMS uses abstracted concepts of messaging to provide a vendor-independent API to messaging, while underneath lies the IBM WebSphere MQ implementation of the JMS interfaces. The real-world entities that are IBM WebSphere MQ queue managers and queues are accessed by

JMS clients through the use of the Java Directory and Naming Service (JNDI). The IBM WebSphere MQ entities are published to JNDI from the WebSphere Administrative Console, or through a tool called JMSAdmin. MQ JMS supports both the point-to-point and publish/subscribe models of JMS.

MQ base JMS classes provide two connection options to IBM WebSphere MQ:

- ▶ Bindings mode to connect to a queue manager directly
- ▶ Client mode using TCP/IP to connect to a queue manager (not supported on z/OS™ or OS/390®)

Both support connection pooling.

Java bindings mode

In binding mode, also known as server connection, the communication to the queue manager utilizes inter-process communications. One of the key factors that should be kept in mind is that binding mode is available only to programs running on the WebSphere MQ server that hosts the queue manager.

The key connection parameter in this case is the queue manager name.

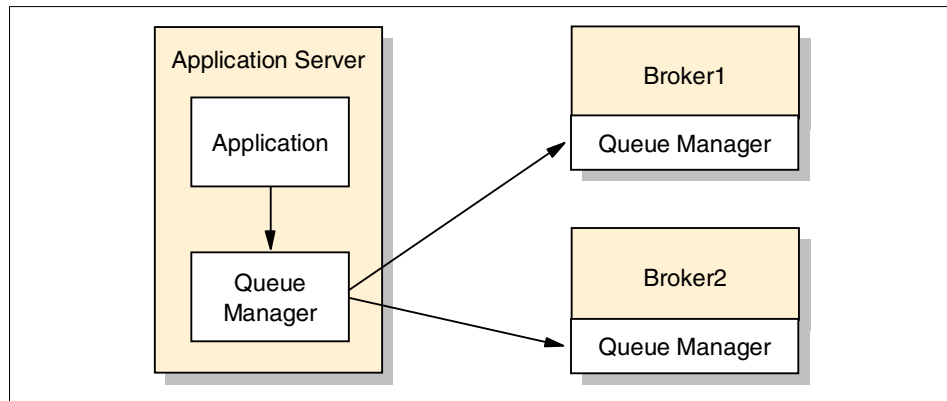


Figure 6-24 Java bindings mode

Connecting to the local queue manager has several major advantages:

- ▶ The probability of establishing a connection to a queue manager in your own host is high, as opposed to a connection with a remote queue manager.
- ▶ The time it takes to establish a network connection to the queue manager is bypassed.
- ▶ The local queue manager can distribute the work among multiple brokers. If connection performance is a high priority in your network, then using bindings mode is the clear choice.

Using bindings mode, you can also use WebSphere as an XA resource coordinator for units of work that involve WebSphere MQ updates and database updates, for databases and drivers that support the XOpen/XA standards.

Java client mode

Client connection uses a TCP/IP connection to the WebSphere MQ Server and enables communications with the queue manager. Programs using client connections can run on a WebSphere MQ client machine as well as on a WebSphere MQ server machine. Client connections use client channels on the queue manager to communicate with the queue manager. The latest version of the client supports XA transaction coordination by the queue manager.

Both the WebSphere MQ classes for Java and the WebSphere MQ classes for JMS are needed to use WebSphere MQ client mode. The key connection parameters are the host name, TCP/IP port, and server connection channel name. If your code is using only the JMS interfaces (for maximum portability), then client mode cannot be achieved since there are no methods exposed in the JMS interface to select a host or port number.

The client mode is best used when you do not want IBM WebSphere MQ to reside on the same machine as the application server. It allows you to connect directly to a remote IBM WebSphere MQ queue manager.

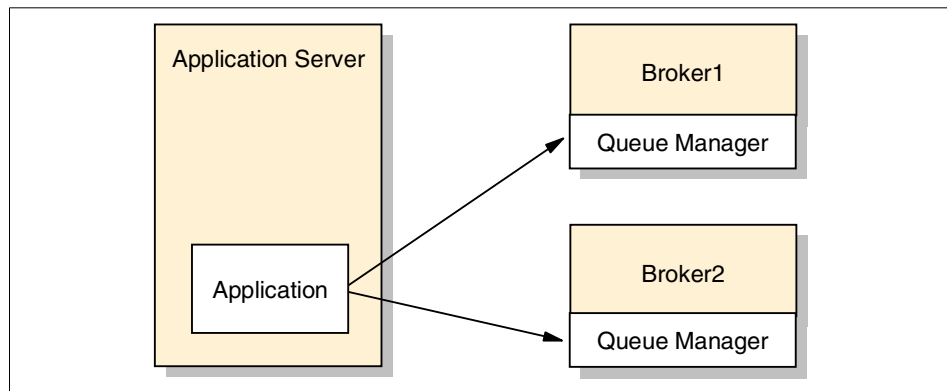


Figure 6-25 Client mode to remote brokers

When you connect directly to a queue manager on a broker, as in Figure 6-25, you relinquish any workload distribution the queue manager offers. The application must decide which broker to send the work to and any workload distribution would have to be done in the application itself, which is not recommended. Even having the queue managers in a cluster does not help, since a queue manager will always send the work to the local instance of the broker. Another pitfall is that XOpen/XA facilities for coordinated commits to

WebSphere MQ/JMS and databases are no longer available using WebSphere as the transaction coordinator.

One way around this is to connect to a remote queue manager that does not have a broker instance, but is there purely for workload distribution, as shown in Figure 6-26.

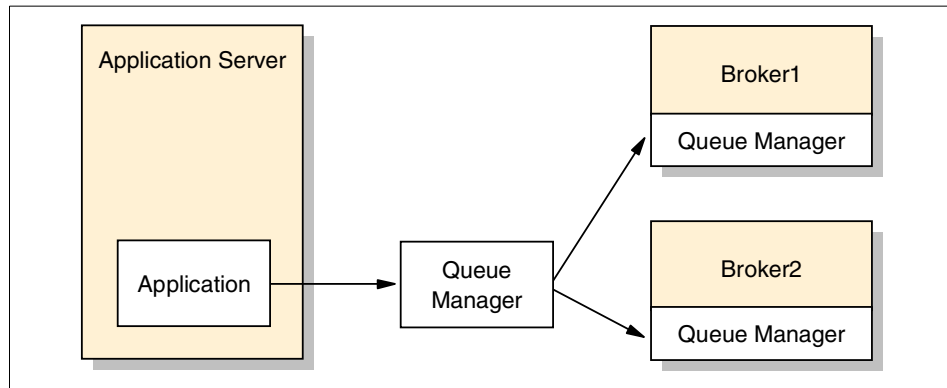


Figure 6-26 Client mode to a remote queue manager

You still have to deal with the network connectivity time; in fact, you have made it a little longer by introducing an intermediate system. But you do have the advantage of the queue manager workload distribution and the ability to connect to a remote queue manager. Yet another method would be to use TCP/IP load balancing, but this is not a function of WebSphere MQ.

The client mode can also be used to connect to the local queue manager by passing through the internal TCP/IP stack. This is obviously not as efficient as using the bindings mode, but it does allow your program to be used in a generic environment where you do not know whether the queue manager will be local. You can also make the different connection options parameter-driven so that the application is ready no matter what the connection type. You do need to ensure that the correct parameters are passed so you get the connection type you desire. A database table would be a good place to store these if you are interacting with a database.

Both MQ JMS classes (not the pure JMS interface) and MQ based Java allow you to put messages from the Java application in WebSphere directly into the remote broker's queue. If you are thinking of doing this, you should consider the performance implications. The cost of creating a network connection is added to the total cost of each request. For each request, an IBM WebSphere MQ-to-client session is created. There is no long-lasting network connection. This will impact the ability to run thousands of sessions in parallel. If you create a local IBM WebSphere MQ session for each request, the overhead will be much

lower. The network connection is now maintained by a sender-receiver channel pair and is long running. Ideally, long-running IBM WebSphere MQ sessions are preferable.

IBM WebSphere MQ clustering

WebSphere MQ offers the ability to create clusters. MQ clusters provide a number of benefits that are silently utilized by JMS applications. Clusters offer:

- Simpler administration of logically related queue managers

Clustering allows communication between queue managers to publish information about the queues they offer. Once in a cluster, queues on remote queue managers are visible to all queue managers if the queues are defined as cluster queues. The number of explicit definitions within IBM WebSphere MQ administration is reduced with the use of clusters.

- Workload management

Adding queue managers to clusters allows access to WebSphere MQ workload and failover features.

As shown in Figure 6-27, QM3 is able to load balance across the queue named ReplyQ, since it is available on both QM1 and QM2. Similarly, if QM1 is disabled, all messages for ReplyQ are routed to QM2.

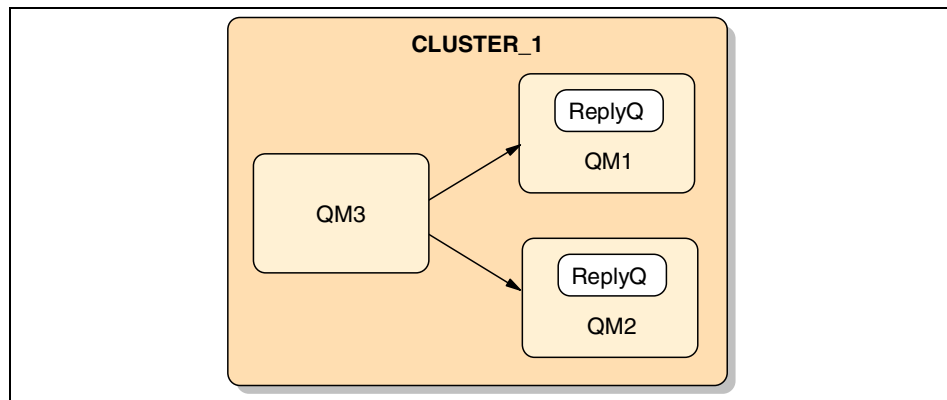


Figure 6-27 Cluster workload management

None of these features can be controlled through the JMS interfaces. However, MQ will automatically utilize the workload and failover under JMS.

These and other features of MQ offer significant benefits and demonstrate that IBM WebSphere MQ is a reliable, scalable, and mature JMS Provider.

6.3.12 Best practices for JMS and IBM WebSphere MQ

This section briefly discusses a number of issues and common best practices for JMS and IBM WebSphere MQ.

Design level

Application design level best practices for JMS and IBM WebSphere MQ include the following.

- ▶ Use message timeouts

Using message timeouts avoids large numbers of messages remaining on a queue, thus reducing performance overheads. It also allows a relevancy aspect on the messages. For example, a message may not be relevant after a certain period of time because the information in the message has been superseded.

- ▶ Use message selectors

JMS provides a useful API that can reduce the need for complex message browsing code. Message selectors allow the underlying provider (through JMS) to browse messages before they are retrieved with little application code.

- ▶ Persistent versus non-persistent

The use of durable messages is often necessary when communication takes place between, for example, systems of a financial nature. However, if messages do not need to be persisted, then messages need to be explicitly set to non-persistent, since the default in JMS is persistent.

- ▶ Clusters

The use of IBM WebSphere MQ clusters allows simple administration, as well as automatic workload management and failover.

- ▶ Message producers

EJB message producers: in a request/reply scenario, it is important that the issue of blocking calls be dealt with correctly. Essentially, EJBs should only be used with appropriate request/reply timeouts and retries.

- ▶ Message consumers

Message-driven beans: business logic should not be implemented in the message-driven bean. Implement the business logic in another component, such as a session bean, and use message-driven beans only for receiving the message.

Never throw application exceptions in the `onMessage` method.

- ▶ Consider using XML-based messages for inter-application integration
XML is commonly used as a messaging structure that allows for a more portable inter-application integration model. Although it does add some overhead to the message payload size and requires XML parsers, it is quickly becoming a standard for inter-operability.

Code level

Application code level best practices for JMS and IBM WebSphere MQ include the following.

- ▶ Explicitly close JMS resources that are no longer required
Java garbage collection alone cannot release all IBM WebSphere MQ resources in a timely manner. This is especially true if the application needs to create many short-lived JMS objects at the Session level or lower. It is therefore important to call the close() methods of the various classes (QueueConnection, QueueSession, QueueSender, and QueueReceiver) when the resources are no longer required.
- ▶ Handling errors
Any runtime errors in a JMS application are reported by exceptions. The majority of methods in JMS throw JMSEExceptions to indicate errors. It is good programming practice to catch these exceptions and report them to suitable output.

Unlike normal Java exceptions, a JMSEException may contain a further exception embedded in it. For JMS, this can be a valuable way to pass important details from the underlying transport. In the case of MQ JMS, when IBM WebSphere MQ raises an MQException, this exception is usually included as an embedded exception in a JMSEException.

The implementation of JMSEException does not include the linked exception in the output of its toString() method. Therefore, it is necessary to check explicitly for an linked exception and print it out, as shown in Example 6-3.

Example 6-3 Checking for JMSEException

```
try {  
    // code which may throw a JMSEException  
    ...  
} catch (JMSEException je) {  
    System.err.println("caught "+je);  
    Exception e = je.getLinkedException();  
    if (e != null) {  
        System.err.println("linked exception: "+e);  
    }  
}
```

- **Exception listeners**

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to `receive()` methods. To cope with this situation, it is possible to register an `ExceptionListener`, which is an instance of a class that implements the `onException()` method. When a serious error occurs, this method is called with the `JMSEException` passed as its only parameter. Further details are in the J2EE JMS specification.

Naming

Each WebSphere MQ network has one or more instances of a queue manager known by a name within the network of interconnected queue managers. For all other object types, each object has a name associated with it and can be referenced by that name.

Since names make up the API of the messaging system, names are not a pure matter of taste or tradition. Names should be considered as tools. We should think about names in terms of:

- Ergonomics or ease of use
- Error rates
- Development costs

Namespaces

A queue manager's name must be unique on the physical machine where the queue manager resides. However, it is obviously better to use a queue manager's name only once in a network, although a good reason to use a queue manager name twice or even more often in one network is to have a stand-in for high-availability purposes.

Object names must be unique within one queue manager and object type. For example, you can have a queue and a process with the same name, but you cannot have two queues with the same name in a single queue manager.

Consistency

The most important element in naming is consistency. In particular, when you design queue names, consider carefully the application programming interface (API) of the system, which is used and known by client applications. Name changes can be very expensive, and often there will not be much of a chance to change a naming policy once the system is in productive use.

Some general rules of thumb for naming

Manydiscussed specialists consider naming rules to be a topic of the utmost importance. Here we discuss only one possible set of rules, and they are not mandatory. The most important rule you have to remember is: be consistent!

- ▶ Adopt a set of workable standards and conventions. We recommend the SupportPac™ found at:

<http://www-3.ibm.com/software/integration/support/supportpacs/individual/md01.html>

- ▶ Develop common naming conventions across the enterprise.
- ▶ Plan ahead to prevent changing names in production environments.
- ▶ Use meaningful names whenever possible. Within namespaces, there is no implied internal structure, in particular no hierarchies, since WebSphere MQ just compares strings. Nevertheless, you should use hierarchical orders whenever possible.
- ▶ Be consistent with related names. For example, you should not call a queue manager ROME if it resides on a machine named ROMA, nor LISBOA when its machine is called LISBON.
- ▶ Abbreviations are always a source of confusion, unless they are very well known. It is better to be verbose whenever possible.
- ▶ Queue managers' names should be short, although they are allowed to be 48 characters long. We recommend a length of no more than eight characters, for the reason described in the following paragraph. On z/OS, a restriction to four characters applies since the queue manager has to correspond to a subsystem of the same name.

Channel names should reflect source and target queue managers of the channels and allow additional qualifier, flag-encrypted or compressed data transfer. Since channel names are restricted to 20 characters in length, this implies the use of short names for the queue manager. ROME/LISBON would be fine.

- ▶ Use underscores to separate suffixes, that is, to distinguish different versions:

PARTY.INVITATION_V21

- ▶ Transmission queues have exactly the same name as the queue manager they are pointing to. Transmission queues for additional channels may be qualified with a suffix.
- ▶ Do not use lowercase letters in names. Object names in WebSphere MQ are case sensitive. We recommend you use uppercase letters only for naming.
- ▶ Do not include types in names. Types may be subject to change, while names are not. Using such names as LQUEUE.SOME.REQUEST or QREMOTE.SOME.RESPONSE is strongly discouraged. Using such names would reveal more internal details to client applications than you need to. Remote queues may soon become local or vice versa. Remember that both MQSC and the graphical Explorer interface allow you to sort or filter objects by type. Thus, there is no need to include object types in names.



Application development

In this chapter, we discuss how to develop a solution using the technologies explained in previous chapters. Our scenario shows a simple, applicable situation where some of the technologies, and mostly the Self-Service and other patterns, can be applied to a real-life situation.

This chapter provides guidelines for developing the sample application, rather than a detailed description of the development process using a specific tool or technology.

7.1 MVC development using the Struts framework

This section describes the development cycles for the Model-View-Controller (MVC) design pattern using the Apache Struts framework.

7.1.1 Creating a Web diagram

WebSphere Studio Application Developer V5 has a built-in plug-in that supports the development of Web applications using the Struts framework. It has two major features:

- Visual Web diagram editor for Struts elements.

You can develop interaction diagrams and Struts applications based on these diagrams. The diagrams look like the one in Figure 7-1.

- Special editor view for the Struts configuration .xml file.

This special editor view simplifies development. Instead of editing the XML source, the developer can create and configure the Struts elements interactively in the editor view.

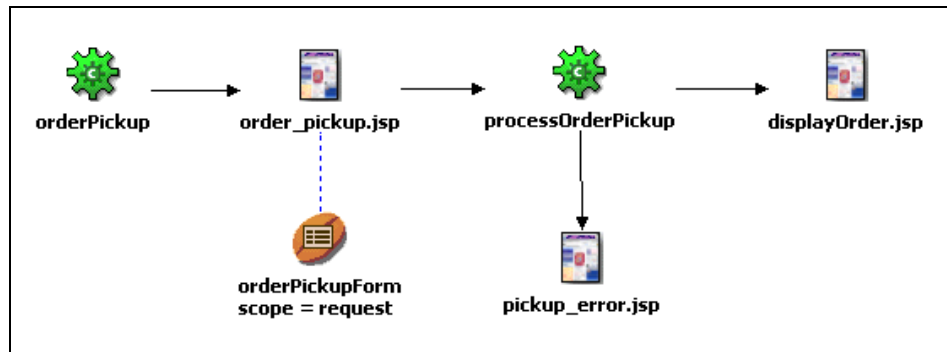


Figure 7-1 Struts Web diagram - OrderPickup process

7.1.2 Coding Struts elements

The following steps are an outline of how a sample Struts flow is created.

1. Create a Web application. In WebSphere Studio, this is done by creating a Web project (**File -> New -> Web Project**). The wizard will allow you to select Web project features. Be sure to include “Add Struts Support”. This will create a Web application with the necessary framework for Struts, including the Struts tag libraries, Struts runtime libraries, Struts configuration file and entries to support Struts in web.xml, and the application resources properties file.

2. Create an input page with an HTML form, for example, input.jsp. Make it a JSP so it can provide dynamic information, for example, using JSP taglibs, or provide error messages when validation fails.

The WebSphere Studio wizard that creates JSPs (**File -> New -> JSP File**) gives you the option of selecting a model on which to base the JSP. Be sure to select **Struts JSP** for this JSP and the next two JSPs. This will include the taglib directives to the Struts tag libraries.

Example 7-1 input.jsp

```
<%@ page language="java" %>
<!-- include all the JSP Taglibs needed -->
<%@ taglib uri="/WEB-INF/tlds/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/tlds/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/tlds/struts-form.tld" prefix="form" %>
<html>
<body>
<!-- print out the errors, if any -->
<html:errors/>
<!-- create a form -->
<html:form action="/sampleaction.do" focus="userid">
<!-- get the value for prompt.name from the applicationresources.properties -->
<bean:message key="prompt.name"/>:
<!-- create a text input field -->
<html:text property="name" size="20">
<!-- create a submit button -->
<html:submit>Submit</html:submit>
</html:form>
</body>
</html>
```

3. Create a Results page with dynamic content, for example, results.jsp.

Example 7-2 results.jsp

```
<html>
<body>
This is the result page.
<!-- Check out the Struts documentation on how to use the form-bean here to
dynamically retrieve information from the bean. Also check the Struts taglibs
-->
</body>
</html>
```

4. Create an error page to indicate errors, for example, error.jsp.

Example 7-3 error.jsp

```
<html>
<body>
This is the error page.
<!-- Check out the Struts documentation for Struts taglibs -->
</body>
</html>
```

5. Create a form bean to store the form data for the action, for example, `com.ibm.itso.msgtrx.SampleForm`. The `FormBean` provides access to the fields within a form submitted on the input page; it also provides validation if implemented.

WebSphere Studio provides wizards to create Struts elements, including form beans (**File -> New -> Other -> Web -> Struts -> ActionForm Class**). The wizard to create `ActionForm` classes allows you to select fields from JSPs for which you want field, setter and getter methods. For example, you would choose the input fields on `input.jsp`. The wizard also automatically registers the form bean in the `Struts-config.xml` file.

Example 7-4 com.ibm.itso.msgtrx.forms.SampleForm

```
package com.ibm.itso.msgtrx.forms;
// You will need the following imports at least
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.*;

public class RegistrationForm extends ActionForm {
    private String name=null;
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

    public ActionErrors validate(ActionMapping mapping, HttpServletRequest
request) {
        ActionErrors errors = new ActionErrors();
        // If the name is empty then add a new error to the error list
        // Get the text for the error from the resource file
        if ((name == null) || (name.length() < 1)) errors.add("username", new
ActionError("error.username.required"));
        return errors;
    }
}
```

6. Create an action that is invoked by the application, for example, `com.ibm.itso.msgtrx.actions.SampleAction`. The action is the engine behind the function. It is the core implementation of a function.

WebSphere Studio provides wizards to create Struts elements, including form beans (**File -> New -> Other -> Web -> Struts -> Action Class**). The wizard to create Action classes allows you to define action forwards.

Example 7-5 com.ibm.itso.msgtrx.actions.SampleAction.java

```
package com.ibm.itso.msgtrx.actions;
// You will need these imports at least
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import org.apache.struts.action.*;
// Import the form-bean also
import com.ibm.itso.msgtrx.forms.SampleForm;

public class RegisterAction extends Action {
    public ActionForward perform(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response)
        throws IOException, ServletException {
        // get the form data
        SampleForm sampleForm=(SampleForm)form;
        if(form==null) {
            return mapping.findForward("failure");
        }
        // ...
        // Here comes implementation of the action
        // ...
        // Forward to the success page
        return mapping.findForward("success");
    }
}
```

7. Modify the `struts-config.xml`. Action and form beans must be registered in the `struts-config.xml` file and mappings for the action bean must be defined.

WebSphere Studio automatically registers the actions and form beans but you may need to update the mappings.

The action in Example 7-6 on page 152 defines the Java class for the action: `com.ibm.itso.msgtrx.actions.SampleAction`. This refers to the form bean: *sampleForm*. It defines the scope: *request*. It defines the page where the input came from: `input.jsp`. It also defines two aliases for forwarding. These aliases

can be accessed from the action class to forward the request to a view component.

Example 7-6 struts-config.xml

```
...
    <form-beans>
...
        <form-bean name="sampleForm"
type="com.ibm.itso.msgtrx.forms.SampleForm"/>
...
    </form-beans>
    <action-mappings>
        <action path="/sample"
            type="com.ibm.itso.msgtrx.actions.SampleAction"
            name="sampleForm"
            scope="request"
            input="input.jsp">
<!-- define aliases for URLs for different situations -->
            <forward name="success" path="results.jsp" />
            <forward name="failure" path="error.jsp" />
        </action>
    </action-mappings>
...
</struts-config>
...
```

8. Modify the resource bundle file. The resource variables you use in the application have to be defined in the applicationresource.properties file in the com.ibm.itso.msgtrx folder in the Web module.

Check the input.jsp. For example, the `<bean:message key="prompt.name"/>` line of code reads the prompt.name variable from the resource file.

Example 7-7 applicationresources.properties

```
...
prompt.name=provide your name
error.username.required=You have to provide your user name
...
```

7.2 Developing a message-driven bean with WebSphere Studio

This section explains how to develop message-driven beans using WebSphere Studio Application Developer. First, we introduce some basic concepts of the

MDB implementation and then walk through the development process using WebSphere Studio.

7.2.1 Message-driven bean implementation

Message-driven beans implement two interfaces:

- ▶ `javax.ejb.MessageDrivenBean` interfaces, which extends the `javax.ejb.EnterpriseBean` interface.
- ▶ `javax.jms.MessageListener` interface

From these two interfaces, message-driven beans implement the following methods:

- ▶ `getMessageDrivenContext()`
This returns the MDB Context object.
- ▶ `setMessageDrivenContext(javax.ejb.MessageDrivenContext)`
This assigns the MDB context to the MDB instance during the creation of the bean.
- ▶ `ejbCreate()`
This is invoked during the creation of the EJB and can be extended to perform initialization processes.
- ▶ `onMessage(javax.jms.Message)`
This method contains the bean business logic. The incoming message is passed to the method as a `javax.jms.Message` that can later be cast to any specific kind of JMS message.
- ▶ `ejbRemove()`
This method is invoked during the MDB remove process.

Example 7-8 Message-driven bean base code

```
public class SampleMDBBean
    implements javax.ejb.MessageDrivenBean, javax.jms.MessageListener {
    private javax.ejb.MessageDrivenContext fMessageDrivenCtx;
    /**
     * getMessageDrivenContext
     */
    public javax.ejb.MessageDrivenContext getMessageDrivenContext() {
        return fMessageDrivenCtx;
    }
    /**
     * setMessageDrivenContext
     */
}
```

```

    public void setMessageDrivenContext(javax.ejb.MessageDrivenContext ctx) {
        fMessageDrivenCtx = ctx;
    }
    /**
     * ejbCreate
     */
    public void ejbCreate() {
    }
    /**
     * onMessage
     */
    public void onMessage(javax.jms.Message msg) {
    }
    /**
     * ejbRemove
     */
    public void ejbRemove() {
    }
}

```

7.2.2 Life cycle of a message-driven bean

In order to fully understand how a message-driven bean actually works, it is useful to have a clear idea of how and when an MDB is created and invoked by the EJB container.

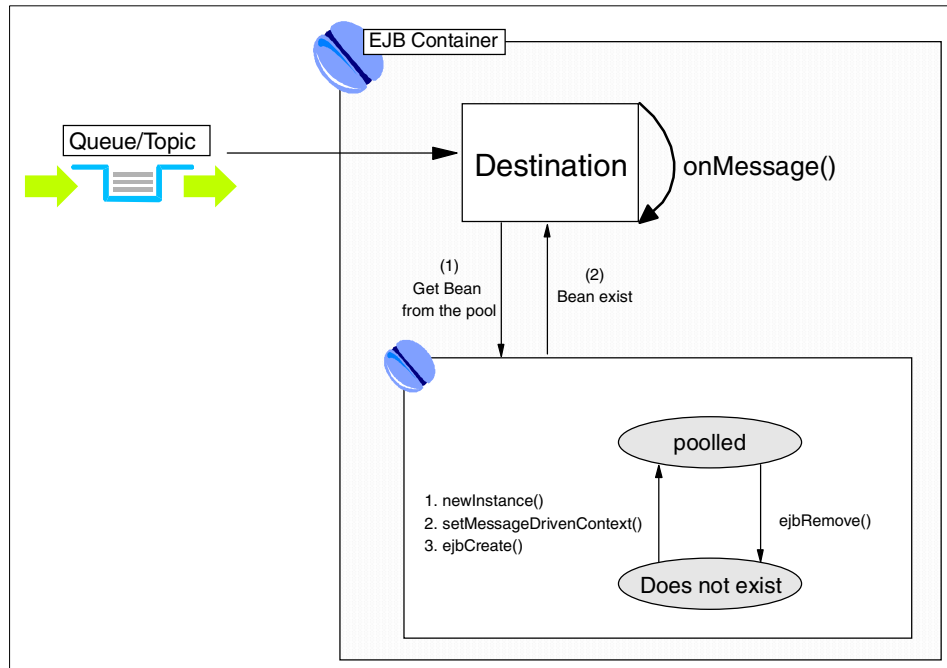


Figure 7-2 Message-driven bean life cycle

The message-driven bean life cycle is completely controlled by the EJB container. The MDB life cycle is as follows:

1. When the EJB container receives a new message from the queue, it looks for an available bean in the bean pool.
 - a. If a bean is available in the pool, the `onMessage()` method is invoked and the MDB is then returned to the pool.
 - b. If there is no bean available, a new bean is created:
 - i. A new instance of the MDB is created.
 - ii. The message-driven bean context is passed to the MDB by invoking the `setMessageDrivenContext()` method.
 - iii. The `ejbCreate()` method on the MDB is invoked.
 - iv. The `onMessage()` method is invoked with the corresponding JMS Message.
 - v. The MDB is added to the pool, to be reused later.
2. If a bean in the pool is not used for a long period of time or the pool has to be purged to free space for new beans, the MDB the `ejbRemove()` method is invoked and the bean is destroyed.

7.2.3 Creating an MDB using WebSphere Studio

As with any other EJB, an EJB project must be created before we can begin developing the MDB.

1. To create an MDB, open the deployment descriptor of the EJB project and go to the Beans tab, then click **Add...** .
2. On the Enterprise Bean creation window, let us select the type of EJB, a message-driven bean, and supply a name and the package of the new EJB, as shown in the following figure.

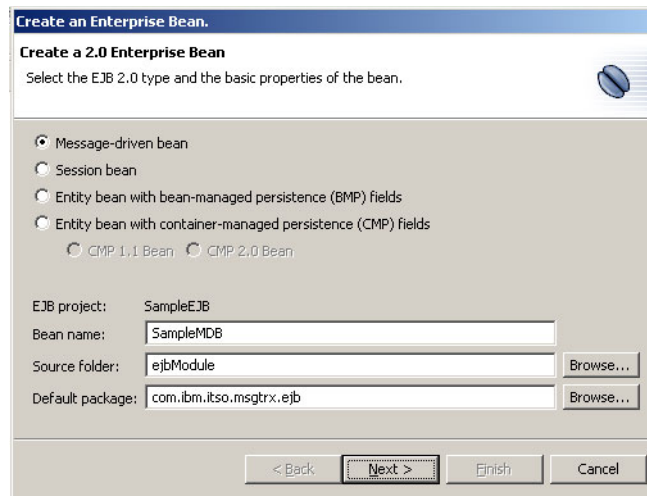


Figure 7-3 Create an Enterprise Bean (1) window

3. Once the EJB Type has been selected and the bean name and package have been provided, click **Next..** Continue on the following window and supply the message-driven bean specific information:
 - **Transaction type:** MDBs can use container- or bean-managed transactions.
 - **Message-Driven Bean destination:** MDBs can consume messages arriving from a JMS queue or a topic. If the MDB is consuming publications from a topic, subscription durability can be set, whether it is a durable or non-durable subscription.
 - **MDB supertype:** one MDB can be the supertype of another MDB, meaning that the MDB will extend the supertype class.
 - **Bean class:** the MDB fully qualified class name.
 - **Message Selector:** message-driven beans can use JMS Selector to filter the message they get from the destination queue.

- **Listener Port name:** this is a WebSphere binding requirement for the MDB. In WebSphere, the MDBs are connected to a Listener Port, where it is actually listening for messages at a given queue.

The following figure shows the window for this operation.

Figure 7-4 Create an Enterprise Bean (2) window

4. In the next window, we can define the superclass for the MDB, unless we are using a Bean supertype, in which case the superclass will be the supertype MDB class.
5. Click **Finish** and the MDB will be created and registered in the deployment descriptor for the project.

This brings us back to the deployment descriptor Beans tab, where we will be able to modify the MDB settings once it has been created. Additional settings are also available in this window.

Once the bean creation is finished, go back to the J2EE Hierarchy view and expand the MDB as shown in the following figure.

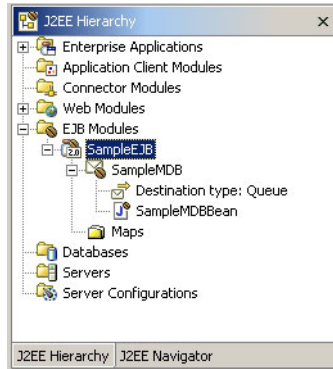


Figure 7-5 J2EE Hierarchy view in WebSphere Studio

The view shows the MDB, the destination type and the actual Java class for the bean.

7.2.4 Coding the message-driven bean

The following sample is based on the OrderProcessMDB message-driven bean from the sample application. It captures the message from the front-end application and checks for the message. Based on the content, the bean executes the appropriate business method, either to process an order or to pick up an order.

In order to minimize the overhead created by the lookup of the session bean in the context, we will do this in the `ejbCreate` method of the MDB, as shown in the following example.

Example 7-9 MDB `ejbCreate()` method

```
public void ejbCreate() {
    try {
        InitialContext ic = new InitialContext();
        orderSessionLocalHome = (OrderSessionLocalHome)
ic.lookup(Constants.ORDER_SESSION_BEAN);
        orderProcessMessengerLocalHome = (OrderProcessMessengerLocalHome)
ic.lookup(Constants.ORDERPROCESSMESSENGER_SESSION_BEAN);
    } catch (javax.naming.NamingException e) {
        LogHelper.getInstance().error("Error creating mdb initial context for
ejb lookup.");
        e.printStackTrace(System.out);
    }
}
```

The `onMessage` method performs the actual Session Bean creation and finally calls the business method, as shown in the following example.

Example 7-10 onMessage method

```
public void onMessage(javax.jms.Message msg) {
    String command = null;
    MapMessage mmsg = null;
    if (msg instanceof MapMessage) {
        mmsg = (MapMessage) msg;
        try {
            command = mmsg.getString("command");
            OrderSessionLocal orderSessionLocal = (OrderSessionLocal)
orderSessionLocalHome.create();
            OrderProcessMessengerLocal orderProcessMessengerLocal =
(OrderProcessMessengerLocal) orderProcessMessengerLocalHome.create();
            if (command.equals("processOrder")) {
                String
reply=orderSessionLocal.processOrder(mmsg.getString("request"));
                orderProcessMessengerLocal.sendOrderProcessReply(command, reply,
mmsg.getJMSMessageID());
            } else if (command.equals("pickupOrder")) {
                String reply=orderSessionLocal.pickupOrder(
mmsg.getString("request"));
                orderProcessMessengerLocal.sendOrderProcessReply(command, reply,
mmsg.getJMSMessageID());
            } else {
                LogHelper.getInstance().info("Did not recognize any command from
the message.");
            }
        } catch (JMSEException je) {
            LogHelper.getInstance().info("Error getting the command from the
message.");
        } catch (CreateException ce) {
            LogHelper.getInstance().info("Error creating the OrderSession EJB.");
        } catch (GenericApplicationException gae) {
            LogHelper.getInstance().info("Application error: "+gae.getMessage());
            // this is an unexpected exception, roll back the transaction
            fMessageDrivenCtx.setRollbackOnly();
        }
    }
}
```

The previous example illustrates some general considerations when writing message-driven beans:

- ▶ Messages are received as `javax.jms.Message` instances, and usually they have to be cast to another subclass, for example, `javax.jms.MapMessage`.
This can be done by verifying whether the instance is an instance of another class and then casting it to the correct subclass, as shown in the example. In our example, we are using XML messages, so we will be using the `javax.jms.MapMessage` class throughout the application.
- ▶ Since the `onMessage` method cannot throw back exceptions, we must catch all possible exceptions that can occur during the `onMessage` method execution.
- ▶ When an unexpected or fatal error occurs during the execution of the MDB, we can roll back the MDB transaction, using the `setRollBackOnly()` method in the `Context` object of the MDB, as long as the MDB is using container-managed transactions.

Note: For information on deploying message-driven beans in WebSphere V5, see 12.4, “Deploying message-driven beans in WebSphere V5.0” on page 253.

7.3 XML and XSLT development

XML is becoming the de facto standard for data representation in the J2EE environment. Along with the XML standard, there is a set of related standards that empowers this technology. In this chapter, we discuss some of these technologies and provide information about how they can be used in a J2EE application.

7.3.1 XML as data transfer technology

In a message-based multi-tier application, XML can play an important role in defining the communication contract between the different layers. The application architect can design a set of messages to transport information between the layers. These messages can be defined using any XML definition standard, such as DTDs or schemas. Once the XML documents have been defined, the application layers can be divided, so different groups can work in parallel in the development.

For example, in our sample scenario, the application was divided into four major layers:

- ▶ The front-end application layer
- ▶ The back-end application layer
- ▶ The integration layer
- ▶ The supplier.

The communication among these layers is done using XML messaging, and the message formats defined using DTDs.

7.3.2 Guidelines for creating an XML message

Before creating your own message structure from scratch, consider the XML schemas and DTDs that already exist for many business communities. More such domain-specific language catalogs are currently being defined by various industry consortiums and standards bodies. Standard schemas make it easier for your application to communicate with other services or applications. A good place to start looking is:

<http://www.xml.org>.

If you expect your XML-based messaging system to evolve into a large system between several applications, it might be worth considering a standardized message model. Standardizing the message model gives you the flexibility, for example, to implement new XML messages for new application versions. A generic model can also protect the message system from too many transformations, which takes a good deal of time. A generic message model will end in a generic data model of all the data that is used within a company.

A few XML guidelines are:

- ▶ Use descriptive names in XML documents.
- ▶ Group elements in related sets and decrease the number of choices that can be made.
- ▶ Create a collection element for the context of repeated elements.
- ▶ Add elements that can be used in a later stage.
- ▶ Separate metadata from the real content. For example, use head and body elements.
- ▶ Include a message version number in the message metadata.
- ▶ With large amounts of metadata, put this data in a different document.

7.3.3 Performing XML transformations

In our sample application, we used XML transformation for two purposes:

- ▶ To generate an HTML page from data stored in XML format. The `order.jsp` uses a helper class (`FCatalogHelper`) to generate the HTML output. The helper class uses the `TransletHelper` class to do the XML transformation. The `TransletHelper` takes two arguments, the URL for the XML file to transform, and the transformer object, `translet`, that is compiled from the XSL file. With this technique, the data stored in XML can be directly displayed on an HTML page. Using translets instead of programming an XSLT transformation increases performance.
- ▶ To generate a message from data stored in XML format. The order pre-processing also uses XML transformation to generate the message sent to the integration server. To generate the message, the `OrderProcess` session bean uses a `TransletHelper` class to perform the XML transformation. The XML document is provided from the front-end application as a message; the stylesheet is compiled into a translet called `ProcessOrder`.

XSLT can be used to perform XML transformations to other XML formats, HTML, or text. The advantage of using XSLT instead of programming the XML transformation using the DOM or SAX API is that the input or output message formats can be changed without affecting the application logic. The transformation logic is completely decoupled from the business logic, making it easier to maintain it later.

There are many XSLT parsers available. WebSphere comes with the Apache Xalan parser.

The Xalan parser provides all the functionalities recommended in the W3C XSL Transformation (XSLT) and XML Path language (XPath) specifications. Additionally, it provides an implementation of the Transformation API for XML interfaces (TRaX) part of the Java API for XML Processing 1.1 Public Review 2. TRaX provides a modular framework and a standard API for performing XML transformations, and it utilizes system properties to determine which transformer and which XML parser to use. It also provides an XSLTC compiler and runtime processor.

Runtime transformation using Xalan

Transformation can be performed using the TRaX interfaces or the XSLTC translet objects. The TRaX interfaces provide a standard interface to XSL implementations like Xalan.

To use the TRaX interface, follow the steps below:

1. Create a TransformerFactory, which may directly provide a transformer, or which can provide templates from a variety of sources.

The Templates object is a processed or compiled representation of the transformation instructions, and provides a transformer.

2. Use the transformer to process the source according to the instructions found in the templates, and produce a result.

Examples showing how this interface can be used to transform a given input document can be found at the Apache Web site:

<http://xml.apache.org/xalan-j>

7.3.4 Working with XSLTC

XSLTC compiles the style sheets at development time, so the transformation process will not have to compile the style sheet at runtime. These compiled style sheets are usually referred to as *translets*.

WebSphere Studio provides an XSLTC Compiler wizard that uses the Xalan XSLTC compiler to produce a .jar file with the translet.

To compile a style sheet, select **File -> New -> Other -> XML -> Compile XSL**.

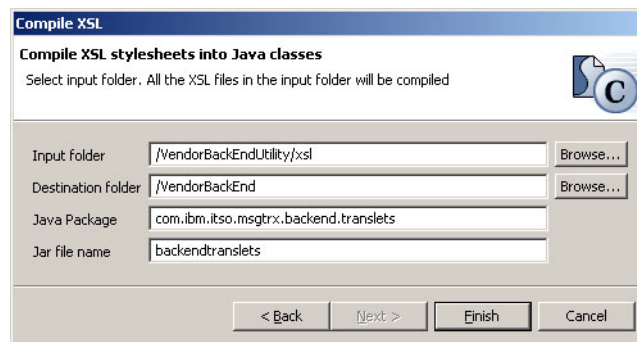


Figure 7-6 Compile XSL window

In this window, we must specify the following:

- **Input folder:** the folder where the XSL files are. All the XSL files in this directory will be compiled and saved to the destination folder.
- **Destination folder:** the folder where the resulting Java classes or .jar files will be saved.

- ▶ **Java Package:** optionally, the package to which the translet classes will belong.
- ▶ **Jar File name:** optionally, the .jar file name where all the translets are saved. If not specified, the classes will be saved as individual class files.

After completing these fields, click **Finish**. Once the translets are compiled, they can be accessed as any normal Java class.

Translets extend the `org.apache.xalan.xsltc.runtime.AbstractTranslet` class. The most used `AbstractTranslet` class methods are:

- ▶ `addParameter(String , Object)`: this method adds parameters to the style sheet during execution.
- ▶ `transform(Dom, TransletOutputHandler)`: this executes the compiled style sheet and returns the output of the transformation in the `TransletOutputHandler`.

To simplify the use of the translets, we developed a `TransletHelper` class that is reusable for any application. The helper class has the following interface:

- ▶ `setTranslet()`
This method specifies the translet to be used in the transformation process. The translet can be passed as a class name or a translet instance.
- ▶ `getTransformedDoc()`
This method returns the resulting string of the transformation process.
- ▶ `setDocument()`
This assigns the input document for the translet. The document can be an `InputStream` or a `String` representation of the XML.
- ▶ `setURI()`
This assigns the URI where the document file can be found if the `setDocument()` method is not used.
- ▶ `setParameter()`
This assigns parameter values that will be passed to the translet when the transformation is executed. This parameter overwrites the default value of the parameter defined in the style sheet using the `<xsl:param/>` tag.
- ▶ `getDocumentInputStream()`
This returns the `InputStream` as it was assigned using the `setDocument()` method.

- ▶ `getDocumentString()`
This returns the document string as it was assigned using the `setDocument()` method.
- ▶ `getURI()`
This returns the document URI as it was assigned using the `setURI()` method.

7.3.5 WebSphere Studio XML support

WebSphere Studio provides multiple tools to work with XML and XSLT. In this section, we introduce these tools and explore how they can be used in a Self-Service application.

The following XML editor tools are available in WebSphere Studio:

- ▶ The *XML editor* is a tool for creating and viewing XML files. You can use it to create new XML files, either from scratch, from existing DTDs, or from existing XML schemas. You can also use it to edit XML files, associate them with DTDs or schemas, and validate them.
- ▶ The *DTD editor* is a tool for creating and viewing DTDs. Using the DTD editor, you can create DTDs, generate XML schema files, and generate Java beans for creating XML instances of an XML schema. You can also use the DTD editor to generate a default HTML form based on the DTDs you created.
- ▶ The *XML schema editor* is a tool for creating, viewing, and validating XML schemas. You can use the XML schema editor to perform tasks such as creating XML schema components, importing and viewing XML schemas, generating DTDs and relational table definitions from XML schemas, and generating Java beans for creating XML instances of an XML schema.
- ▶ The *XSL editor* can be used to create new XSL files or to edit existing ones. You can use content assist and various wizards to help you create or edit the XSL file. Once you have finished editing your file, you can also validate it. Also, you can associate an XML instance files with the XSL source file you are editing and use that for guided editing when defining constructions, such as an XPath expression.

Wizards are also available to simplify some of the common operations; for example:

- ▶ The *XPath expression wizard* to create XPath expressions, which are used to search through XML documents, extracting information from the nodes (such as an element or an attribute).
- ▶ The *XSL debugging and transformation tool* to apply XSL files to XML files, transforming them into new XML, HTML, or text files. After the transformation has taken place, the XSL Debug perspective opens, allowing you to visually

step through an XSL transformation script, highlighting the transformation rules. The views in the XSL Debug perspective can help you debug the XML or XSL files.

- ▶ The *XML and SQL query wizard* to create an XML file from the results of an SQL query or take an XML file and store it in a relational table. When creating an XML file from an SQL query, you can optionally choose to create an XML schema or DTD file that describes the structure that the XML file has for use in other applications. Two Java class libraries, SQLToXML and XMLToSQL, are included so you can use them in your applications at runtime.
- ▶ The *SQL wizard* or *SQL query builder* to create the SQL queries from which your XML files are generated.

Additionally, WebSphere Studio provides mapping tools, such as:

- ▶ The *XML to XML mapping editor*, used to map one or more XML source files to a single target XML file. You can add XPath expressions, groupings, Java methods, or conversion functions to your mapping. Mappings can also be edited, deleted, or persisted for later use. After defining the mappings, you can generate an XSLT script. The generated script can then be used to combine and transform any XML files that conform to the source DTDs.
- ▶ The *RDB to XML mapping editor*, which is a tool for defining the mapping between one or more relational tables and an XML file. After you have created the mapping, you can generate a document access definition (DAD) script, which can be run by the DB2R XML Extender to either compose XML files from existing DB2 data or decompose XML files into DB2 data.

WebSphere Studio also provides a wizard for creating XML JavaBeans.

7.3.6 Using XML JavaBeans

Once you have designed and implemented an XML message structure, you can create Java classes from this structure. These classes can be used for creating and validating the messages in a Java application. There are several products on the market for generating Java classes from the DTD or XML schema. It is preferable to use a generating tool that is based on a standard DOM or SAX implementation.

WebSphere Studio provides a wizard to create JavaBeans that will represent an XML document based on a DTD or schema. The wizard can be initiated from the context menu of the DTD or schema file by selecting **Generate -> JavaBeans....** This brings up the JavaBean wizard window shown in Figure 7-7 on page 167.

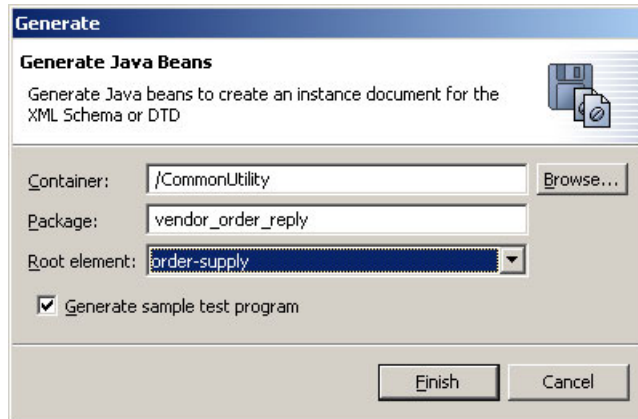


Figure 7-7 Generating XML JavaBeans

In this window, specify the following parameters:

- ▶ **Container:** the name of the WebSphere Studio container where the generated JavaBeans will be stored.
- ▶ **Package:** the package name for the generated JavaBeans.
- ▶ **Root element:** the root element of the document represented by the Schema or DTD.
- ▶ **Generate sample test program:** select this if you want to have a sample application using the generated JavaBeans.



Developing WebSphere MQ Integrator message flows

.This section discusses the development cycles and considerations for WebSphere MQ Integrator.

The development of message flows is often considered to be a very complex task, because it takes a set of quite different skills to get started:

- ▶ Information design issues - designing message formats and database tables, or learning about existing formats or data structures.
- ▶ System administration tasks - configuring users, setting up WebSphere MQ objects and database connections.
- ▶ Programming ESQL, the programming language used in WebSphere MQ Integrator.

The following is simply a quick overview to get you started with WebSphere MQ Integrator in a simple broker domain.

8.1 What is a broker domain?

A broker domain is a kind of *habitat* for the five WebSphere MQ Integrator components:

- ▶ Control Center
- ▶ Broker
- ▶ Configuration Manager
- ▶ User name server
- ▶ NNSY rules engine

For the purposes of this book, we will concentrate on the following components:

- ▶ Broker - the component that does the productive work, available for Win32, some UNIX systems, and z/OS. Each broker needs a dedicated WebSphere MQ queue manager installation to live on.
- ▶ Configuration Manager - used for development and administrative purposes only. It is currently only available for Win32 environments since the Configuration Manager needs a WebSphere MQ queue manager of its own.
- ▶ Control center - implements the graphical user interface to manipulate WebSphere MQ Integrator objects. Control center is available for Win32 environments only. It will access the Configuration Manager through a WebSphere MQ client connection using the TCP/IP protocol.

These three components are essential for any WebSphere MQ Integrator runtime environment. They can coexist on a single physical machine to be used as the development box.

In practical use, there may be broker domains with hundreds of brokers and Control Center installations. While these figures may grow, there is always one single Configuration Manager per broker domain.

Note: The word domain is related to the term *dominion*, which is a good mnemonic device since the Configuration Manager governs the brokers.

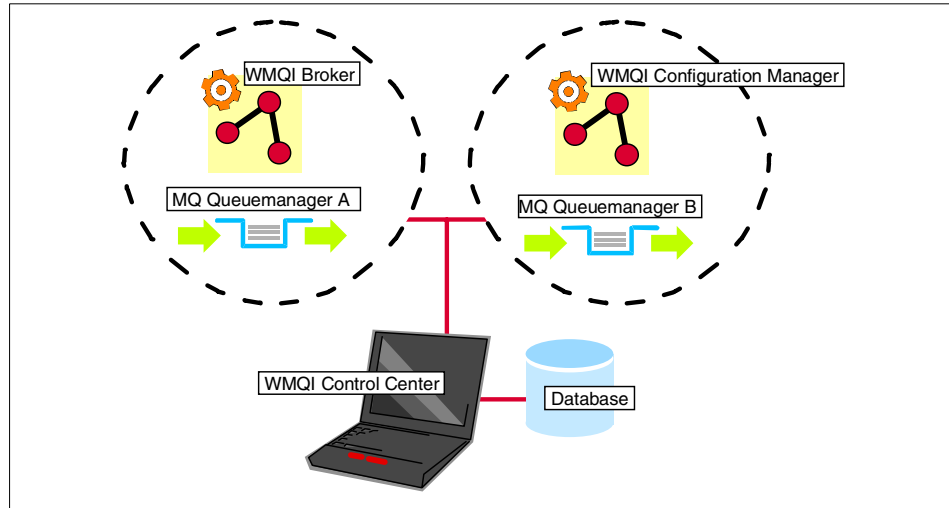


Figure 8-1 A simple WebSphere MQ Integrator broker domain fits on a single physical machine

Note: Choose a fast machine with at least 512 MB RAM for the development scenario. Configurations with less memory are not recommended.

8.2 Developing message flows

As reflected in the graphical appearance of the Control Center, WebSphere MQ Integrator (WMQI) provides three main types of applications which can be used in combination.

- ▶ Message format translation with the Message Repository Manager
- ▶ Advanced publish/subscribe functionality, including user management
- ▶ Component integration and point to point message routing

The capabilities of a message repository manager and subscription management are discussed here only in general. In this section, we concentrate on the design of message flows.

Note: Please be careful not to confuse the terms *message flow* and *workflow*. A message flow as defined in WebSphere MQ Integrator is on a more technical layer than the business-related workflow. Indeed, message flows can be used to implement workflows.

8.2.1 Preparations: creating queue managers and defining queues

In order to set up WebSphere MQ Integrator components, basic WebSphere MQ facilities are required. We need to install queue managers and define certain MQ objects that WebSphere MQ Integrator relies on, as well as the queues and channels we need for the sample application. We assume the WebSphere MQ software is already installed on the system.

On systems running Windows, you can use either the graphical user interface or the MQ Explorer to create new queue managers. For other operating systems, similar tools exist. On AIX, there is a plug-in for the system management console called System Management Interface Tool (SMIT), available as a SupportPac. The process is mostly self-explanatory. You can keep the default values.

- ▶ Do not choose the queue manager as the default queue manager of the system. We do not recommend choosing a default queue manager at all, since this is a common source of errors.
- ▶ Identify the dead letter queue you wish to use (and keep in mind that this queue also has to be defined).
- ▶ Select **Circular logging**, which is always a good choice for development environments.
- ▶ Choose **Start Listeners**, so you do not have to worry about starting the listeners every time the server starts.
- ▶ Select **Automatic start**, a convenient way to start up the queue manager.

Note: Save the queue manager's configurations whenever they are changed; use SupportPac MS03.

8.2.2 Using the Control Center

WebSphere MQ Integrator message flows are built using the Control Center GUI interface. The Control Center only runs on Windows but can be used to deploy and monitor applications to brokers on any MQSI-supported platform.

To begin building an application, start the Control Center by selecting **Start -> Programs -> IBM WebSphere MQ Integrator 2.1 -> Control Center**. A Configuration Manager Connection window should appear as shown in Figure 8-2 on page 173, prompting you for details required to connect to the Configuration Manager.

Note: Before starting the Control Center, you will have to include the user ID under which you are running the Control Center into the local user management of the user the Configuration Manager runs on. Be sure to add the user ID to the WebSphere MQ Integrator groups that match the roles you will be using.

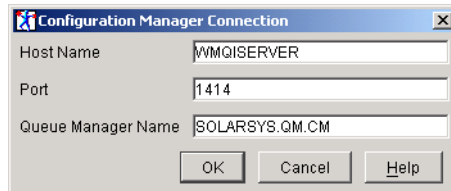


Figure 8-2 Configuration Manager Connection window

These fields tell the Control Center how to connect to the appropriate Configuration Manager. These values were determined when the Configuration Manager was created.

If this window does not appear, you may get an error window that reports a problem connecting to the Configuration Manager. This can be the case if the Control Center was used to work with a different Configuration Manager instance that is no longer active, or if this is the first time you have opened the Configuration Manager. You may see no window appear at all. This indicates the connection to the Configuration Manager was successful. If the connection window has not appeared, you can go to **File -> Choose Connection**, and the required window will appear.

Click **OK** and the Control Center will open.

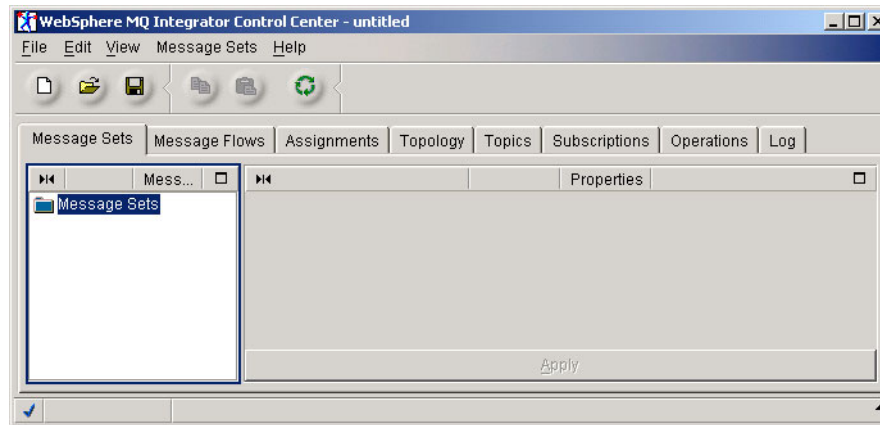


Figure 8-3 Starting the Control Center

8.2.3 Creating message flows

The general process to begin creating message flows is as follows.

1. Click the **Message Flows** tab.
2. Right-click **Message Flows** and select **Create -> Message Flow Category**. Enter a new name for the category. Creating categories is not necessary but provides a way of organizing related message flows. It simply makes it easier to navigate your way through a large number of message flows while working in the Control Center.
3. Highlight the new category, right-click, and select **Create -> Message Flow**.
4. Create new nodes in the message flow by copying existing nodes from the right pane to the left. The properties of the new node can be accessed and modified by double-clicking the new node.
5. Check in your changes. The next time, you will need to check out the node for updates.

Sample application message flows

The sample application consists of three WebSphere MQ Integrator message flows to demonstrate how a one-to-many scenario can be implemented using WebSphere MQ Integrator. We discuss the three flows generally to illustrate the design principles.

1. **Order Decomposition Flow:** a composite order coming from another system is acknowledged and then decomposed. The parts are routed separately to the different suppliers.

2. Suppliers' response storage flow: response messages from the suppliers are gathered in a relational database for persistent storage.
3. Response Composition flow: upon request from the third system, the responses are aggregated into a single message and returned to this system.

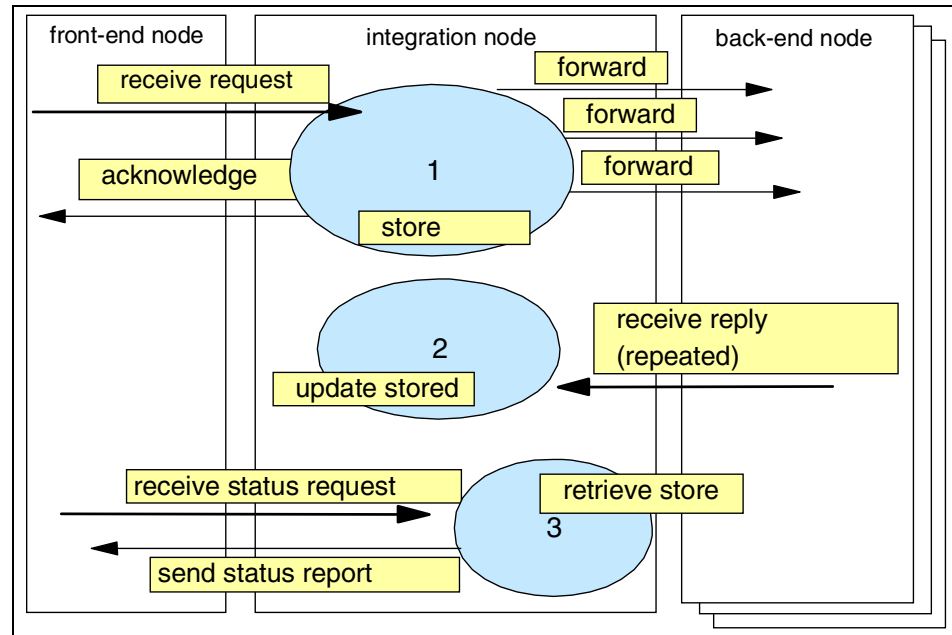


Figure 8-4 Overview on the flows (generic)

The first flow, shown in Figure 8-5 on page 176, is the order submit process. The customer places an order using the front-end application. The back-end receives the order, prepares it, then sends it to the integration node. The diagram illustrates the flow from the moment when the integration node gets the complex message with the order. There are two actions that the integration node performs:

- ▶ Sends an acknowledgment message back to the back-end system.
- ▶ Decomposes the complex order message, which consists of three (or fewer) elements, the three color components. The order is stored in a database. Then, each component of the decomposed message is stored as a separate record in the database. Also, the system creates sub-orders from the order components and sends them out to the suppliers.

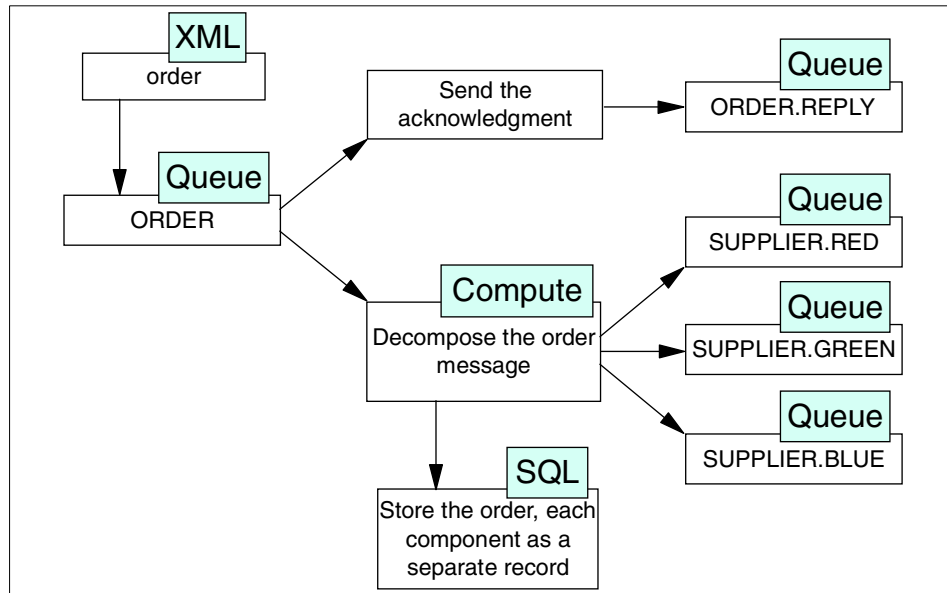


Figure 8-5 Sending order to the system

The next flow, illustrated in Figure 8-6, takes place after the supplier gets the sub-order message and sends a reply back to the integration node about the committed amount of supply. When the integration node receives the message with the sub-order reply, it looks up the message based on the order ID. If the message is found, it stores the details of the reply in the same record where the original sub-order details were stored.

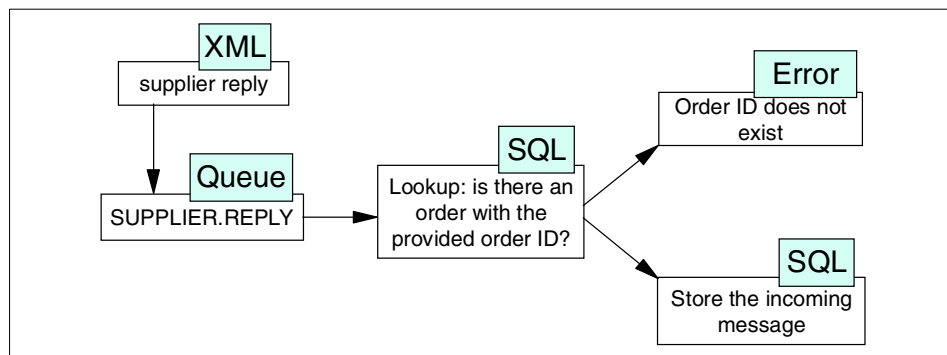


Figure 8-6 Storing the reply from the suppliers

The last flow, illustrated in Figure 8-7 on page 177, is a separate process, where the customer can pick up the order and find out if it can be fulfilled or not. The message comes through the front end to the back end, then hits the integration

node. The integration node looks up the order items' sub-components from the database and composes a complex message that is sent back to the back-end system. The back end forwards the message to the front end, which creates the response window.

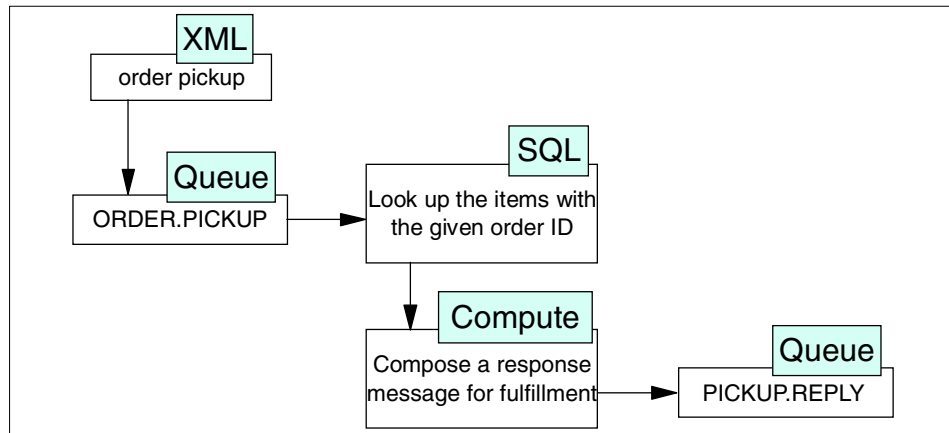


Figure 8-7 Order pickup

Input node

Simply set the transaction flag in your database (or compute node) on your data source to Yes. By default, it is set to Automatic, which means supporting transactions if the input message was persistent. The only difference with XA control of WebSphere MQ is that the broker issues two separate commit calls, one for MQ resources and one for the database connection. Thus, it is a one-phase commit.

Decomposition of messages

There are two ways to decompose a message within WebSphere MQ Integrator. Multiplication of a message into different new messages can happen in two ways:

- Cloning messages in the flow

A straightforward way to clone messages is simply by connecting several destinations to a single output terminal of a node, as shown in Figure 8-8 on page 178.

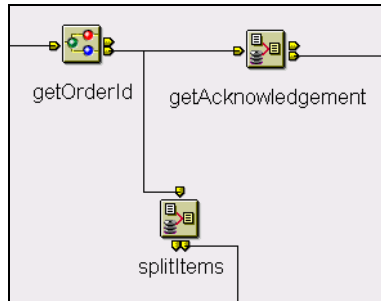


Figure 8-8 Cloning messages in the flow

The advantage of this approach is the graphical visibility in the flow diagram. The messages that have been generated are equal. This will often not be suitable for decomposition purposes. If you want to decompose the message into parts, you will have to remove different parts of it in the branches. This solution can only be used where the message parts are known. It is not suitable for a dynamic decomposition because of the undeterminable length of the elements.

► Decomposition using the ESQL propagate statement

The propagate statement allows us to generate new messages from within ESQL. While the ESQL program of a compute node executes, it can propagate messages to the out terminal of that node.

Thus, lists of variable length can be decomposed with ESQL using the propagate statement in a loop.

Example 8-1 Splitter node ESQL

```
-- declare a counter variable for the loop
DECLARE i INTEGER;
-- initialize the counter
SET i =1;
-- loop above the items to split
WHILE i <=CARDINALITY(InputRoot.XML.order."order_item"[]) DO
  -- to copy all headers into the new message
  SET OutputRoot = InputRoot;
  -- remove the payload
  SET OutputRoot.XML =NULL;
  -- copy only one part
  SET OutputRoot.XML.order.order_item= InputRoot.XML.order."order_item"[i];
  -- add identifier
  SET OutputRoot.XML.order.orderid=InputRoot.XML.order.order_id;
  --Trial and Error
  SET OutputRoot.MQMD.MsgId=OVERLAY(InputRoot.MQMD.MsgId PLACING uuidasblob
FROM 9);
```

```

-- forwards actual state of message to the output terminal
PROPAGATE;
-- increase counter
SET i = i+1;
END WHILE;
-- passing it to the failure terminal, which is unconnected
RETURN FALSE;

```

Routing

There are two approaches to routing:

- Output to destination list

A powerful way to route messages is through manipulation of the `DestinationList` in the `LocalEnvironment`.

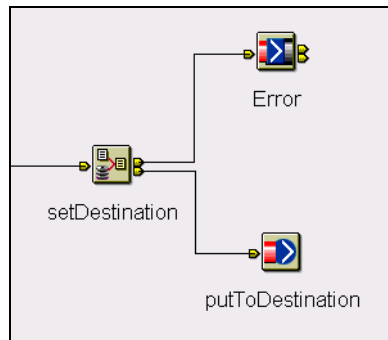


Figure 8-9 Dynamic routing by `DestinationList` values

Configure the computation of the `DestinationList` in a compute node. If you use a node only for that purpose, you can set the value of the property `compute mode` in the `Advanced` tab to `LocalEnvironment`. If there is also other computation performed in the node, use `Message` and `LocalEnvironment`. The folder has to be called `Destination`, with a subfolder named either `MQ` or `MQDestinationList` or `RouterList`, for example:

```

SET OutputLocalEnvironment.Destination.MQ.DestinationData[1].queueName =
'Q1';

```

Configure the output node, examine the list of destinations and send the message to those destinations, then set the property `Destination Mode` in the `Advanced` properties tab to the value `Destination List`.

- Visible routing through filter nodes

You can perform routing using filter nodes. These branch off to different destinations. The advantages of this technique are as follows.

- The routing is represented graphically
- Different branches may process the message in different ways

The filter node can route a message depending on a value:

- In the message body
- Resulting from a calculation
- True or false, being returned from an expression

This node accepts ESQL statements in the same way as the compute and database nodes. In the case of the filter node, these should be ended by a statement such as:

```
RETURN <boolean expression>
```

Contrary to compute nodes, the filter node addresses Root and Body elements. There is no Input/Output- Root or Body, because the node does not change the message. Filter nodes have boolean results. In the sample flow, the test is straightforwardly checking for the value of a given element. For example:

```
(Body.A = 'red' OR Root.XML.A = 'RED') AND NOT(Body.B = 'transparent')
```

Note: NULL and UNKNOWN are treated in the same way. References to missing elements propagate the message to the unknown terminal.

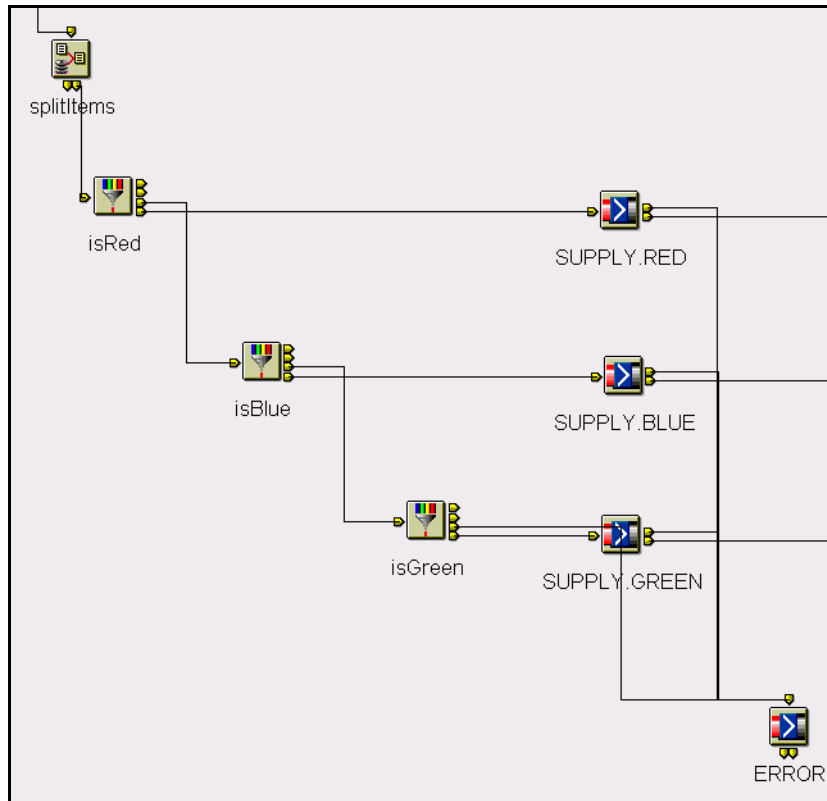


Figure 8-10 Cascading filters forming a bus

Generating IDs

If you have no need to store new message IDs, they can be generated when writing the new message to the queue. Select the MQOutput nodes'

Properties -> Advanced -> New Message ID to have new IDs generated.

If you want to store the message ID to a database or include it in another message, this option is not sufficient. In this particular case, message IDs have to be generated with ESQL in a compute node before the message is propagated to an MQOutput node.

Message IDs (MQMD.MsgId) are 24-byte binary objects (BLOB), basically made up of a constant string 'AMQ', the first eight characters of the queue manager name, and the 16-byte universally unique identifier (UUID). Only the length of 24 bytes is required; the first eight bytes could also be empty. We wanted to keep

the information about the originally sending queue manager provided in the first part and only generated a new UUID, using the following expression:

```
SET OutputRoot.MQMD.MsgId=OVERLAY(InputRoot.MQMD.MsgId PLACING uuidasblob
FROM 9);
```

When reading a message ID and storing, it is casted to a different type. Casting a BLOB UUID to CHAR results in a hexadecimal string (format X'0123456789ABCDEF'); a 24-byte BLOB will result in a character string of length 51 (48 + "X" + "").

Keeping context through request/reply cycles

It is often necessary to keep some context between message flows, for example in request/reply message flows, where both the request message and the reply message need to be processed by the broker. Typically, the broker will manipulate the ReplyToQ field of the request message so that any reply message is redirected to a broker queue for further processing. The problem is then how to restore the ReplyToQ to its original setting so that the reply message can be sent to the originator of the request. Various approaches to solving this problem exist:

- ▶ Database tables
- ▶ Queues
- ▶ Plug-in node

Storing messages

WebSphere MQ Integrator enables you to store and retrieve message contents to relational databases from database insert or update nodes as well as from compute nodes.

It is possible to mix the transaction types of nodes that operate on external databases set by the property Transaction Mode. Some nodes in a message flow might specify automatic transactionality, meaning that any work they perform is not committed until the message flow successfully completes. Others might specify commit transactionality, in which case behavior differs from system to system:

- ▶ On z/OS, commit transactionality in a single node behaves as expected. Actions taken in this node only are committed regardless of the subsequent success or failure of the message flow. Any actions taken prior to this node under automatic transactionality are not committed, but remain within a unit of work and may either be committed or rolled back, depending on the success of the message flow.
- ▶ On distributed systems, commit transactionality means that any work that has been performed in this message flow over the same ODBC database connection to date, including any actions taken in this node, is committed

regardless of the subsequent success or failure of the message flow. ODBC commits connections, not single statements.

Thus, a mix of nodes with automatic and commit transactionality in a single message flow requires two ODBC connections to the same data source: one for the nodes that are not to commit until the completion of the message flow, and one for the nodes that are to commit immediately. The nodes that commit immediately will also cause all operations carried out by preceding automatic nodes to be committed as well.

Important: Locking problems can occur when automatic and commit transactionality are mixed.

In particular, if a node with automatic transactionality carries out an operation, such as an INSERT or an UPDATE, that causes a database object (such as a table) to be locked, and a subsequent node tries to access that database object using a different ODBC connection, an infinite lock (deadlock) occurs. The second node waits for the lock acquired by the first to be released, but the first node will not commit its operations and release its lock until the message flow completes, and this will never happen because the second node is waiting for the first node's database lock to be released.

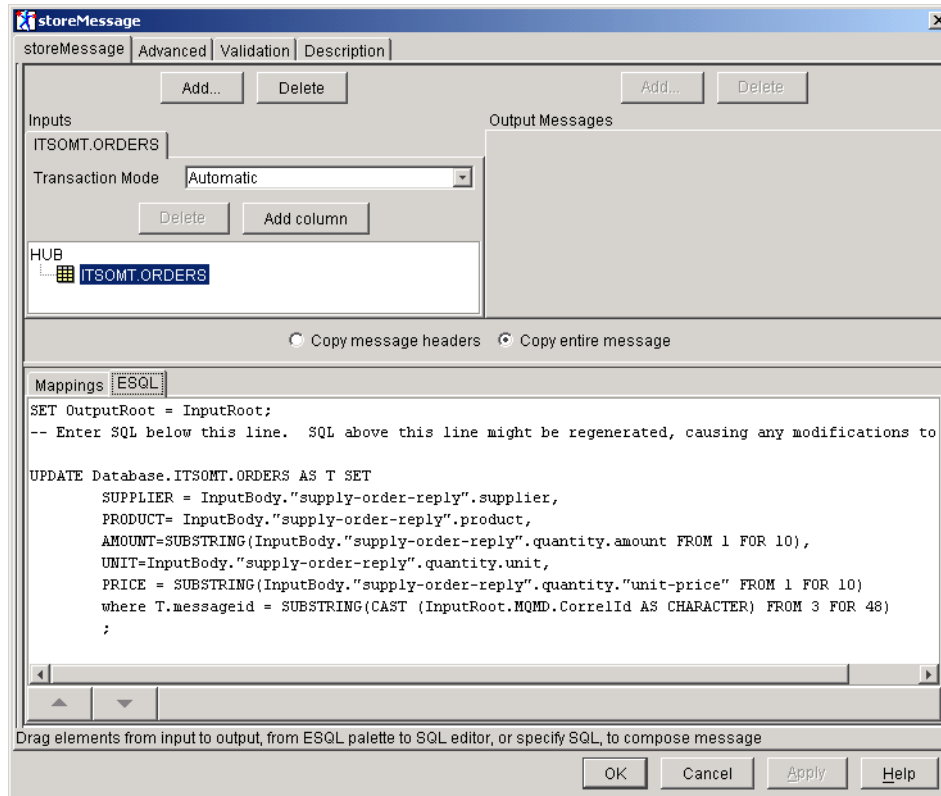


Figure 8-11 storeMessage, ESQL script

Sub-flows

Sub-flows in WebSphere MQ Integrator can help to break the complex flow into smaller modules. These modules can also be reused in other flows.

By using flows, developers can eliminate the “spaghetti code”, the long flows and codes that handle the whole process.

It also helps to develop the flows step-by-step, maybe even by multiple developers.

Flows that are built from sub-flows are faster to develop and easier to debug.

Note: For information on how this MQSI message flow was deployed, see Chapter 13, “Configuring WebSphere MQ and MQ Integrator” on page 259.



Security

As information technology increases in importance, so do the number of threats directed against this critical infrastructure. A comprehensive security strategy is essential to protect vital data and to ensure continuity of operations.

An enterprise should be able to deliver a security solution that will protect data and assets, detect threats and intrusions, and recover from incidents. It should also be able to provide services to manage the end-to-end security needs.

This chapter discusses the security considerations for end-to-end solutions in the context of the Patterns for e-business, J2EE messaging applications. The chapter also includes some of the system management considerations.

The chapter begins with a short introduction to end-to-end security, describing security principles and concepts.

Using the IV Corp sample application, a high-level security solution design is presented based on selected patterns.

At the end application security is discussed.

9.1 End-to-end security

Security is a vast subject. It encompasses a broad expanse of issues and measures that must be put in place to ensure a safe environment. We start by reviewing some of the basic principles and recommendations.

Protecting assets is a multi-faceted task in a Self-Service e-business solution. The ultimate goal in this environment is for users to access business information. The trick is to protect the assets that need protection, while making that information available to all who are authorized to access it. A big part of the challenge is determining what resources need protection, and at what stage. Optimally, access rights should be determined and denied at the earliest possible stage, rather than letting a user traverse deep into the network, then deny access to a resource at the last step, which wastes system resources and the user's time.

It is also important to prevent duplicate or unnecessary security checks, which can impact the performance of the network and applications. Each situation is highly dependent on the application and the types of resources used. In this section, we point out the resources that may need protection, the types of protection to consider, and possible solutions.

The principles of security

These are the five principles of security:

- ▶ Confidentiality - Protect data from disclosure, whether the data is being stored locally or being transmitted on a network.
- ▶ Integrity - Detect unauthorized modification of data. Modification could be inadvertent, for example through hardware or transmission errors, or deliberate, for example by attack from unauthorized persons.
- ▶ Authenticity - Verify a user's identity and control access. "Users" in this context can, for example, be program names, transaction IDs, user IDs, address space and IDs.
- ▶ Accountability - Track actions and events to unique individuals or entities.
- ▶ Non-Repudiation - Verify with virtually 100% certainty that a particular piece of data is associated with a particular individual or company, just as a handwritten signature on a bank check is tied back to the account owner.

In addition to these principles, we always should have in mind some basic rules to improve the way that we can keep our business secure.

Basic rules of security

The following are the basic rules of security:

- ▶ Security is policy enabled by technology.

Good security is what the customer's business rules dictate, so security is policy enabled by technology. Often customers have policies regarding the use of data, and other things such as door locks and badge readers on the physical plant. The existing policies might not extend to the use of data on the Internet, so a policy review and development project might be in order. Not having a good policy means that the customer might not understand the actual value of the data being used in the e-business application. To decide how much security to apply, or rather to decide how much risk needs to be reduced, the value of the data used in the application needs to be understood. The other risk element is the brand name of the company. Businesses always fear that the company name will appear on the front page of the newspaper, saying that their Web application was compromised.
- ▶ Never spend more solving a problem than it will cost to tolerate it.

The trade-off is the risk assumed for the money spent in reducing that risk, and perhaps other risks unforeseen. Management can decide these trade-offs.
- ▶ There are management solutions to technical problems, but there are no technical solutions to management problems.

If management will not make decisions on policy or on what should be done with the application, all the technology in the world cannot fix the problem.
- ▶ Nothing useful can be said about a security mechanism outside the context of an application.

To be able to discuss specific use of security technology, a specific application must be the context of the discussion. Specific security function deployments are made to secure the value of the data in a specific application. What is good protection in one application, might be too much or too little in another.
- ▶ Application programmers should never write "security code".

Application programmers should have standard security facilities available for their use, and should not be responsible for implementing the security of the application.

Five points for good security policy

These are five good points to have in mind when developing a security policy:

- ▶ Good security begins with a good policy: start your end-to-end security by establishing a good security policy directed to the business requirements.

- ▶ Know the enemy: study and understand the possible attacks that your business data and process might suffer.
- ▶ Minimalism is a virtue: expose and use the minimum assets required for the business process to exist.
- ▶ The price of security is eternal vigilance: never rest thinking that the final degree of security is reached. Processes change, people change, rules change, data change. Periodically review your policy and the technology to enable it.
- ▶ Always have an escape plan: prepare your company for an alternative plan in case you have a breach in security.

The following questions are a good starting point to build a comprehensive security policy and should be extensively refined to meet your particular needs.

Questions for business management

Questions to ask when developing a security policy:

- ▶ What is your business need for security?
- ▶ How do you plan for the future?
- ▶ What are you protecting yourself from?
- ▶ How do you know if you are successful?
- ▶ What security policies are implemented and enforced?
- ▶ What security controls are in place?
- ▶ Do they align with your business requirements? How do you know?
- ▶ What processes are in place to support the policies?
- ▶ How do you enforce consistency across the organization?

9.2 Applying security to our Runtime patterns

In this section we will extend the product mapping for our Runtime pattern to include security products.

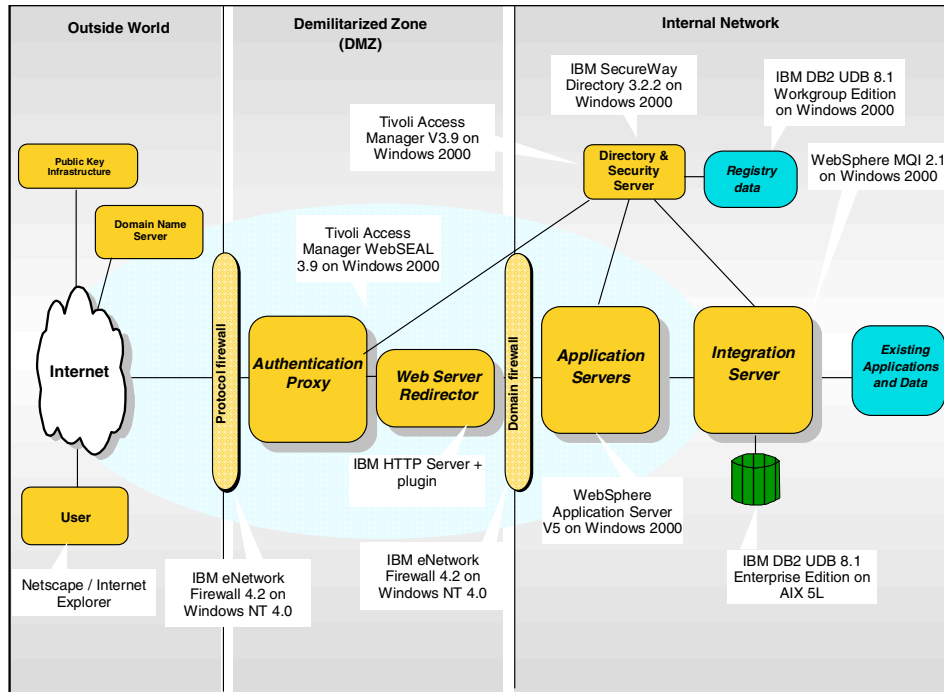


Figure 9-1 Product mappings for the improved security for the IV Corp application scenario

In this pattern, there are multiple application servers and an integration server. This implies that we need to have an external security server that will serve as a security proxy, which intercepts a request in order to map/transform the user credentials into the appropriate credential format acceptable by the other servers. To support this part of our design, we can use a Runtime pattern diagram for heterogeneous servers with external authentication and authorization servers. For more complex Single Sign-On solutions and detailed information, refer to the IBM Redbook *Access Integration Pattern Using IBM WebSphere Portal Server*, SG24-6267 at:

<http://www.redbooks.ibm.com>

The technique described in the previous chapter is called Single Sign-On, with which users can move between applications without re-authenticating every time they access a new application. Single Sign-On has to be configured for the domain where this service is used. In our sample application, we do not have multiple applications, but let's assume we have a separate Web application for order fulfilment and payment processing. Single Sign-On would allow us to log in to the domain using one application, for example order pickup, then access another application, for example payment processing, without re-authentication.

The following products support Single Sign-On for Web applications in our architecture:

- ▶ Tivoli Access Manager's WebSEAL

WebSEAL is a security reverse proxy that is used to authenticate the user, create and maintain a session with the user, and provide URL-level authorization. It also hides the internal structure of Web resources through URL mapping. WebSEAL supports multiple types of authentication and implements stepping up to a stronger authentication type if necessary.

- ▶ IBM HTTP Server

The Tivoli Access Manager can protect any static content on the Web server including the server itself, so that non-authenticated users will not be able to communicate with the Web server behind the security reverse proxy, WebSEAL.

- ▶ IBM Directory Server

Access Manager supports numerous LDAP directories. The IBM Directory Server is shipped with the Tivoli Access Manager; it stores user information and user privileges, in addition to other application information.

- ▶ Tivoli Access Manager

Tivoli Access Manager consists of the following runtime components:

- Management Server

The Management Server manages the Access Manager security policy. The Management Server receives updates from the console, Administration API, or Administration command line interface.

- Authorization Server

The Authorization Servers are used by applications in remote mode. Remote mode means that the application sends a request to the server to answer the question “Can the user perform the action on the resources?”. Local mode means that the application has an in-memory cache of the policy so the application can check this for a decision without sending a message outside the application; for example, WebSEAL works in local mode.

For more information on the Tivoli Access Manager product and integration with WebSphere Application server, please refer to:

<http://www.tivoli.com/products/index/access-mgr-e-bus/>.

The following steps provide a simple technical walkthrough for user authentication in this example.

1. The customer (application end user) uses a browser to locate the Web application from the Web.
2. The request hits the protocol firewall, which only allows appropriate traffic. The HTTP request is passed to the security reverse proxy. An extension to this can be to implement a network dispatcher that selects the most available Web reverse proxy at a time.
3. The security reverse proxy is responsible for authentication and for session establishment and maintenance. The proxy authenticates the user if it is required for the resource, then establishes the session. Authentication is checked against the LDAP user registry.
4. Once a session is established, the security reverse proxy will authorize the user based on the URL the user is trying to access. This authorization is coarse grained, as it can only affect the URL requested.
5. If the request is authorized, then it is forwarded to the Web server (in this case, a Web server redirector). The reverse proxy may perform load balancing across the Web servers. An extension can be introduced here for managing the load between Web servers by introducing a load balancer between the security reverse proxy and the Web servers.
6. The request is then sent through the second firewall to the application servers. The Web application servers execute business logic and call on the authorization service for finer-grained control. This authorization service is accessible via an API or through standard J2EE security. If the request is authorized, then a function is executed on behalf of the authenticated user. If the function communicates with back-end systems through an integration server, then it is up to the design of the integration layer to call authorization service for further authorization.

For additional information on security products, please refer to:

<http://www-3.ibm.com/security/index.shtml>,
<http://www.tivoli.com/products/solutions/security/news.html>

9.3 Security guidelines

The following sections point out some common security guidelines that should be taken into consideration while designing an e-business solution.

Securing connections in a solution

On the architecture level (as opposed to the application level), connections between nodes should be secured. The purpose of securing the communication is to prevent non-authorized persons and systems from listening to the communication or to participate in the interaction.

Figure 9-2 illustrates the commonly used and highly recommended secure communication lines between nodes.

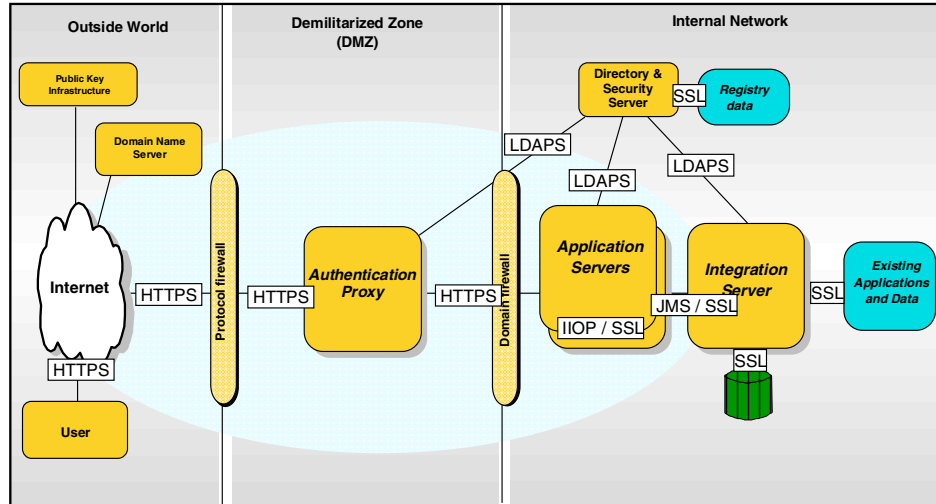


Figure 9-2 Secure connection between nodes

The secure communications are as follows:

- ▶ HTTPS is the secure HTTP connection using SSL. Nodes that are communicating via TCP/IP using the HTTP protocol should use secure SSL communication. The level of security depends on the options set for the connection.
- ▶ LDAPS is the secure LDAP connection to a directory server using SSL. Since LDAP directories store essential and sensitive applications and business information, the communication should be secured.
- ▶ JMS/SSL is the secure communication for JMS connections using SSL.
- ▶ IIOP/SSL (IIOPS) is the secure communication for IIOP connections using SSL. Two or more application servers may be communicating via IIOP, for example the EJB client and the EJB container.

Note: Two application servers can also communicate via HTTP with SOAP using the Web services technology. The HTTP communication should be secured using SSL.

- ▶ SSL is a transport layer security protocol that can be applied to most of the protocols in use with an e-business application. Other connections without named protocols can also use SSL to secure the communication.

Other communication channels between nodes can be secured on a transport layer, for example using IPSec.

System hardening

In addition to protecting the nodes from being attacked from outside, systems have to be secured from inside attacks as well. Operating systems security is an essential part of every system and should be mandatory. System hardening is a global philosophy of system security that focuses not only on detection, but also on prevention. It involves removing unnecessary services from the base operating system, restricting user access to the system, enforcing password restrictions, controlling user and group rights, and enabling system accounting.

System administrators are responsible for following the system and corporate guidelines to ensure security at every level. System security has to be maintained and set correctly. Part of system security is hardening the system and preventing attacks from inside and outside.

System hardening relies on the system management guidelines and the advanced security settings and functions provided by the system.

Applications have to be in sync with system security. However, sometimes the applications require some flexibility on the security side, perhaps because of unresolved design issues or special requirements. These cases can open security holes or can weaken the system security if they are not monitored nor maintained correctly.

9.4 Application security

The notion of security has several aspects:

- ▶ Identification is the process by which users or programs that communicate with each other provide their identity.
- ▶ Authentication is the process used to validate the identity information provided by the communication partner.

- Authorization is the process by which a program determines whether a given identity is permitted to access a resource, such as a file or application component.

The J2EE specifications describe the concepts to be used for these processes. Although data integrity, confidentiality, non-repudiation, and auditing are also important aspects of security, the J2EE specifications do not address them in any detail.

J2EE specifies a component programming model for security of both application programming interfaces (APIs) and declared properties. J2EE does not provide a security policy.

The security APIs used in the logic of application programs are referred to as *programmatically security*.

The declared security properties, called *declarative security*, are found in components' deployment descriptors.

One objective of the J2EE programming model is to encourage the use of declarative security, which is enforced by the container. This removes much of the responsibility for security from the application developer.

The Java 2 Enterprise Edition (J2EE) specification defines the building blocks and elements of a J2EE application that builds an enterprise application. The specification also provides details on security related to the different elements.

The J2EE application consists of multiple modules and components. These elements are connected to each other and communicate via certain protocols. This section only discusses the connection on the application level, without going into details on protocols.

Figure 9-3 on page 195 illustrates most of the elements in a J2EE application and their relationships. The arrows indicate connections between elements; these are the connections and connection groups that have to be secured in a J2EE application.

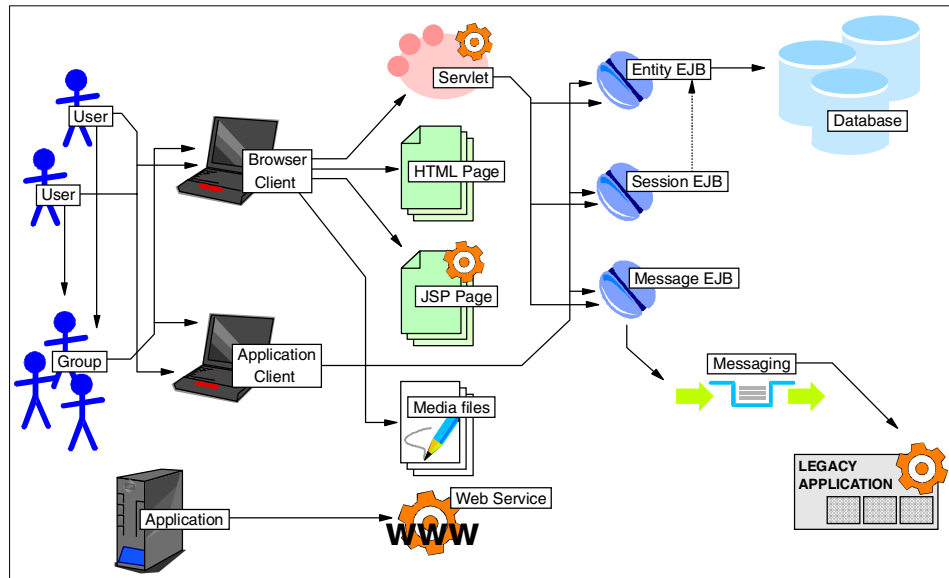


Figure 9-3 J2EE application

For example, the user wants to access a JSP on the application server. This JSP is a secured resource. In this situation the application server has to authenticate the user and decide whether the user is authorized to access the JSP or not. In this instance, the connection between the user's browser and the JSP required security.

In another example, a servlet in the Web container on the application server accesses an EJB in the EJB container on the application server. The application server has to authenticate the servlet's request on behalf of the EJB, and then check the authorization.

When you design an enterprise application or security for an application, you will have a similar but more detailed diagram for your solution. Make sure that you have taken every connection into consideration between each element and module. Security in this context consists of two major parts: authentication and authorization. Make sure that the access is always authenticated or the security credentials are propagated. Also make sure that the access is authorized and be prepared with an action if authorization is not granted.

For more information, read the security-related sections of the Java 2 Platform Specification V1.3 at:

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

J2EE container-based security

J2EE containers are responsible for enforcing access control on component objects and methods. Containers provide two types of security:

- ▶ Declarative security
- ▶ Programmatic security

Declarative security

Declarative security is the means by which an application's security policies can be expressed externally to the application code. At application assembly time, security policies are defined in an application's *deployment descriptor*. A deployment descriptor is an XML file that includes a representation of an application's security requirements, including the application's security roles, access control, and authentication requirements

When using declarative security, application developers are free to write component methods that are completely unaware of security. By making changes to the deployment descriptor, an application's security environment can be radically changed without requiring any changes in application code.

Programmatic security

Programmatic security is used when an application must be "security aware". For instance, a method might need to know the identity of the caller for logging purposes, or it might perform additional actions based on the caller's role. The J2EE Specification provides an API that includes methods for determining both the caller's identity and the caller's role.

9.5 Messaging security

Security services are the services within a computer system protecting resources. There are five security services that are identified in this book regarding the messaging security:

- ▶ Identification and authentication
- ▶ Access control
- ▶ Confidentiality
- ▶ Data integrity
- ▶ Non-repudiation

Security mechanisms are technical tools and techniques that are used to implement security services. A mechanism might operate by itself, or in

conjunction with others, to provide a particular service. Examples of common security mechanisms are:

- ▶ Access control lists
- ▶ Cryptography
- ▶ Digital signatures

When you are planning a messaging implementation, you need to consider which security services and mechanisms you need. For information about what to consider, see *WebSphere MQ Security*, SC34-6079.

For more information about the IBM Security Architecture, see *IBM Security Architecture: Securing the Open Client/Server Distributed Enterprise*, SC28-8135.

Identification and authentication

Identification is being able to identify uniquely a user of a system or an application that is running in the system. Authentication is being able to prove that a user or application is genuinely who that person or application claims to be. For example, consider a user who logs on to a system by entering a user ID and password. The system uses the user ID to identify the user and, at the time of logon, authenticates the user by checking that the supplied password is correct. Here is an example of the identification and authentication service in a Messaging environment:

- ▶ Every message can contain message context information. This information is held in the message descriptor and can be generated by the queue manager when a message is put on a queue by an application.
- ▶ Alternatively, the application can supply the information if the user ID associated with the application is authorized to do so. The context information in a message allows the receiving application to find out about the originator of the message. It contains, for example, the name of the application that put the message and the user ID associated with the application.

Access control

The access control service protects critical resources in a system by limiting access only to authorized users and their applications. It prevents the unauthorized use of a resource or the use of a resource in an unauthorized manner.

Here are some examples of the access control service in a messaging environment:

- ▶ Allowing only an authorized administrator to issue commands to manage messaging resources.
- ▶ Allowing an application to connect to a queue manager only if the user ID associated with the application is authorized to do so.
- ▶ Allowing a user's application to open only those queues that are necessary for its function.
- ▶ Allowing a user's application to perform only those operations on a queue that are necessary for its function. For example, an application might need only to browse messages on a particular queue, and not to put or get messages.

Confidentiality

The confidentiality service protects sensitive information from unauthorized disclosure. When sensitive data is stored locally, access control mechanisms might be sufficient to protect it on the assumption that the data cannot be read if it cannot be accessed. If a greater level of security is required, the data can be encrypted.

Sensitive data should be encrypted when it is transmitted over a communications network, especially over an unsecure network such as the Internet. In a networking environment, access control mechanisms are not effective against attempts to intercept the data, such as wiretapping.

Here are some examples of the confidentiality service that can be implemented in a messaging environment:

- ▶ After a sending partner gets a message from a transmission queue, the message is encrypted before it is sent over the network to the receiving partner. At the other end of the channel, the message is decrypted before the receiving partner puts it on its destination queue.
- ▶ While messages are stored on a local queue, the access control mechanisms provided by the message system might be considered sufficient to protect their contents against unauthorized disclosure. However, for a greater level of security, their contents can be encrypted as well.

Data integrity

The data integrity service detects whether there has been unauthorized modification of data. There are two ways in which data might be altered: accidentally, through hardware and transmission errors, or because of a deliberate attack. Many hardware products and transmission protocols now have mechanisms to detect and correct hardware and transmission errors. The purpose of the data integrity service is to detect a deliberate attack.

The data integrity service aims only to detect whether data has been modified. It does not aim to restore data to its original state if it has been modified.

Access control mechanisms can contribute to data integrity insofar as data cannot be modified if access is denied. But, as with confidentiality, access control mechanisms are not effective in a networking environment.

Here are some examples of the data integrity service that can be implemented in a messaging environment:

- ▶ A data integrity service can be used to detect whether the contents of a message have been deliberately modified while it was being transmitted over a network.
- ▶ While messages are stored on a local queue, the access control mechanisms provided by the messaging system might be considered sufficient to prevent deliberate modification of the contents of the messages. However, for a greater level of security, a data integrity service can be used to detect whether the contents of a message have been deliberately modified between the time the message was put on the queue and the time it was retrieved from the queue.

Non-repudiation

The non-repudiation service can be viewed as an extension to the identification and authentication service. In general, non-repudiation applies when data is transmitted electronically - for example, an order to a stock broker to buy or sell stock, or an order to a bank to transfer funds from one account to another. The overall goal is to be able to prove, with virtually 100% certainty, that a particular message is associated with a particular individual.

The non-repudiation service can contain more than one component, where each component provides a different function. If the sender of a message ever denies sending it, the non-repudiation service with proof of origin can provide the receiver with undeniable evidence that the message was sent by that particular individual. If the receiver of a message ever denies receiving it, the non-repudiation service with proof of delivery can provide the sender with undeniable evidence that the message was received by that particular individual.

In practice, proof with virtually 100% certainty, or undeniable evidence, is a difficult goal. In the real world, nothing is fully secure. Managing security is more concerned with managing risk to a level that is acceptable to the business. In such an environment, a more realistic expectation of the non-repudiation service is to be able to provide evidence that is admissible, and supports your case, in a court of law.

Non-repudiation is a relevant security service in a messaging environment because messaging-based systems are a means of transmitting data electronically. For example, you might require contemporaneous evidence that a particular message was sent or received by an application associated with a particular individual.

9.5.1 Securing WebSphere MQ resources

WebSphere MQ, V5.3 brings new performance levels, enhanced security, and added features to enhance cross-platform consistency (harmonization).

Security enhancements

Secure Sockets Layer (SSL) provides a comprehensive solution for security problems with:

- ▶ 128-bit encryption to prevent eavesdropping
- ▶ Message integrity checking to prevent tampering
- ▶ Authentication to prevent impersonation
- ▶ A range of cryptographic algorithms
- ▶ Support for public/private keys
- ▶ No key distribution problems

SSL is a protocol designed to allow the transmission of secure data over an unsecure network. Widely accepted in the Internet community, SSL has been subjected to significant testing by the “hacker” community.

SSL makes use of digital certificates to enable authentication of the partner. It also uses encryption to prevent eavesdropping and hash functions to enable detection of tampering. SSL can be used on channels in WebSphere MQ through new parameters introduced in WebSphere MQ V5.3. It can be used with both MCA channels for queue manager to queue manager communication and MQI channels for client applications connecting to a queue manager. A digital certificate must be obtained for each queue manager and each client user ID that wishes to communicate over an SSL secured channel.

WebSphere MQ now provides built-in functions to solve common security problems. Users can specify which symmetric key cryptography algorithm and which hash function to use by providing WebSphere MQ with a CipherSpec. Digital certificates and public keys are found in a key ring, which can be specified to WebSphere MQ. Users also can check that they are talking to the partner they expect to be talking to and can choose to authenticate both ends of the connection or only the SSL Server end of the connection. Certificate revocation lists are also available to cancel security certificates.

Wildcards can now be used with security settings. With WebSphere MQ V5.3, you can define groups of objects to which users have access. New objects will inherit these definitions automatically. The wildcard matching, based on qualifiers with “.” (dot) separators, is based on RACF®, and migration from the old Object Access Manager is transparent.

Extensions are provided to the security management commands and to the interface to the installable service that implements authorization checks, to display all of the users or groups who have access to particular objects, or to allow easy cloning of settings to new queue managers.

WebSphere MQ security facilities

WebSphere MQ offers security facilities as part of the distributed services layer, which can be used or invoked by the application.

The WebSphere MQ facilities include the following:

- ▶ Access control
Checks when accessing queue manager resources and commands against the user ID under which the application program is running. Uses an external security manager supplied by the application enabling services.
- ▶ Message context
Contextual information is contained within the descriptor part of each message describing who generated the original request and where this specific message came from.
- ▶ MCA exits
Exit programs can be attached to the message channel agents. This exit facility has been designed to allow security-related functions to be implemented in the exit routines associated with links between systems.
- ▶ Security management and audit
Facilities allow system administrators to set up and manage the various security operations, and to verify that the security facilities are working in the expected manner.

Basic considerations

Because WebSphere MQ queue managers handle the transfer of information that is potentially valuable, you need the safeguard of an authority system. This ensures that the resources that a queue manager owns and manages are protected from unauthorized access, which could lead to the loss or disclosure of the information.

In a secure system, it is essential that none of the following are accessed or changed by any unauthorized user or application:

- ▶ Connections to a queue manager
- ▶ Access to WebSphere MQ objects such as queues, clusters, channels, and processes
- ▶ Commands for queue manager administration, including MQSC commands and PCF commands
- ▶ Access to WebSphere MQ messages
- ▶ Context information associated with messages

The basic considerations are those aspects of security you must consider when implementing WebSphere MQ. On UNIX and Windows systems, if you ignore these considerations and do nothing, you cannot implement WebSphere MQ. On z/OS, the most likely effect is that your WebSphere MQ resources are unprotected. That is, all users can access and change all WebSphere MQ resources.

Authority to administer WebSphere MQ

WebSphere MQ administrators need authority to:

- ▶ Issue commands to administer WebSphere MQ
- ▶ Use the WebSphere MQ Explorer and the WebSphere MQ Services snap-in on Windows systems
- ▶ Use the operations and control panels on z/OS
- ▶ Use the WebSphere MQ utility program, CSQUTIL, on z/OS
- ▶ Access the queue manager data sets on z/OS

This is an aspect of access control. For more information, see “Authority to administer WebSphere MQ” in *WebSphere MQ Security*, SC34-6079.

Authority to work with WebSphere MQ objects

Applications can access the following WebSphere MQ objects by issuing MQI calls:

- ▶ Queue managers
- ▶ Queues
- ▶ Processes
- ▶ Namelists

On UNIX and Windows systems, applications can also use Programmable Command Format (PCF) commands to access these WebSphere MQ objects, and to access authentication information objects as well. These objects are

protected by WebSphere MQ and the user IDs associated with the applications need authority to access them.

This is another aspect of access control. For more information, see “Authority to work with WebSphere MQ objects” in *WebSphere MQ Security*, SC34-6079.

Channel security

The user IDs associated with message channel agents (MCAs) need authority to access various WebSphere MQ resources. For example, an MCA must be able to connect to a queue manager. If it is a sending MCA, it must be able to open the transmission queue for the channel. If it is a receiving MCA, it must be able to open destination queues. On UNIX and Windows systems, the user IDs associated with applications need authority to use PCF commands to administer channels, channel initiators, and listeners.

This is another aspect of access control. For more information, see “Channel security” in *WebSphere MQ Security*, SC34-6079.

Additional considerations

The following are aspects of security you need to consider only if you are using certain WebSphere MQ functions or base product extensions:

- ▶ Queue manager clusters
- ▶ WebSphere MQ publish/subscribe
- ▶ WebSphere MQ Internet Passthrough

Queue manager clusters

A queue manager cluster is a network of queue managers that are logically associated in some way. A queue manager that is a member of a cluster is called a cluster queue manager.

A queue that belongs to a cluster queue manager can be made known to other queue managers in the cluster. Such a queue is called a cluster queue. Any queue manager in a cluster can send messages to cluster queues without needing any of the following:

- ▶ An explicit remote queue definition for each cluster queue
- ▶ Explicitly defined channels to and from each remote queue manager
- ▶ A separate transmission queue for each outbound channel

You can create a cluster in which two or more queue managers are clones. This means that they have instances of the same local queues, including any local queues declared as cluster queues, and can support instances of the same server applications.

When an application connected to a cluster queue manager sends a message to a cluster queue that has an instance on each of the cloned queue managers, WebSphere MQ decides which queue manager to send it to. When many applications send messages to the cluster queue, WebSphere MQ balances the workload across each of the queue managers that have an instance of the queue. If one of the systems hosting a cloned queue manager fails, WebSphere MQ continues to balance the workload across the remaining queue managers until the system that failed is restarted.

If you are using queue manager clusters, you need to consider the following security issues:

- ▶ Allowing only selected queue managers to send messages to your queue manager
- ▶ Allowing only selected users of a remote queue manager to send messages to a queue on your queue manager
- ▶ Allowing applications connected to your queue manager to send messages only to selected remote queues

These considerations are relevant even if you are not using clusters, but they become more important if you are using clusters.

If an application can send messages to one cluster queue, it can send messages to any other cluster queue without needing additional remote queue definitions, transmission queues, or channels. It therefore becomes more important to consider whether you need to restrict access to the cluster queues on your queue manager, and to restrict the cluster queues to which your applications can send messages.

There are some additional security considerations, which are relevant only if you are using queue manager clusters:

- ▶ Allowing only selected queue managers to join a cluster
- ▶ Forcing unwanted queue managers to leave a cluster

For more information about all these considerations, see *WebSphere MQ Queue Manager Clusters*. For considerations specific to WebSphere MQ for z/OS, see the *WebSphere MQ for z/OS System Setup Guide* product documentation.

WebSphere MQ publish/subscribe

In a publish/subscribe scenario, there are two types of applications: publisher and subscriber. Publishers supply information in the form of WebSphere MQ messages. When a publisher publishes a message, it specifies a topic, which identifies the subject of the information inside the message.

Subscribers are the consumers of the information that is published. A subscriber specifies the topics it is interested in by sending a subscription request to a broker in the form of a WebSphere MQ message.

The broker is an application supplied with WebSphere MQ publish/subscribe. It receives published messages from publishers and subscription requests from subscribers, and routes the published messages to the subscribers. A subscriber is sent messages only on those topics to which it has subscribed.

There are additional security considerations if you are using WebSphere MQ publish/subscribe. The user IDs associated with publishers and subscribers need authority to access the queues that they use to communicate with a broker. For more information, see the *WebSphere MQ Publish/Subscribe User's Guide* product documentation.

WebSphere MQ internet Passthrough

WebSphere MQ Internet Passthrough enables two queue managers to exchange messages, or a WebSphere MQ client application to connect to a queue manager, over the Internet without requiring a direct TCP/IP connection. This is useful if a firewall prohibits a direct TCP/IP connection between two systems. It makes the passage of WebSphere MQ channel protocol flows into and out of a firewall simpler and more manageable by tunnelling the flows inside HTTP or by acting as a proxy. Using the Secure Sockets Layer (SSL), it can also be used to encrypt and decrypt messages that are sent over the Internet.

Link level security and application level security

The remaining security considerations are discussed under two headings: link level security and application level security.

Link level security

Link level security refers to those security services that are invoked, directly or indirectly, by an MCA, the communications subsystem, or a combination of the two working together. This is illustrated in Figure 9-4 on page 206.

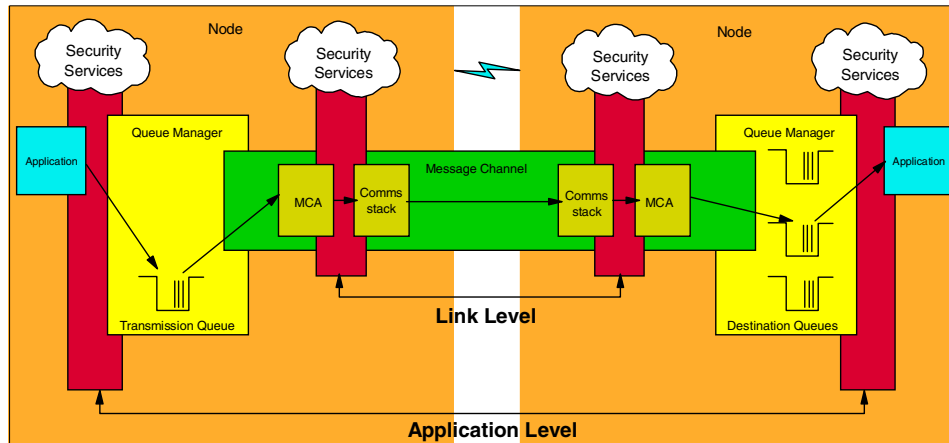


Figure 9-4 Link level security and application level security

Here are some examples of link level security services:

- ▶ The MCA at each end of a message channel can authenticate its partner. This is done when the channel starts and a communications connection has been established, but before any messages start to flow. If authentication fails at either end, the channel is closed and no messages are transferred. This is an example of an identification and authentication service.
- ▶ A message can be encrypted at the sending end of a channel and decrypted at the receiving end. This is an example of a confidentiality service.
- ▶ A message can be checked at the receiving end of a channel to determine whether its contents have been deliberately modified while it was being transmitted over the network. This is an example of a data integrity service.

Application level security

Application level security refers to those security services that are invoked at the interface between an application and a queue manager to which it is connected. These services are invoked when the application issues MQI calls to the queue manager. The services might be invoked, directly or indirectly, by the application, the queue manager, another product that supports WebSphere MQ, or a combination of any of these working together. Application level security is illustrated in Figure 9-4.

Application level security is also known as end-to-end security or message level security. Here are some examples of application level security services:

- ▶ When an application puts a message on a queue, the message descriptor contains a user ID associated with the application. However, there is no data present, such as an encrypted password, that can be used to authenticate the

user ID. A security service can add this data. When the message is eventually retrieved by the receiving application, another component of the service can authenticate the user ID using the data that has travelled with the message. This is an example of an identification and authentication service.

- ▶ A message can be encrypted when it is put on a queue by an application and decrypted when it is retrieved by the receiving application. This is an example of a confidentiality service.
- ▶ A message can be checked when it is retrieved by the receiving application. This check determines whether its contents have been deliberately modified since it was first put on a queue by the sending application. This is an example of a data integrity service.

9.5.2 Securing WebSphere MQ Integrator resources

WebSphere MQ Integrator exploits WebSphere MQ and the operating system facilities to control security for components and tasks:

Topic-based security

The WebSphere MQ Integrator User Name Server interacts with the operating system security system to control user and group access to publications and subscriptions.

Operational control of components

WebSphere MQ Integrator uses the operating system access control. Operational roles are used in the Control Center.

Security control of WebSphere MQ Integrator components, resources, and tasks depends on the definition of users and groups of users (principals) to the security subsystem of the operating system (the Windows User Manager or the UNIX user/group database).

When WebSphere MQ Integrator is installed, it automatically creates five groups in the operating system's security mechanism:

- ▶ mqbrkrs
- ▶ mqbrasgn
- ▶ mqbrdevt
- ▶ mqbrops
- ▶ mqbrtpic

Assigning a user to one or more of these groups determines the WebSphere MQ Integrator tasks they are allowed to perform.

WebSphere MQ Integrator depends on a number of WebSphere MQ resources to operate successfully. You must control access to these resources to ensure

that WebSphere MQ Integrator can access the resources it needs, while limiting access to other users. Some authorizations are granted automatically on your behalf when commands are issued, primarily the authority to put messages on a queue and to retrieve (get) messages from a queue. Others depend on the configuration of your broker domain. The transmission queues handling the message traffic between the WebSphere MQ Integrator component queue managers must have put and setall authority granted to the local mqbrkrs group or to the service user ID of the WebSphere MQ Integrator component. When you create, assign, and deploy a message flow, you must grant the following:

- ▶ GET authority to each input queue identified in an MQInput node, for the broker's service user ID.
- ▶ PUT authority to each output queue identified in an MQOutput node, or by an MQReply node, for the broker's service user ID.
- ▶ GET authority to each output queues identified in an MQOutput node or an MQReply node to the user ID under which a receiving or subscribing client application runs.
- ▶ PUT authority to each input queue identified in an MQInput node to the user ID under which a sending or publishing client application runs. WebSphere MQ Integrator security is discussed in more detail in *WebSphere MQ Integrator Administration Guide*, SC34-5792.

Database security

The Configuration Manager service user ID must be authorized for create and update tasks on the database in which both configuration and message repositories are defined.

Each broker service user ID must be authorized for create and update tasks on the database that contains the broker internal tables. Each broker service user ID must also be authorized for the appropriate access for every database referenced and accessed by a message processing node in any deployed message flow.

Of course, access to the above databases must be controlled and limited to the designated Configuration Manager, broker and User Name Server service IDs.

Application security

When you deploy a message flow on one or more brokers, applications can start to feed messages into the message flow by putting messages to the queue that is identified as the input queue. You set up the association between the input node and the queue by setting the queue name as a property of the node.

Similarly, applications access queues to receive messages placed on those queues by MQOutput or publication nodes, when the message flow has completed processing for those messages.

The user IDs under which applications are executing must therefore be authorized to write to, or read from, the queues used by the message flow the applications are interacting with.

9.6 Security design principles summary

The following security guidelines should be used when designing, architecting an application or infrastructure:

- ▶ No business logic, data in DMZ.
- ▶ No user state maintained on Internet connected systems (only connection state).
- ▶ Separate authentication domain from authorization domain.
- ▶ Use frameworks, standards whenever you can.
- ▶ Design for multiple authentication mechanisms.
- ▶ Assume that the desktop is untrusted.
- ▶ Dual controls, separation of authority.
- ▶ Least privilege, role base access controls.
- ▶ Isolate systems and networks of different vulnerability levels.



Performance and availability

In this chapter, we identify some of the most important considerations in performance when working in transactional and messaging-based applications.

10.1 Introduction

The use of patterns can have an important impact on performance. In this chapter, we present some performance gains or losses when using these different patterns, and offer some general guidelines on how to choose one pattern over the other.

10.2 Performance analysis

One of the major problems with multiple-tier applications is identifying performance bottlenecks in a runtime environment. Bottlenecks are usually hard to identify during the programming of the application, since they might depend on workloads and concurrency, for example.

WebSphere Studio Application Developer V5 and WebSphere Application Server V5 offer great profiling capabilities that can help to identify bottlenecks and eliminate them.

For further information on the profiling functionality in WebSphere Studio, please refer to the redbook *WebSphere Studio Application Developer Programming Guide*, SG24-6585.

10.3 Performance considerations in messaging

This section gives a general overview of performance considerations of messaging-based applications, and provides information related to Quality of Service and reliability of messaging applications.

10.3.1 Connection pooling

WebSphere MQ 5.3 provides connection pooling to significantly improve the underlying performance in getting a connection to a queue manager. When a connection is no longer required, instead of destroying it, it can be pooled and later reused. This can provide a substantial performance enhancement for applications and middleware that connect serially to arbitrary queue managers.

The scope of the pooling is global for the JVM. If many JMS sessions are simultaneously active, the connections will be pooled between all of them. However, when the number of sessions falls to zero, the pooling is disabled and all pooled connections are closed. Hence, if one session at a time is sequentially used there will be no pooled connections persisted between them.

The following code uses the `MQSimpleConnectionManager` class to replace the default connection manager. The connection manager constructed will keep up to 100 unused connections alive for reuse for up to 1000 milliseconds and will always return a connection to the pool.

```
myConMgr = new MQSimpleConnectionManager();
myConMgr.setTimeout(1000);
myConMgr.setHighThreshold(100);
myConMgr.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQEnvironment.setDefaultConnectionManager(myConMgr);
```

There is no hard limit on the number of connections in use. It is possible to have more connections in use than you have your connection pool configured to. This is because the pool is only keeping unused connections and not keeping track of any in-use connections. When a connection is closed and returned to the pool, if the pool is already at maximum size, the oldest connection in the pool is removed. With no hard limit on the number of connections in use, it is possible to exhaust system resources if you have too many connections running at one time.

Default connection pool

Consider the following example application:

Example 10-1 MQApp1.java code snippet

```
import com.ibm.mq.*;
public class MQApp1 {
    public static void main(String[] args) throws MQException {
        for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            // do something with qmgr
            mgr.disconnect();
        }
    }
}
```

The above snippet class `MQApp1` takes a list of local queue managers from the command line, connects to each in turn, and performs some operation. However, when the command line lists the same queue manager many times, it is more efficient to connect only once, and to reuse that connection many times.

WebSphere MQ base Java provides a default connection pool that you can use to do this. To enable the pool, use one of the `MQEnvironment.addConnectionPoolToken()` methods. To disable the pool, use `MQEnvironment.removeConnectionPoolToken()`.

The example application, `MQApp2`, in Example 10-2 on page 214 is functionally identical to `MQApp1`, but connects only once to each queue manager.

Example 10-2 MQApp2.java

```
import com.ibm.mq.*;
public class MQApp2 {
    public static void main(String[] args) throws MQException {
        MQPoolToken token=MQEnvironment.addConnectionPoolToken();
        for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            // do something with qmgr
            qmgr.disconnect();
        }
        MQEnvironment.removeConnectionPoolToken(token);
    }
}
```

The addConnectionPoolToken object activates the default connection pool by registering an MQPoolToken object with MQEnvironment. The MQQueueManager constructor now searches this pool for an appropriate connection and only creates a connection to the queue manager if it cannot find an existing one. The qmgr.disconnect() call returns the connection to the pool for later reuse. These API calls are the same as the sample application MQApp1.

The removeConnectionPoolToken object deactivates the default connection pool, which destroys any queue manager connections stored in the pool. This is important because otherwise the application would terminate with a number of live queue manager connections in the pool. This situation could cause errors that would appear in the queue manager logs. The default connection pool stores a maximum of 10 unused connections, and keeps unused connections active for a maximum of five minutes.

Instead of using MQEnvironment to supply an MQPoolToken, the application can construct its own:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Some applications or middleware vendors provide subclasses of MQPoolToken in order to pass information to a custom connection pool. They can be constructed and passed to addConnectionPoolToken() in this way so that extra information can be passed to the connection pool.

10.3.2 Multithreaded programs

Multithreaded programs are hard to avoid in Java. The Java runtime environment is inherently multithreaded. Therefore, your application initialization occurs in one thread, and the code that executes in response to the button press executes in a separate thread (the user interface thread).

With the “C” based WebSphere MQ client, this would cause a problem, because handles cannot be shared across multiple threads. WebSphere MQ classes for Java relaxes this constraint, allowing a queue manager object (and its associated queue and process objects) to be shared across multiple threads.

The implementation of WebSphere MQ classes for Java ensures that, for a given connection (MQQueueManager object instance), all access to the target WebSphere MQ queue manager is synchronized. Therefore, a thread wishing to issue a call to a queue manager is blocked until all other calls in progress for that connection are complete. If you require simultaneous access to the same queue manager from multiple threads within your program, create a new MQQueueManager object for each thread that requires concurrent access. (This is equivalent to issuing a separate MQCONN call for each thread.)

10.3.3 Persistent versus non-persistent messages

WebSphere MQ queue can hold persistent and non-persistent messages. Persistent messages are always written directly to disk, and are restored to the queue if the queue goes down for some reason. This type of message is resilient, but there is a performance cost because of the disk access.

The performance of your application is affected when you use persistent messages; the extent of the effect depends on the performance characteristics of the machine’s I/O subsystem and how you use the syncpoint options on each platform.

If you wish to do transactional writing to the queue then you will want this level of resilience too. JMS does not enforce the type of message it writes in a transaction so you must ensure that the message is persistent yourself. When using JMSAdmin, persistent messages are the default. If you do not explicitly set the persistence property of the queue, it will use the default, PERSISTENCE(APP), that is, the application determines persistence for the queue in JMSAdmin. The default delivery mode for JMS messages is also persistent. This property is set on the QueueSender, or specified on the call to the send method, not directly on the message. If you want to change this, follow the steps below:

1. Set the persistence in the definition of the Queue, using JMSAdmin and/or
2. Use setDeliveryMode on the QueueSender or on the send method. Setting it on the message will have no effect when sending messages.

Persistent messaging

When using persistent messaging the likely constraining factor will usually be WebSphere MQ logging. Table 10-1 on page 216 shows the maximum message rates achieved in the tests. The number of round trips was very much dependent

on the message size. IBM SSA cached disks have been shown to significantly improve (by up to a factor of two) persistent messaging on WebSphere MQ for AIX.

Table 10-1 Persistent messaging size

Message Size	Total round trips per second
2K Bytes	25 - 47
20K Bytes	19 - 21
50K Bytes	14 - 14

Nonpersistent messaging

The maximum number of clients that can be supported on a single queue manager is 2300. A design limitation in WebSphere MQ requires that additional queue managers be used to go beyond this point. The message rates achieved varied according to the message size and the number of clients being supported. When using a high number of clients, context switching between clients consumes CPU.

Assuming the maximum number of clients is not exceeded, it is likely that a CPU or storage (RAM) constraint will be encountered when using nonpersistent messages. A combination of the number of clients, the message size, message rate and amount of RAM indicates the outcome.

Table 10-2 Non-persistent messaging

Message size	Total round trips per second
2 KB	78 - 300
20 KB	66 - 140
50 KB	56 - 89

10.3.4 One-phase commit optimization

Using a two-phase commit transaction incurs a performance overhead. However, if applicable, WebSphere will optimize this by using one-phase commit where possible. When a transaction is due to be committed, the WebSphere transaction coordinator will check to see if, in fact, a transaction is being performed across two resources. If only one resource is being used, then it will automatically perform a one-phase commit (1-PC) and so optimize transaction performance.

10.3.5 Caching WebSphere MQ JMS objects

It is likely that your application will do more than just write one message to a message queue. Applications that can cache some JMS objects in the beans would provide some performance enhancements. If you are intending to do this, then you need to be aware that some of the objects are not thread safe.

Table 10-3 gives the thread safety of JMS objects.

Table 10-3 JMS objects

JMS Object	Thread Safe
Connection Factory	Yes
Connection	Yes
Queue / Topic	Yes
Session	No
Message Producer	No
Message Consumer	No
Message	No

10.3.6 Message-driven beans performance considerations

Message-driven beans are deployed as clients to a particular topic or queue that always plays the role of a message consumer. There is no client view to a message-driven bean, meaning that there are no home and remote interfaces. A message producer writes to a topic or queue and has no knowledge of the fact that a message-driven bean is acting as a message consumer. This leads to loose coupling between JMS-based systems, allowing for more flexibility in assembling a distributed computing environment.

The life cycle of a message-driven bean corresponds to the lifespan of the EJB server in which it is deployed. Since message-driven beans are stateless, bean instances are typically pooled by the EJB server and retrieved by the container when a message becomes available on the topic or queue for which it is acting as a message consumer. The container creates a bean instance by invoking the `newInstance()` method of the bean instance class object. After the instance is created, the container creates an instance of `javax.ejb.MessageDrivenContext` and passes it to the bean instance via the `setMessageDrivenContext()` method. The bean instance is then placed in the pool and is available to process messages.

Message-driven beans - threading and concurrency

A message-driven bean instance is assumed to execute in a single thread of control, which greatly simplifies the bean developer's task. The EJB server will guarantee this behavior. In addition, the EJB server may provide a mode of operation that allows multiple messages to be handled concurrently by separate bean instances. This deployment option uses “expert level” classes defined in the JMS specification. The JMS provider is not required to provide implementations for these classes, so the EJB server may not be able to take advantage of them with every JMS implementation. Using these classes involves a trade-off between performance and serialization of messages delivered to the server.

10.4 High availability with WebSphere MQ

Clusters are not a new concept to IT. We have had High Availability Cluster Multi-Processing (HACMP) clusters for some time now, and most IT shops design most of their systems/applications to provide high availability to their users so that no server or service is unavailable at any given time. The Distributed Computing Environment (DCE) accomplishes this same availability using the notion of “cells” and a naming service. A DCE service can be located on multiple host systems within a DCE cell. Client applications can access or bind to these services in one of three ways:

- ▶ Automatic allows the Remote Procedure Call (RPC) library to find the host and service to handle your request.
- ▶ Implicit will allow the client application some control over where the service is selected.
- ▶ Explicit allows the client application full control over the host and service for each RPC call made.

Cluster

Clustering in its simplest form means logically grouping two or more components and making them appear as one to a consumer of the components. There are two types of clustering available to provide high availability solutions: shared cluster and shared-nothing cluster.

Shared cluster

A shared cluster generally involves creating redundant, shareable components for the purpose of high availability, fault tolerance and load balancing. Typically, system resources that can be shared include disk storage and CPU/applications. When disk storage is clustered, the operating system must manage locking conditions across the CPUs to maintain the integrity of the data being accessed.

This is no simple task, and is the reason many vendors provide only shared-nothing clustering solutions.

Shared-nothing cluster

A shared-nothing cluster (such as Microsoft's Cluster Server) does not share any components, but usually involves replication of data (in a timely manner) that can be used by a hot standby system/application. In this scenario, clustering provides fail-over capabilities, but not load-balancing.

Clustered applications in a shared-nothing environment are a little different than their shared cluster counterparts. Since no common data is shared among the applications, each instance of the application has access to a replicated copy of configuration and/or production data. WebSphere MQ clustering works on this concept of replicated data. WebSphere MQ object definitions are stored in a repository queue and replicated, either in full or partially, to replication partners known as repository queue managers.

Clustering versus distributed queuing

Clustering offers several benefits over distributed queuing. By clustering and sharing data, the WebSphere MQ network becomes more dynamic and more manageable. As the WebSphere MQ network grows, the risk of errors increases, and the flexibility decreases. With clustering however, the risk of errors can be greatly reduced, while maintaining the flexibility to modify your configuration on the fly.

Clustering also provides a simple way to provide fault-tolerance and high availability. The product provides built-in, customizable logic for routing messages to destination queues. You can use the default method for message distribution, or create customized "workload exits" for advanced routing algorithms.

In a distributed queuing architecture, this logic has to be coded into the applications that are sending the messages. In addition, error-handling routines need to be developed to manage the condition where one or more queue managers become unavailable. Furthermore, additional code is needed to recognize when these unavailable queue managers become available once again.

WebSphere MQ clustering will dynamically create and destroy certain objects as needed, further increasing the ease of management. Specifically, the channel definitions to/from cluster queue managers are created automatically - however, the first channel pair must be defined manually. These are known as Cluster Sender (CLUSDR) and Cluster Receiver (CLUSRCVR) channels. As with all WebSphere MQ implementations, all channels are uni-directional and therefore require this pairing. Perhaps the best feature of WebSphere MQ clustering is that

existing applications do not need to be modified in order to benefit from the features of clustering. The entire cluster configuration is handled at the administration layer - entirely transparent from users and applications.

10.4.1 Overview of WebSphere MQ cluster components

In this section, we describe WebSphere MQ cluster components. Figure 10-1 on page 221 shows the components of a cluster called CLUSTER.

- ▶ In this cluster there are three queue managers: QM1, QM2, and QM3.
- ▶ QM1 and QM2 host repositories of information about the queue managers in the cluster. They are referred to as full repository queue managers. The repositories are represented in the diagram by the shaded cylinders.
- ▶ QM2 and QM3 host some queues that are accessible to any other queue manager in the cluster. These are called cluster queues.
- ▶ As with distributed queuing, an application uses the MQPUT call to put a message on a cluster queue at any queue manager. An application uses the MQGET call to retrieve messages from a cluster queue on the local queue manager.
- ▶ Each queue manager has a definition for the receiving end of a channel called TO.qmgr on which it can receive messages. This is a cluster-receiver channel. A cluster-receiver channel is similar to a receiver channel used in distributed queuing, but in addition to carrying messages this channel can also carry information about the cluster.
- ▶ Each queue manager also has a definition for the sending end of a channel, which connects to the cluster-receiver channel of one of the full repository queue managers. This is a cluster-sender channel. In Figure 10-1 on page 221, QM1 and QM3 have cluster-sender channels connecting to TO.QM2. QM2 has a cluster-sender channel connecting to TO.QM1. A cluster-sender channel is similar to a sender channel used in distributed queuing, but in addition to carrying messages this channel can also carry information about the cluster. Once both the cluster-receiver end and the cluster-sender end of a channel have been defined, the channel starts automatically.

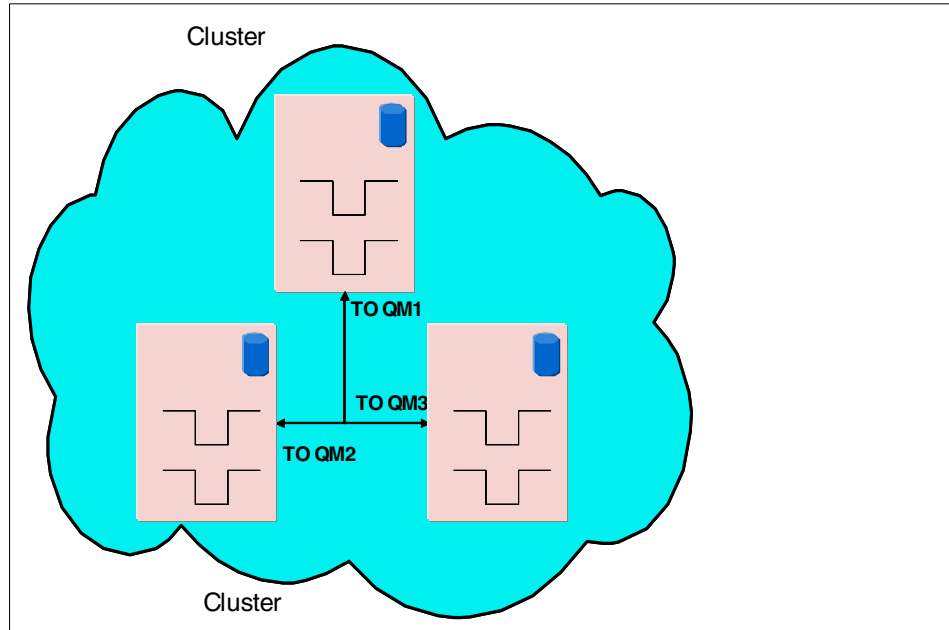


Figure 10-1 Cluster of queue managers

Cluster workload management algorithm

An application running on QM1 needs to put a message to Q1. But which queue will be the target? In this case, Q1 on QM1 will always be used. WebSphere MQ clustering uses a workload management algorithm, which operates according to the following rules:

- ▶ If a local queue exists, the message will always be sent to the local queue. That is, a queue managed by the same queue manager that is putting the message.
- ▶ If multiple instances of the queue are available, but only on non-local queue managers, then a round robin approach is used. The state of each sender channel, as well as the priority associated with the channel (NETPRTY channel attribute) is used in determining which queue is selected first.
- ▶ If an attempt fails to put the message to the selected target queue, each alternate target is attempted in succession.
- ▶ If all target queues fail, then the message is sent to the deal letter queue.

Cluster workload exits

In addition to the workload management algorithm, you can customize how workload is distributed using cluster workload exits. There are many reasons you may need to customize. For example:

- ▶ If you need to determine which channels use a high-speed connection in order to favor that channel.
- ▶ Perhaps application performance is your primary concern and you want to send messages to the least busy processor.
- ▶ What if your data is partitioned? You may have the same application running on QM1 and QM2, but each instance is responsible for a certain range of data, like customer identifier. Your workload exit could examine the message data first, and then route to the application that handles that specific customer.
- ▶ Perhaps the previous applications are designated by a quality of service (QOS) and high-value customer requests need to be routed to the system running the more reliable or faster instance of the application.

Cluster workload exits are called at open or put time. That is, when the sending application calls MQPUT, MQPUT1 or MQOPEN.

Note: Only one cluster workload exit can be defined for a given queue manager.

10.4.2 WebSphere MQ simplified management

This section gives a general management practices for WebSphere MQ regardless of operating system. Any WebSphere MQ administrator will agree that managing MQ objects in a complex network can be very cumbersome, especially without a third-party management tool such as Candle Command Center or Tivoli's MQ. The fewer the objects, the easier it is to manage both a static MQ network and a growing one.

Including remote queue definitions, a simple WebSphere MQ network with four queue managers with two local queues would require a minimum of 68 objects to be defined, broken down as:

- ▶ 3 sender channels per queue manager (12 total)
- ▶ 3 receiver channels per queue manager (12 total)
- ▶ 3 transmission queues per queue manager (12 total)
- ▶ 2 local queue definitions per queue manager (8 total)
- ▶ 6 remote queue definitions per queue manager (24 total)

In a clustered environment, the number of objects to be defined is reduced to 16. This is because only one cluster sender and cluster receiver channel is needed, and no transmission queues or remote queue definitions are necessary.

Therefore the only objects required are:

- ▶ 1 cluster sender channel per queue manager (4 total)
- ▶ 1 cluster receiver channel per queue manager (4 total)
- ▶ 2 local queue definitions per queue manager (8 total)

With this type of configuration, the risk of making errors in defining transmission queues and remote queue definitions is eliminated.



Part 3

Implementation



Technical scenarios

Two scenarios were designed to run the sample application provided with this book. The scenarios for a development environment and for a runtime environment are documented in this chapter.

Information about the application from a developers point of view can be found in 6.1.3, “Sample application” on page 99. This chapter will concentrate on runtime environment.

11.1 Application flow

The following two sections describe the steps that are performed by the system during the two process:

- ▶ Order placement, the customer places an order.
- ▶ Order pickup, the customer comes back and checks for the order; it is a step prior to order fulfillment.

Order placement

1. The customer goes to the IV Corp Web site to place an order, selects an item, and sets the amount, then submits the order.
2. The front-end application, that is responsible for the presentation and interaction with the back-end system, sends a message to the back-end application with the order details.
3. A message-driven bean at the back-end application receives the message. The order consists of one item, a color. The application generates a complex order from this item by breaking it down to sub-components, it calculates the color components (red, green, blue) for the requested color.
4. The complex message is then sent to the integration node.
5. The integration node decomposes the message, stores each sub-components in a database under the same order ID, then sends out sub-orders to the suppliers.
6. The supplier side is simulated with a simple application in this sample scenario. The application receives the order, and replies accordingly. The application simulates all three suppliers: red, green, blue color suppliers.
7. The replies from the suppliers are sent back to the integration node, where each reply is stored in the database where the original sub-component orders were registered.

Order pickup

1. The customer goes to the IV Corp Web site to pick up an order, provides an order number, and submits the form.
2. The front-end application sends a message to the back-end, which forwards it to the integration node. The integration node collects the sub-orders with the supplier replies from the database and composes a complex message, with the order reply.
3. The reply is sent back to the back-end, then forwarded to the front-end where the response page is generated.

11.2 System setup

The runtime environment used to prove the Runtime pattern consisted of the following:

1. A Web application server was used to host the front-end application. This portion of the application provides the user interface and initiates the back-end operations.
2. A Web application server was also used to host the back-end application. This application provides the order processing application logic.
3. An integration node hosts message flows. These flows are responsible for decomposing complex messages, routing messages to the appropriate back-end system, and then recomposing the incoming messages into a complex response.
4. A back-end node hosts the supplier application. This application is a stand-alone Java application, simulating multiple suppliers to supply goods for the enterprise to fulfill the incoming orders.

The applications use different databases for operation:

1. The back-end application uses the VENDOR database to store order details.
2. The integration application uses the HUB database to keep track of the decomposed orders for future re-composition.
3. The integration server also requires some databases for normal operation.

11.2.1 Products used to prove the scenarios

The development environment is a single-machine system, where you will have the integrated development environment (WebSphere Studio) and all the external runtime components for testing purposes (DB2, WebSphere MQ, WebSphere MQ Integrator).

The runtime environment is a multi-system environment with multiple systems and heterogeneous operating systems on them.

The following products are needed for the scenarios:

► **WebSphere Application Server V5**

The WebSphere Application Server provides the application server runtime for the whole solution. It runs two separate applications, one for the front-end and one for the back-end.

The front-end application is a classic Web application that servers as a user interface for back-end operations.

The back-end application is a simulated order processing system, implemented by using EJBs.

- ▶ WebSphere MQ V5.3

WebSphere MQ provides the messaging layer for the whole solution. It is utilized by all the components: application server, integration server, standalone application.

- ▶ WebSphere MQ Integrator Broker 2.1 with CSD 04

The Integrator implements most of the major functions described in this book: decomposition, routing, re-composition.

Note: WebSphere Application Server V5 has a built-in embedded messaging server, based on the WebSphere MQ runtime. This embedded messaging provides messaging capabilities only between application servers. When the application has to communicate with non-WebSphere application servers or other applications, using MQ resources, the embedded messaging becomes a limitation.

In our sample application there is a communication channel built between the front-end application and the back-end application using JMS over MQ. It could be implemented using embedded messaging, although all the rest of the solution demands external messaging. Therefore the whole solution is built on top of the external messaging provider, WebSphere MQ.

- ▶ DB2 Universal Database Workstation

DB2 is the database server for the solution. The applications use multiple databases and they are hosted either locally or remotely on a DB2 database server.

11.2.2 Development environment

The development environment is a one-machine system with all the necessary tools and runtime components for application testing and debugging.

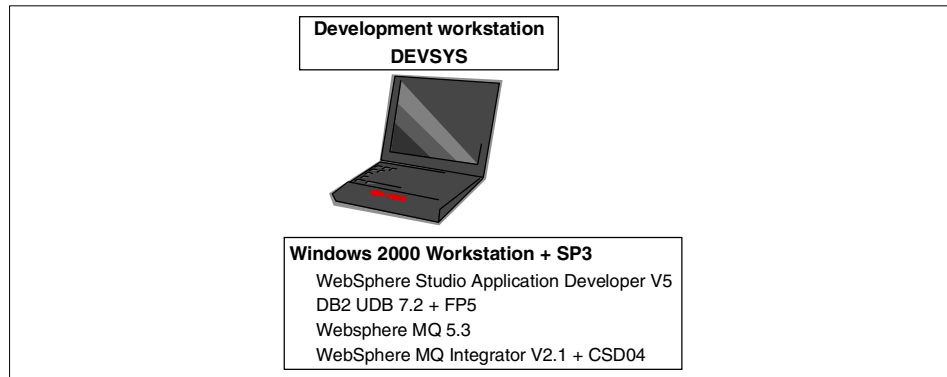


Figure 11-1 Development environment

The installation of the system does not require any special step. Follow the product documentation for each component during installation. The recommended installation order for the components is:

1. Windows 2000 Workstation
2. Windows 2000 Service Pack 3 and other critical updates
3. DB2 UDB 7.2 Workstation or Enterprise Edition with Fixpak 5 or higher.
Recommended install directory: c:\sqllib
4. WebSphere Studio Application Developer V5
Recommended install directory: c:\wsad5
5. WebSphere MQ 5.3 with Java libraries
Recommended install directory: c:\ibm\webspheremq
6. WebSphere MQ Integrator V2.1 Broker with the components: Broker, Configuration Manager, Control Center.
Recommended install directory: c:\ibm\wmqi
WebSphere MQ Integrator V2.1 Broker CSD04

11.2.3 Runtime environment

The runtime environment is a four-machine system, where the solution components are distributed between multiple servers and even multiple platforms.

Figure 11-2 on page 232 depicts the systems involved in the solution with the operating system and installed components on them. One of the application servers and the database server is running an AIX 5L™ operating system, while

the other systems are running Windows 2000 Server. Varying the systems is not necessary, but we did that to prove that the solution can run on multiple platforms and to show the differences in the configuration.

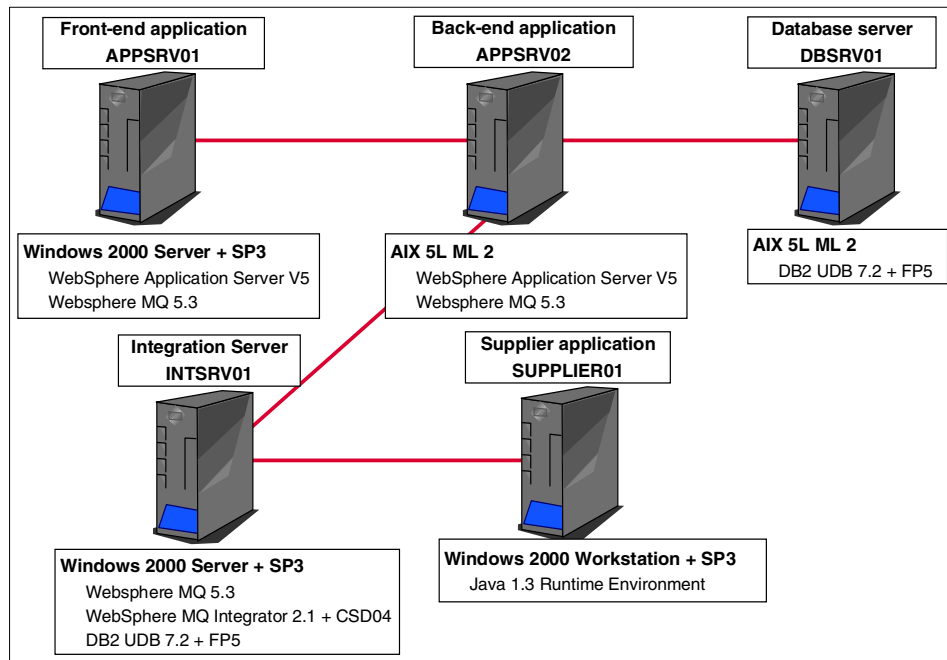


Figure 11-2 Runtime environment

The installation of the system does not require any special step. Follow the product documentation for each component during installation. The recommended installation order for each system follows.

Database server (DBSRV01)

1. AIX 5L
2. AIX 5L Maintenance Level 2
3. DB2 UDB 7.2 Workstation or Enterprise Edition with FixPak 5 or higher.

Recommended install directory: c:\sqllib

Front-end application server (APPSRV01)

1. Windows 2000 Server
2. Windows 2000 Service Pack 3 and other critical updates

3. WebSphere MQ 5.3 with Java libraries
Recommended install directory: c:\ibm\webspheremq
4. WebSphere Application Server V5
Do not install the application server samples and the embedded messaging components for the server.
Recommended install directory on Windows: c:\websphere\appserver

Back-end application server (APPSRV02)

1. AIX 5L
2. AIX 5L Maintenance Level 2
3. WebSphere MQ 5.3 with Java libraries
Recommended install directory: c:\ibm\webspheremq
4. WebSphere Application Server V5
Do not install the application server samples and the embedded messaging components for the server.
Recommended install directory on Windows: c:\websphere\appserver

Integration server (INTSRV01)

1. Windows 2000 Server
2. Windows 2000 Service Pack 3 and other critical updates
3. DB2 UDB 7.2 Workstation or Enterprise Edition with FixPak 5 or higher.
Recommended install directory: c:\sqllib
4. WebSphere MQ 5.3 with Java libraries
Recommended install directory: c:\ibm\webspheremq
5. WebSphere MQ Integrator V2.1 Broker with the components: Broker, Configuration Manager, Control Center.
Recommended install directory: c:\ibm\wmqi
WebSphere MQ Integrator V2.1 Broker CSD04

Supplier application (SUPPLIER01)

1. Windows 2000 Workstation
2. Windows 2000 Service Pack 3 and other critical updates
3. WebSphere MQ 5.3 with Java libraries
Recommended install directory: c:\ibm\webspheremq

4. IBM Java V1.3 Runtime environment.

Optional - Network Deployment Manager (DMGR01)

Organizing the servers into a cell is optional. In this case the two application servers, the front-end server and back-end server, are attached to the same cell, managed by a Network Deployment Manager.



Configuring WebSphere

This chapter describes the necessary steps to configure WebSphere Application Server V5 to use a messaging provider.

The step-by-step instructions apply to the following scenarios:

- ▶ Using WebSphere Application server with the embedded JMS provider
- ▶ Using WebSphere Application server with an external WebSphere MQ messaging provider

12.1 Defining JMS resources to WebSphere

As required by the J2EE 1.3 standard, WebSphere Application Server 5.0 provides an embedded JMS provider.

This embedded JMS provider is fully configured using the WebSphere Application Server Administrative Console. In this section, we show you how to define queue connection factories and queues using the Administrative Console.

Three steps are required to define a queue in the embedded JMS provider:

1. Define a queue connection factory.
2. Define a queue destination.
3. Define the queue in the JMS server.
4. Define a listener port for queues that are reusing messages on behalf of the application server.

12.1.1 Determining the correct scope

In WebSphere the resources, including the connection factories and destinations, have a scope where they are defined. The scope determines the visibility of the particular resource for other server processes. The following scopes are available:

► Server

Server represents the application server in WebSphere. When a resource is on a server scope, it is only visible for the server it was defined for.

► Node

Node represents the node where the application server(s) are running. One node, eventually one physical machine, can run multiple application servers. When a resource is defined on the node scope level, the resource is visible for each application server on that given node.

► Cell

Cell represents a collection of application servers, that can be from multiple nodes. When a resource is defined at the cell level, the resource is visible for all the application servers in that given cell.

When defining a resource, the first step is to point out the scope for the resource. For example, the WebSphere Administrative Console provides a list of the three scopes as an option. When you have multiple servers or nodes in your runtime, you will have to select the actual server or node where you want the resource defined.

12.2 Using the embedded JMS server

This section provides details about configuring WebSphere Application Server to use WebSphere MQ.

12.2.1 Defining a queue connection factory

The following steps show how to define a queue connection factory in WebSphere.

- 1. Start the WebSphere administrative console and log in.
- 2. To define a queue connection factory, select **Resources -> WebSphere JMS Provider**.

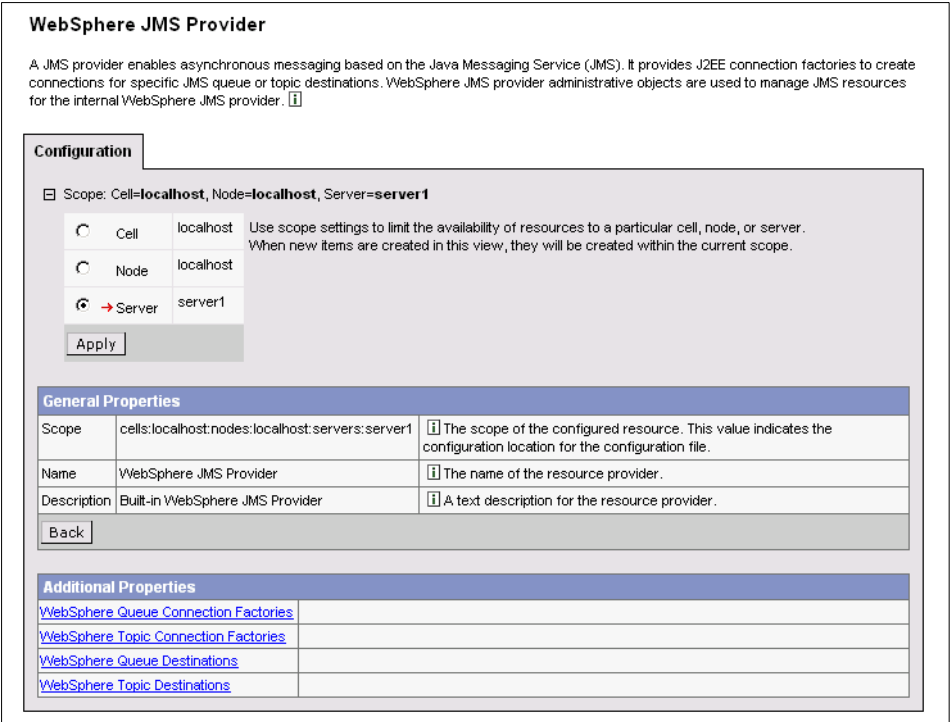


Figure 12-1 WebSphere JMS Provider configuration window

- 3. In the WebSphere JMS Provider configuration window, select the scope where you want to create the new queue connection factory and queue and click **Apply**.
- 4. Then click **WebSphere Queue Connection Factories**. This takes you to the queue connection factories window, shown in Figure 12-2 on page 238.

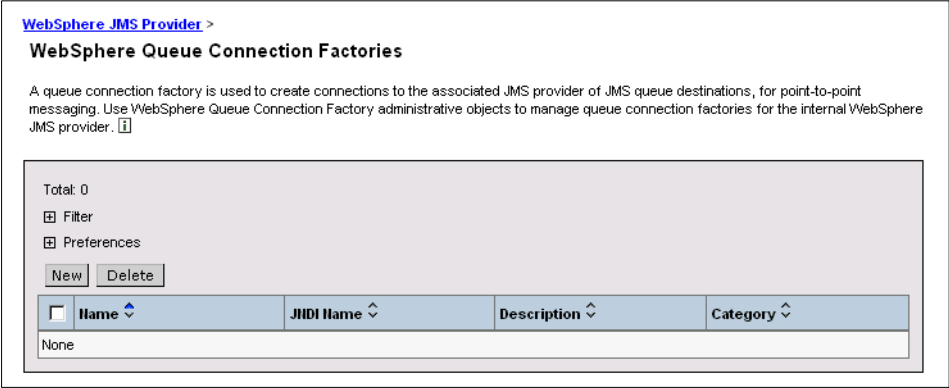


Figure 12-2 WebSphere Queue Connection Factories window

5. In this window, you can create, modify, or delete queue connection factories defined for the WebSphere JMS provider. To create a new queue connection factory click **New**, this will take you to the queue connection factory configuration window, as shown in Figure 12-3 on page 239.

[WebSphere JMS Provider](#) > [WebSphere Queue Connection Factories](#) >

New

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere Queue Connection Factory administrative objects to manage queue connection factories for the internal WebSphere JMS provider. [i]

Configuration

General Properties		
Scope	* cells:localhost:nodes:localhost:servers:server1	[i] The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* My Queue Connection Factory	[i] The required display name for the resource.
JNDI Name	* jms/MyQCF	[i] The JNDI name for the resource.
Description	<div></div>	[i] An optional description for the resource.
Category		[i] An optional category string which can be used to classify or group the resource.
Node	localhost	[i] The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server.
Component-managed Authentication Alias	localhost/mq_user	[i] References authentication data for component-managed signon to the resource.
Container-managed Authentication Alias	localhost/mq_user	[i] References authentication data for container-managed signon to the resource.
XA Enabled	<input checked="" type="checkbox"/> Enable XA	[i] Attribute to indicate whether or not the JMS provider is XA enabled or not. This attribute only applies to specialized models of JMSConnectionFactory. It is meaningless for GenericJMSConnectionFactories, as they define such feature enablements through name/value property pairs.

Apply OK Reset Cancel

Figure 12-3 Configuration window for a queue connection factory

6. In this window, we can specify the following settings for the queue connection factory:
- **Scope:** The scope of the queue connection factory identifies the location for the configuration file. This value is static and depends on the server selection made in the WebSphere JMS Provider window.
 - **Name:** This is the required display name for the connection factory.
 - **JNDI Name:** This is the resource name, which is used to locate the queue connection factory in the JNDI context.
 - **Description:** An optional description for the resource.

- **Category:** An optional category string that can be used to classify or group the resource.
 - **Node:** The WebSphere node name of the administrative node where the JMS server runs for this connection factory. Connections created by this factory connect to that JMS server
 - **Component-managed Authentication Alias:** References authentication data for component-managed sign-on to the resource.
 - **Container-managed Authentication Alias:** References authentication data for container-managed sign-on to the resource.
 - **XA Enable:** This attribute indicates whether the JMS provider is XA-compliant or not.
7. Click **OK** and save the configuration before continuing.

Important: In order to have changes applied to the server configuration, the changes must be saved, using the Save link that appears at the top of any window after a change has been made.

12.2.2 Defining a queue destination

The following section provides the steps for defining a new queue destination.

1. From the WebSphere JMS Provider window, click **WebSphere Queue Destinations**. You will see the following window.

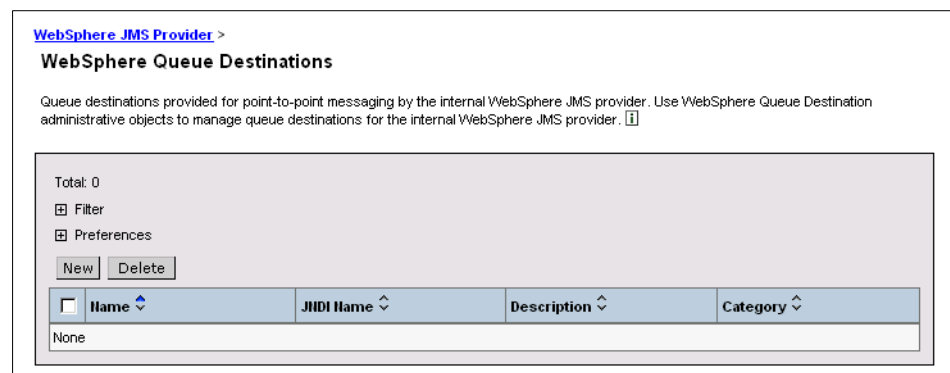


Figure 12-4 WebSphere Queue Destinations window

2. Similar to the Queue Connection Factories window, here you can create, modify or delete queue destinations defined in the WebSphere JMS provider. To create a new queue destination, click **New**. This will take you to the queue destination configuration window, shown in Figure 12-5 on page 241.

[WebSphere JMS Provider](#) > [WebSphere Queue Destinations](#) >

New

Queue destinations provided for point-to-point messaging by the internal WebSphere JMS provider. Use WebSphere Queue Destination administrative objects to manage queue destinations for the internal WebSphere JMS provider. NOTE: The queue name must also be added to the list of queue names in the configuration of the JMS server(s) where the queue is to be hosted. [i](#)

Configuration

General Properties

Scope	* cells:localhost:nodes:localhost:servers:server1	i The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* MyQueue	i The required display name for the resource.
JNDI Name	* jms/MyQ	i The JNDI name for the resource.
Description	<div></div>	i An optional description for the resource.
Category		i An optional category string which can be used to classify or group the resource.
Persistence	APPLICATION DEFINED	i Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
Priority	APPLICATION DEFINED	i Whether the message priority for this destination is defined by the application or the Specified priority property.
Specified Priority		i If the Priority property is SPECIFIED, type here the message priority for this queue, in the range 0 through 9 with 0 as the lowest priority and 9 as the highest priority..
Expiry	APPLICATION DEFINED	i Whether the expiry timeout for this queue is defined by the application or the Specified expiry property, or messages on the queue never expire (have an unlimited expiry timeout).
Specified Expiry	<div></div> milliseconds	i If the Expiry timeout property is SPECIFIED, type here the number of milliseconds after which messages on this queue expire. Valid values are any long value greater than zero.

Apply

OK

Reset

Cancel

Figure 12-5 Configuration for a queue destination

- In this window, you can set the following fields for the queue definition:
 - Scope:** The scope of the queue destination identifies the location for the configuration file. This value is static and depends on the server selection made in the WebSphere JMS Provider window.
 - Name:** This is the required display name for the queue destination. This name is used later when defining the queue on the JMS server.
 - JNDI Name:** This is the resource name, which is used to locate the queue in the JNDI context.

- **Description:** An optional description for the resource.
- **Category:** An optional category string that can be used to classify or group the resource.
- **Persistence:** This parameter lets you specify whether the messages sent to the destinations are persistent, non-persistent, or defined by the application.
- **Priority & Specified Priority:** This defines the message priority for the destination if the application is defined or specified. In the last case, the Specified Priority parameter must be a number in the range 0 through 9, where 0 is the lowest priority and 9 the highest priority.
- **Expiry & Specified Expiry:** This defines the conditions when a message sent to this destination will expire. It accepts three possible values:
 - Application defined: the JMS client will be responsible for specifying the expiry conditions for every message sent to this destination.
 - Unlimited: messages to this destination will not expire.
 - Specified: messages to this destination will expire after the amount of milliseconds specified in the Specified Expiry parameter.

12.2.3 Define the queue for the JMS server

The following steps show the queue definition for the embedded JMS server in WebSphere V5:

1. Start the WebSphere Administrative Console and log in.
2. Queues (as well as topics) are finally defined in the JMS server component. To define the queue, click **Servers -> Application Servers**.
3. Select the server you want to create the queue for, for example select **server1**.
4. Click the **Server Components** link.
5. Select the **JMS Server** component.
6. On the following window, list the queue names you want to use with the JMS server.

[Application Servers](#) > [server1](#) > [Server Component](#) >

Internal JMS Server

The JMS functions on a node within the WebSphere Application Server administration domain are served by the JMS server on that node. Use this panel to view or change the configuration properties of the selected JMS server. [i](#)

Configuration

General Properties

Name	Internal JMS Server	i The name of the server.
Description	Internal WebSphere JMS Server	i A description of the JMS server, for administrative purposes.
Number of threads	1	i The number of concurrent threads to be used by the Pub/Sub matching engine.
Queue names	MyQueue	i The names of queues hosted by this JMS server.
Initial State	Stopped	i The execution state requested when the server is first started.

Figure 12-6 Embedded JMS Server configuration window

- You might also want to change the initial state to Started in order to start the embedded JMS server the next time you start the application server.
- Click **OK**, then save the configuration.

12.3 Using WebSphere MQ V5.3

WebSphere Application V5 can also use WebSphere MQ as a JMS provider. The WebSphere administrative console gives an interface similar to the one provided for the embedded JMS provider to define WebSphere MQ resources.

Before an external WebSphere MQ JMS provider can be used, we must verify that the MQJMS_LIB_ROOT variable is set in the environment. This can be done in the WebSphere Variables configuration window.

12.3.1 Defining a queue connection factory

To configure the WebSphere MQ JMS provider, click **Resources -> WebSphere MQ JMS Provider**. This will take you to the window shown in Figure 12-7 on page 244.

WebSphere MQ JMS Provider

A JMS provider enables asynchronous messaging based on the Java Messaging Service (JMS). It provides J2EE connection factories to create connections for specific JMS queue or topic destinations. WebSphere MQ JMS provider administrative objects are used to manage JMS resources for WebSphere MQ as the JMS provider. [i](#)

Configuration

Scope: Cell=localhost, Node=localhost, Server=server1

Celllocalhost

Nodelocalhost

Serverserver1

Use scope settings to limit the availability of resources to a particular cell, node, or server. When new items are created in this view, they will be created within the current scope.

Apply

General Properties

Scope	cells:localhost:nodes:localhost:servers:server1	i The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	WebSphere MQ JMS Provider	i The name of the resource provider.
Description	WebSphere MQ JMS Provider	i A text description for the resource provider.
Classpath	\$(MQJMS_LIB_ROOT)	i A list of paths or JAR file names which together form the location for the resource provider classes. Classpath entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Classpaths may contain variable (symbolic) names which can be substituted using a variable map. Check your drivers installation notes for specific JAR file names which are required.
Native Library Path	\$(MQJMS_LIB_ROOT)	i An optional path to any native libraries (.dll's, .so's). Native path entries are separated by using the ENTER key and must not contain path separator characters (such as ';' or ':'). Native paths may contain variable (symbolic) names which can be substituted using a variable map.

Back

Additional Properties

WebSphere MQ Queue Connection Factories	
WebSphere MQ Topic Connection Factories	
WebSphere MQ Queue Destinations	
WebSphere MQ Topic Destinations	


Figure 12-7 WebSphere MQ JMS Provider configuration window

To create a queue connection factory, we can follow a procedure similar to the one used with the embedded JMS provider.

Once we get to the queue connection factory configuration window, we have to specify some additional WebSphere MQ parameters for the JMS provider, as shown in Figure 12-8 on page 245.

244 Self-Service Applications using IBM WebSphere V5.0 and IBM MQSeries Integrator

New

A queue connection factory is used to create connections to the associated JMS provider of JMS queue destinations, for point-to-point messaging. Use WebSphere MQ Queue Connection Factory administrative objects to manage queue connection factories for the WebSphere MQ JMS provider. 

Configuration


















General Properties		
Scope	* cells:localhost:nodes:localhost:servers:server1	 The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	* MyMQ Queue Connection Factory	 The required display name for the resource.
JNDI Name	* jms:MyMQGCF	 The JNDI name for the resource.
Description	<div></div>	 An optional description for the resource.
Category		 An optional category string which can be used to classify or group the resource.
Component-managed Authentication Alias	localhost/mq_user	 References authentication data for component-managed signon to the resource.
Container-managed Authentication Alias	localhost/mq_user	 References authentication data for container-managed signon to the resource.
Queue Manager	MY.QM	 The name of the WebSphere MQ queue manager for this connection factory. Connections created by this factory connect to that queue manager.
Host	localhost	 The name of the host on which the WebSphere MQ queue manager runs, for client connection only.
Port	1414	 The TCP/IP port number used for connection to the WebSphere MQ queue manager, for client connection only.
Channel		 The name of the channel used for connection to the WebSphere MQ queue manager, for client connection only.
Transport Type	BINDINGS	 Whether WebSphere MQ client TCP/IP connection or inter-process bindings connection is to be used to connect to the WebSphere MQ queue manager. Inter-process bindings may only be used to connect to a queue manager on the same physical machine.
Model Queue Definition		 The name of the model queue definition that can be used by the queue manager to create temporary queues if a queue requested does not already exist.
Client ID		 The JMS client identifier used for connections to the WebSphere MQ queue manager.
CCSID		 The coded character set identifier for use with the WebSphere MQ queue manager.
Message Retention	<input checked="" type="checkbox"/> Enable message retention	 Select this tick box to specify that unwanted messages are to be left on the queue. Otherwise, unwanted messages are dealt with according to their disposition options.
XA Enabled	<input checked="" type="checkbox"/> Enable XA	 Attribute to indicate whether or not the JMS provider is XA enabled or not. This attribute only applies to specialized models of JMSConnectionFactory. It is meaningless for GenericJMSConnectionFactory, as they define such feature enablements through name/value property pairs.

Figure 12-8 WebSphere MQ Queue Connection Factory configuration window

In the Queue Connection Factory configuration window, we specify the following parameters:

- ▶ **Scope:** The scope of the Queue Connection Factory identifies the location for the configuration file. This value is static and depends on the server selection made in the WebSphere JMS Provider window.
- ▶ **Name:** This is the required display name for the connection factory.
- ▶ **JNDI Name:** This is the resource name, which will be used to locate the queue connection factory in the JNDI context.
- ▶ **Description:** An optional description for the resource.
- ▶ **Category:** An optional category string that can be used to classify or group the resource.
- ▶ **Component-managed Authentication Alias:** References authentication data for component-managed sign-on to the resource.
- ▶ **Container-managed Authentication Alias:** References authentication data for container-managed sign-on to the resource.
- ▶ **Queue Manager:** The name of the WebSphere MQ queue manager for this connection factory. Connections created using this connection factory connect to that queue manager. The queue manager must be previously defined using the WebSphere MQ Explorer.
- ▶ **Host:** The name of the host on which the WebSphere MQ queue manager runs. This is required only for client connections as defined using the Transport Type properties.
- ▶ **Port:** The TCP/IP port number used to connect to the WebSphere MQ queue manager as defined in the queue manager creation process. This is required only for client connections.
- ▶ **Channel:** The name of the channel used to connect to the WebSphere MQ queue manager. This is required only for client connections.
- ▶ **Transport Type:** Connections to WebSphere MQ can be accomplished using either client or inter-process bindings connection. For the client connection the host, port and channel properties must be properly set. Inter-process binding connection can only be used if the WebSphere MQ resides in the same physical machine as the WebSphere Application Server.
- ▶ **Model Queue Definition:** The name of the model queue used by the queue manager to create temporary queues.
- ▶ **Client ID:** The JMS client used for connections to the WebSphere MQ queue manager.
- ▶ **CCSID:** The coded character set identifier for use with the WebSphere MQ queue manager.

- ▶ **Message Retention:** This property makes unwanted messages stay in the queue. Otherwise, the messages will be dealt with according to their disposition options.
- ▶ **XA Enable:** This attribute indicates whether the JMS provider is XA-compliant or not.

Additionally, we can configure the connection and session pools for the WebSphere Application Server.

Once configuration is done for the Connection Factory properties, click **Apply** to make the changes take effect and then save the configuration.

12.3.2 Define a queue destination

To create queue destinations for WebSphere MQ, we can use the WebSphere MQ Queue Destinations link from the WebSphere MQ JMS Provider window. In the Queue Destinations list window, you can create, modify, or delete queue destination configurations. To add a new queue destination, click **New**, and it will take you to the following window.

New

Queue destinations provided for point-to-point messaging by the WebSphere MQ JMS provider. Use WebSphere MQ Queue Destination administrative objects to manage queue destinations for the WebSphere MQ JMS provider. [i](#)

Configuration		
General Properties		
Scope	cells:localhost:nodes:localhost:servers:server1	i The scope of the configured resource. This value indicates the configuration location for the configuration file.
Name	MyMQQueue	i The required display name for the resource.
JNDI Name	jms/MyMQQ	i The JNDI name for the resource.
Description	<div></div>	i An optional description for the resource.
Category		i An optional category string which can be used to classify or group the resource.
Persistence	APPLICATION DEFINED	i Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
Priority	APPLICATION DEFINED	i Whether the message priority for this destination is defined by the application or the Specified priority property.
Specified Priority		i If the Priority property is set to Specified, type here the message priority for this queue, in the range 0 through 9.
Expiry	APPLICATION DEFINED	i Whether the expiry timeout for this queue is defined by the application or the Specified expiry property, or messages on the queue never expire (have an unlimited expiry timeout).
Specified Expiry		i If the Expiry timeout property is set to Specified, type here the number of milliseconds (greater than 0) after which messages on this queue expire.
Base Queue Name	MY.QUEUE	i The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	MY.QM	i The name of the WebSphere MQ queue manager to which messages are sent.
CCSID		i The coded character set identifier for use with the WebSphere MQ queue manager.
Native Encoding	<input type="checkbox"/> Use native encoding	i When enabled, native encoding is used. When disabled, the settings for integer, decimal and floating point are used.
Integer Encoding	Normal	i If native encoding is not enabled, select whether integer encoding is normal or reversed.
Decimal Encoding	Normal	i If native encoding is not enabled, select whether decimal encoding is normal or reversed.
Floating Point Encoding	IEEENormal	i If native encoding is not enabled, select the type of floating point encoding.
Target Client	JMS	i Whether the receiving application is JMS-compliant or is a traditional WebSphere MQ application.
WebSphere MQ Queue Connection Properties		
Queue Manager Host	localhost	i The name of host for the queue manager on which the queue destination is created.
Queue Manager Port	1414	i The number of the port used by the queue manager on which this queue is defined.
Server Connection Channel Name		i The name of the channel to use to connect to the queue manager.
User ID		i The user ID used, with the Password property, for authentication when connecting to the queue manager to define the queue destination.
Password		i The password, used with the User name property, for authentication when connecting to the queue manager to define the queue destination.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 12-9 WebSphere MQ JMS Provider Queue configuration window

In the Queue Destination configuration window, we specify the following parameters for the queue:

- ▶ **Scope:** The scope of the Queue Destination identifies the location for the configuration file. This value is static and depends on the server selection made in the WebSphere MQ JMS Provider window.
- ▶ **Name:** The required display name for the queue destination. This name is used when defining the queue on the JMS server.
- ▶ **JNDI Name:** The resource name, this name is used to locate the Queue in the JNDI context.
- ▶ **Description:** An optional description for the resource.
- ▶ **Category:** An optional category string that can be used to classify or group the resource.
- ▶ **Persistence:** It specifies whether the messages sent to the destinations are persistent, non-persistent, or defined by the application.
- ▶ **Priority & Specified Priority:** It defines the message priority for the destination of the application defined or specified. The Specified Priority parameter must be a number in the range 0 through 9, where 0 is the lowest priority and 9 the highest priority.
 - **Expiry & Specified Expiry:** This defines the conditions when a message sent to this destination will expire. It accepts three possible values:
 - Application defined: the JMS client will be responsible for specifying the expiry conditions for every message sent to this destination.
 - Unlimited: messages to this destination will not expire.
 - Specified: messages to this destination will expire after the amount of milliseconds specified in the Specified Expiry parameter.
- ▶ **Base Queue Name:** The name of the queue to which messages are sent or received.
- ▶ **Base Queue Manager Name:** The name of the WebSphere MQ queue manager where the base queue resides.
- ▶ **CCSID:** The coded character set identifier for use with the WebSphere MQ queue manager.
- ▶ **Native Encoding:** If checked, the native encoding for the queue manager is used. Otherwise the Integer, Decimal and Floating Point Encoding settings will be used.
- ▶ **Integer Encoding:** It defines whether the integer encoding is Normal or Reversed.
- ▶ **Decimal Encoding:** It defines whether the integer encoding is Normal or Reversed.

- ▶ **Floating Point Encoding:** It defines the type of floating point encoding.
- ▶ **Target Client:** It defines whether the receiving application is JMS-compliant or a traditional WebSphere MQ application.

If the connection to this queue destination is created without a queue connection factory, the following settings are required:

- ▶ **Queue Manager Host:** The name of the host for the queue manager where the queue destination is created.
- ▶ **Queue Manager Port:** The number of the port used by the queue manager on which the queue is defined.
- ▶ **Server Connection Channel Name:** The name of the channel used to connect to the queue manager.
- ▶ **User ID:** The user ID used to connect to the queue manager.
- ▶ **Password:** The password for this user ID.

If the WebSphere MQ Queue is a local queue, some queue settings can be changed using the WebSphere Administration Console by clicking the **MQ Config** link at the bottom of the Queue Destination configuration window.

WebSphere MQ Queue

A description of the queue, for administrative purposes. ⓘ

Configuration		
General Properties		
Base Queue Name	MY.QUEUE	ⓘ The name of the queue to which messages are sent, on the queue manager specified by the Base queue manager name property.
Base Queue Manager Name	MY.QM	ⓘ The name of the WebSphere MQ queue manager to which messages are sent.
Queue Manager Host	localhost	ⓘ The name of host for the queue manager on which the queue destination is created.
Queue Manager Port	1414	ⓘ The number of the port used by the queue manager on which this queue is defined.
Server Connection Channel Name	SYSTEM.DEF.SVRCONN	ⓘ The name of the channel to use to connect to the queue manager.
User ID		ⓘ The user ID used, with the Password property, for authentication when connecting to the queue manager to define the queue destination.
Password	*****	ⓘ The password, used with the User name property, for authentication when connecting to the queue manager to define the queue destination.
Description	<input type="text"/>	ⓘ An optional description for the resource.
Inhibit Put	<input type="button" value="Put Allowed"/>	ⓘ Whether or not put operations are allowed for this queue.
Persistence	<input type="button" value="Not Persistent"/>	ⓘ Whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
Cluster Name	<input type="text"/>	ⓘ The name of the cluster to which the WebSphere MQ queue manager belongs.
Cluster Name List	<input type="text"/>	ⓘ The name of the cluster namelist to which the WebSphere MQ queue manager belongs.
Default Binding	<input type="button" value="On Open"/>	ⓘ The default binding to be used when the queue is defined as a cluster queue.
Inhibit Get	<input type="button" value="Get Allowed"/>	ⓘ Whether or not get operations are allowed for this queue.
Maximum Queue Depth	<input type="text" value="5000"/>	ⓘ The maximum number of messages allowed on the queue.
Maximum Message Length	<input type="text" value="4194304"/>	ⓘ The maximum length, in bytes, of messages on this queue.
Shareability	<input type="button" value="Shareable"/>	ⓘ Whether multiple applications can get messages from this queue.
Input Open Option	<input type="button" value="Shared"/>	ⓘ The default share option for applications opening this queue for input.
Message Delivery Sequence	<input type="button" value="Priority"/>	ⓘ The order in which messages are delivered from the queue in response to get requests.
Backout Threshold	<input type="text" value="0"/>	ⓘ The maximum number of times that a message can be backed out. If this threshold is reached, the message is requeued on the backout queue.
Backout Requeue Name	<input type="text"/>	ⓘ The name of the backout queue to which messages are requeued if they have been backed out more than the backout threshold.
Harden Get Backout	<input type="button" value="Hardened"/>	ⓘ Whether hardening should be used to ensure that the count of the number of times that a message has been backed out is accurate.
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>		

Figure 12-10 WebSphere MQ queue configuration window

In this window, you configure the following WebSphere MQ queue settings:

- ▶ **Base Queue Name:** The name of the WebSphere MQ queue on the queue manager, corresponding to the queue destination.
- ▶ **Base Queue Manager Name:** The name of the WebSphere MQ queue manager where the queue resides.
- ▶ **Queue Manager Host:** The name of the host for the queue manager where the destination queue is defined.
- ▶ **Queue Manager Port:** The port number for the queue manager on which the destination queue is defined.
- ▶ **Server Connection Channel Name:** The name of the channel to connect to the queue manager.
- ▶ **User ID:** The user ID to connect to the queue manager.
- ▶ **Password:** The user password to connect to the queue manager.
- ▶ **Name:** The resource queue name used for display purposes.
- ▶ **Description:** An optional description of the queue.
- ▶ **Inhibit Put:** It defines whether put operations are allowed for this queue.
- ▶ **Persistence:** It defines whether all messages sent to the destination are persistent, non-persistent, or have their persistence defined by the application.
- ▶ **Cluster Name:** The name of the cluster where the queue manager belongs.
- ▶ **Cluster Name List:** The name of the cluster name list where the queue manager belongs.
- ▶ **Default Binding:** The default binding to be used when the queue is defined as a cluster queue.
- ▶ **Inhibit Get:** It defines whether or not get operations can be performed on this queue.
- ▶ **Maximum Queue Depth:** The maximum number of messages allowed on the queue.
- ▶ **Maximum Message Length:** The maximum length, in bytes, of the messages on this queue.
- ▶ **Shareability:** It defines whether multiple applications can get messages from this queue simultaneously.
- ▶ **Input Open Option:** The default share option for applications opening this queue for input.

- ▶ **Message Delivery Sequence:** The order in which messages are delivered from the queue in response to get requests. There are two possible options: FIFO or Priority.
- ▶ **Backout Threshold:** The maximum number of times that a message can be backed out. If this threshold is reached, the message is requeued on the backout queue.
- ▶ **Backout Requeue Name:** The name of the backout queue.
- ▶ **Harden Get Backout:** It defines whether hardening should be used to ensure that the count of the number of times that a message has been backed out is accurate.

12.3.3 Define the queue for WebSphere MQ

Once the queue connection factory and queue destination are defined for the WebSphere Application Server, you will need queue manager(s) and queue(s) defined for WebSphere MQ. You will probably have queue managers and queues created earlier than resources defined for WebSphere Application Server.

You can create these objects for WebSphere MQ using the MQ Explorer application on the Windows platform, or you can use the MQSC command line application.

12.4 Deploying message-driven beans in WebSphere V5.0

Message-driven beans have to be linked to the corresponding queue or topic it will be listening to. This is accomplished in WebSphere Application Server by a ListenerPort definition. A ListenerPort contains information regarding the connection factory and queue (or topic) the MDB will be linked to. Multiple MDBs can be listening to a single listener port. There can only be one listener port by queue or topic defined for the server.

The listener port is defined using the WebSphere Application Server V5.0 Administrative Console, as follows:

1. Click **Server -> Application Servers** and select the server where the listener port is going to be defined. This will take you to the Application Server configuration window, as shown in the following figure.

Application Servers

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

Runtime

Configuration

General Properties

Name		The display name for the application server
Initial State	Started	The desired execution state.
Application ClassLoaderPolicy	MULTIPLE	Defines whether there is a single classloader (SINGLE) for all applications or a classloader per application (MULTIPLE).
Application ClassLoadingMode	PARENT_FIRST	Defines the classloading mode when the applicationClassLoaderPolicy is SINGLE only.

Apply

OK

Reset

Cancel

Additional Properties

Transaction Service	Specify settings for the Transaction Service, as well as manage active transaction locks.
Web Container	Specify thread pool and dynamic cache settings for the container . Also, specify session manager settings such as persistence and tuning parameters, and HTTP transport settings.
EJB Container	Specify cache and datasource information for the container.
Dynamic Cache Service	Specify settings for the Dynamic Cache service of this server.
Logging and Tracing	Specify Logging and Trace settings for this server.
Message Listener Service	Configuration for the Message Listener Service.This service provides the Message Driven Bean (MDB) listening process, whereby MDBs are deployed against ListenerPorts that define the JMS destination to listen upon. These Listener Ports are defined within this service along with settings for its Thread Pool.
ORB Service	Specify settings for the Object Request Broker Service.
Custom Properties	Additional custom properties for this runtime component. Some components may make use of custom configuration properties which can be defined here.
Administration Services	Specify various settings for administration facility for this server, such as administrative communication protocol settings and timeouts.
Diagnostic Trace Service	Diagnostic Trace Service description
Debugging Service	Specify settings for the debugging service, to be used in conjunction with a workspace debugging client application.
IBM Service Logs	Configure settings for service logs.
Custom Services	Define custom service classes that will run within this server and their configuration properties.
Server Components	Additional runtime components which are configurable.
Process Definition	A process definition defines the command line information necessary to start/initialize a process.
Performance Monitoring Service	Specify settings for Performance Monitoring
End Points	Configure important TCP/IP ports which this server uses for connections.
ClassLoader	ClassLoader configuration

Related Items

Installed Applications	Manage applications installed into this server.
--	---

Figure 12-11 Application Server Configuration window

- From this window, click the **Message Listener Service** link. This will take you to the following window.

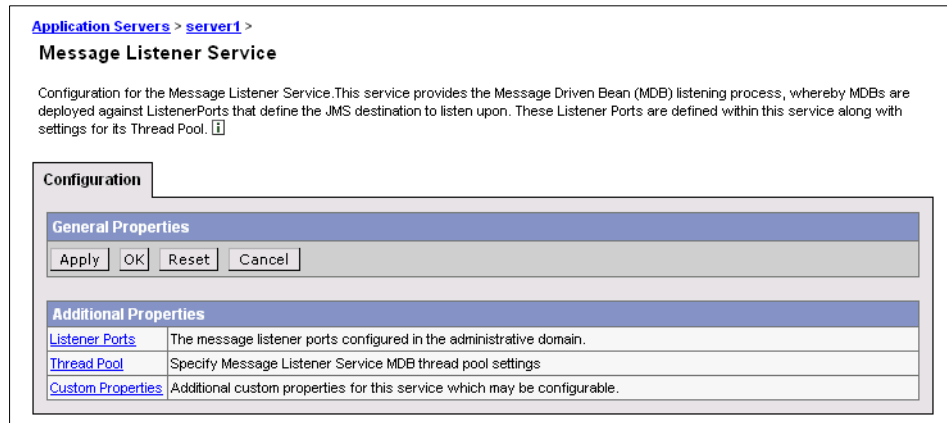


Figure 12-12 Message Listener Service Configuration window

- On this window you have access to the Listener Ports definition window, the thread pool settings for the service, and additional custom properties. Click the **Listener Ports** link, which will take you to the Listeners Ports list window, as shown in the following figure.



Figure 12-13 Listener Ports list window

In this window we create, modify and delete listener ports. Additionally, we can start or stop a specific listener port.

- To create a new listener port, click **New**. This will take you to the Listener Port configuration window shown in Figure 12-14 on page 256.

[Application Servers](#) > [server1](#) > [Message Listener Service](#) >

New

Listener ports for Message Driven Beans to listen upon for messages. Each port specifies the JMS Connection Factory and JMS Destination that an MDB, deployed against that port, will listen upon. [i](#)

Runtime		Configuration	
General Properties			
Name	MyListenerPort	i Name of the listener port	
Initial State	Started	i The execution state requested when the server is first started.	
Description		i A description of the listener port, for administrative purposes	
Connection factory JNDI name	jms/MyQCF	i The JNDI name for the JMS connection factory to be used by the listener port, for example, jms/connFactory1.	
Destination JNDI name	jms/MyQ	i The JNDI name for the destination to be used by the listener port, for example, jms/destn1.	
Maximum sessions	1	i The maximum number of concurrent JMS server sessions used by a listener to process messages, in the range 1 through 2147483647.	
Maximum retries	0	i The maximum number of times that the listener tries to deliver a message before the listener is stopped, in the range 0 through 2147483647.	
Maximum messages	1	i The maximum number of messages that the listener can process in one JMS server session, in the range 0 through 2147483647.	
<input type="button" value="Apply"/> <input type="button" value="OK"/> <input type="button" value="Reset"/> <input type="button" value="Cancel"/>			

Figure 12-14 Listener Port configuration window

Configure the listener port parameters:

- ▶ **Name:** The name for the listener port. This name will be use in the deployment of a enterprise application to link an MDB with its corresponding listener port.
- ▶ **Initial State:** Started or stopped.
- ▶ **Description:** Optional description for the item.
- ▶ **Connection factory JNDI name:** JNDI resource name for the connection factory.
- ▶ **Destination JNDI name:** JNDI resource name for the destination.
- ▶ **Maximum sessions:** The maximum number of concurrent sessions.
- ▶ **Maximum retries:** The maximum number of retries to deliver a message to an MDB before the listener stops.
- ▶ **Maximum messages:** Maximum number of messages processed at one time.

5. Click **OK** and save the configuration.

12.5 Testing, logging, debugging

Log files are the primary target for testing and debugging. Applications should use the standard log files within the applications to provide additional information in runtime. The runtime environment itself provides log information that can help to ensure the proper operation of the runtime environment.

The WebSphere Application Server's log files can be found in the `<WebSphere_root>/logs` directory. In this directory each managed process (application server, JMS server, Node agent) has a sub-directory where the processes put their log files.



Configuring WebSphere MQ and MQ Integrator

This chapter provides details about configuring WebSphere MQ and WebSphere MQ Integrator.

The steps provided in the chapter apply to any particular solution similar to the one described in the sample scenario.

13.1 WebSphere MQ objects

WebSphere MQ administration tasks include creating, starting, altering, viewing, stopping, and deleting WebSphere MQ objects, including:

- ▶ Messages
- ▶ WebSphere MQ queues
- ▶ WebSphere MQ queue managers and associated listener ports
- ▶ Channels
- ▶ Queue manager clusters
- ▶ Namelists

These object types are summarized in the following text.

Messages

Messages consist of a payload, which is the content to be transported, and one or more headers that contain metadata, which describes the message.

Queues

A queue is obviously a central concept in WebSphere MQ. A queue is more than a location for temporary storage or a gateway for delivery to a certain endpoint. A queue usually is also the command to process the queued contents in a particular way. From the perspective of a client application, the queue and its symbolic name represents the whole process.

WebSphere MQ has four basic types of queue definitions:

- ▶ Local queues, used to store and retrieve messages.
- ▶ Remote queues are local queues of a remote queue manager. You can only write to remote queues.
- ▶ Alias queues are definitions used to hide the actual names of local or remote queues from client applications.
- ▶ Transmission queues are specialized local queues. These transmission or xmit queues temporarily store messages put to remote queues until these messages have been delivered to their destination.

Important: There is no way to read from a remote queue.

Queue manager

The WebSphere MQ server is called the queue manager. Applications connect to a queue manager in order to make use of the messaging and queuing service, either as a producer or as a consumer of messages. WebSphere MQ can run several queue managers on a single machine. If so, the different queue

managers will be independent of each other, each of them establishing a namespace of its own.

Listener ports

The basic connectivity needed to connect to other queue managers is provided by one or more listener processes associated with the queue manager. It is possible to handle all traffic of a single queue manager through one listener port. Listeners have to be dedicatedly assigned to one particular queue managers. There is no way for a queue manager to share a listener port.

Note: Queue managers in security-aware environments should separate administrative and payload to different listeners to make special internal security checks and apply different firewall rules for each.

Connecting client applications

There are two different ways for a client application to use a queue manager: binding mode and client mode.

- ▶ Binding mode is available locally on the same physical machine. Client applications connect to WebSphere MQ by using original code libraries of WebSphere MQ and thus sharing physical data store with the WebSphere MQ Server. Binding mode provides full functionality to the application, including distributed transactions.
- ▶ Client mode allows access to WebSphere MQ queue managers on local as well as on remote machines. The client application establishes a synchronous connection with the queue manager via TCP/IP. The client application can manipulate most of the queue manager resources in the same way as in binding mode while connected, but there is no message store at the client side of the remote connection. Furthermore, applications connected in client mode cannot use XA-enabled distributed transactions.

A major design consideration depends on whether there is a need to queue messages for deferred delivery when there is temporarily no network connection to a particular destination system. This is not possible in client mode. If deferred delivery is an issue or if there is an unreliable network connection, for example via dialed line, the use of local queue managers should be considered.

Restriction: Applications that need syncpoint coordination with resource managers other than WebSphere MQ (distributed transactions) cannot connect to WebSphere MQ in client mode.

MQ channels

Queue managers communicate through channels. In general, you will need to define a pair of channels in a connection between two queue managers. Each definition has to be made on both endpoints of the connection to become effective.

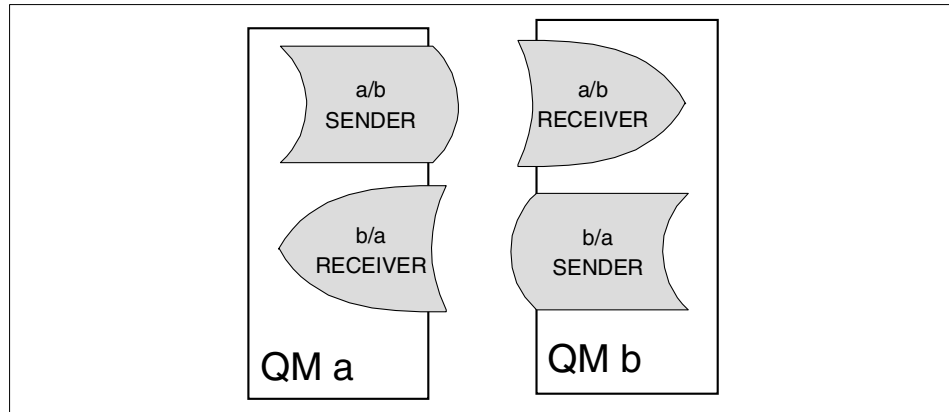


Figure 13-1 Queue managers connected through a channel pair

Queue manager clusters

Queue manager clusters are useful for reducing administration effort in medium sized or large networks of queue managers, in particular when there is a dynamic landscape.

- ▶ Writing to queues in the cluster is possible without defining remote queues, no matter which cluster queue manager hosts the queue.
- ▶ Inter-queue manager connections are established automatically by the cluster software after minimal configuration.
- ▶ Cluster software manages workload between instances of a queue hosted on several systems.

A queue manager cluster does not change the application programming interface of WebSphere MQ. Please consult the WebSphere MQ manual to learn more about queue manager clusters. Also, refer to Chapter 10, “Performance and availability” on page 211.

13.2 WebSphere MQ system management

The WebSphere MQ objects are managed by MQ administration tasks that can be performed by using any of the following MQ interfaces:

- ▶ Control commands
- ▶ WebSphere MQ commands (MQSC)
- ▶ Programmable Command Format (PCF)
- ▶ WebSphere MQ Administration Interface (MQAI)
- ▶ WebSphere MQ Explorer (available on Windows only)
- ▶ WebSphere MQ Services Console (available on Windows only)

Control commands

You use control commands to perform operations on queue managers, command servers, and channels. Control commands can be divided into three categories as shown in Table 13-1.

Table 13-1 Control command categories

Category	Description
Queue manager commands	-Creating, starting, stopping, and deleting queue managers and command servers
Channel commands	-Starting and ending channels and channel initiators
Utility commands	<ul style="list-style-type: none">- Running MQSC commands- Conversion exits- Authority management- Recording and recovering media images of queue manager resources- Displaying and resolving transactions- Trigger monitors- Displaying the file names of WebSphere MQ objects

MQSC commands

MQSC commands are used to perform operations on queue manager objects. They are issued using the **runmqsc** command. This can be done interactively from a keyboard, or by redirecting the standard input device (stdin) to run a sequence of commands from an ASCII text file. In both cases, the format of the commands are the same.

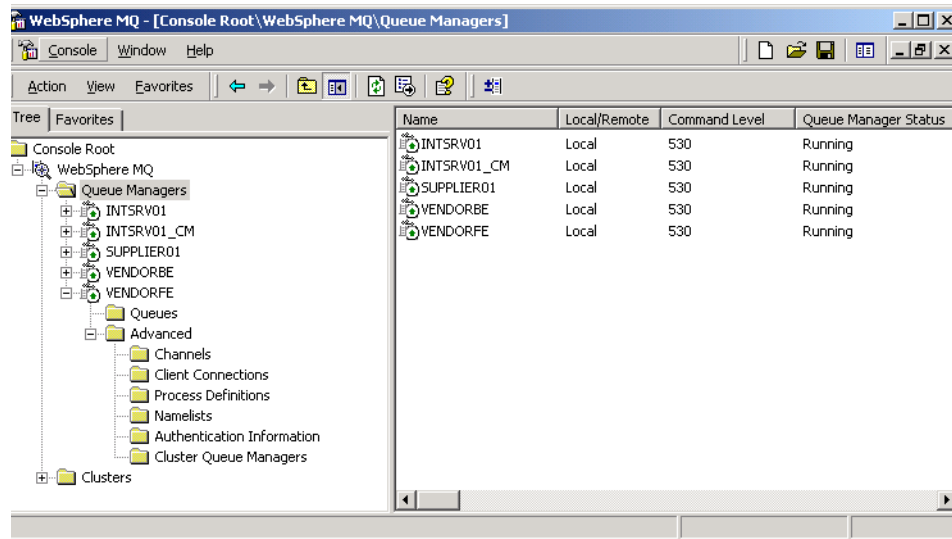
WebSphere MQ Explorer

WebSphere MQ for Windows provides an administration interface called the WebSphere MQ Explorer to perform administration tasks as an alternative to using control or MQSC commands.

The WebSphere MQ Explorer allows you to perform remote administration of your network from a computer running Windows simply by pointing the WebSphere MQ Explorer at the queue managers and clusters you are interested in.

The platforms and levels of WebSphere MQ that can be administered using the WebSphere MQ Explorer, and the configuration steps you must perform on remote queue managers to allow the WebSphere MQ Explorer to administer them, are outlined in 13.2.1, “Remote administration” on page 266.

Figure 13-2 WebSphere MQ Explorer



With the WebSphere MQ Explorer, you can:

- ▶ Start and stop queue managers.
- ▶ Define, display, and alter the definitions of WebSphere MQ objects such as queues and channels.
- ▶ Browse the messages on a queue.
- ▶ Start and stop a channel.
- ▶ View status information about a channel.
- ▶ View queue managers in a cluster.
- ▶ Create a new queue manager cluster using the Create New Cluster wizard.
- ▶ Add a queue manager to a cluster using the Add Queue Manager to Cluster wizard.
- ▶ Add an existing queue manager to a cluster using the Join Cluster wizard.

WebSphere MQ Services Console

The WebSphere MQ Services Console can be used to administer local or remote WebSphere MQ for Windows servers. It also allows you to monitor alerts created by problems in the local system.

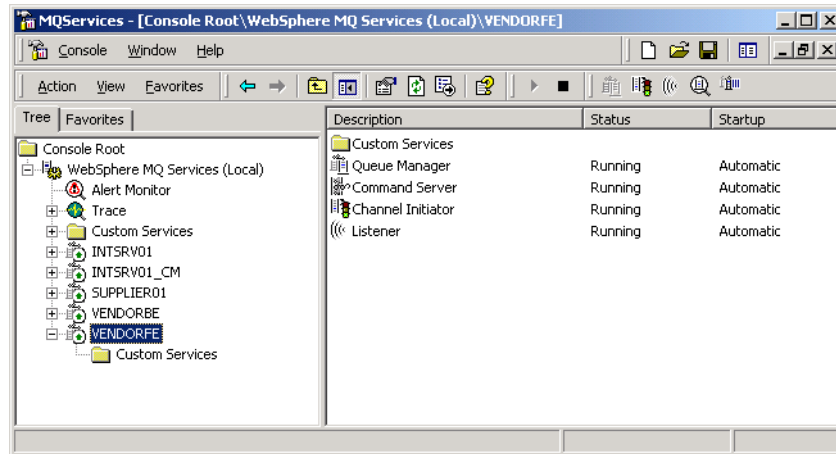


Figure 13-3 WebSphere MQ Services Console

With the WebSphere MQ Services Console, you can:

- ▶ Start or stop a queue manager.
- ▶ Start or stop the command servers, channel initiators, trigger monitors, and listeners.
- ▶ Create and delete queue managers, command servers, channel initiators, trigger monitors, and listeners.
- ▶ Set any of the services to start up automatically or manually during system start up.
- ▶ Modify the properties of queue managers. This function replaces the use of stanzas in configuration (mqs.ini and qm.ini) files.
- ▶ Change the default queue manager.
- ▶ Modify the parameters for any service, such as the TCP port number for a listener, or a channel initiator queue name.
- ▶ Modify the behavior of WebSphere MQ if a particular service fails, for example, retry starting the service x number of times.
- ▶ Start or stop the service trace.

13.2.1 Remote administration

WebSphere MQ Explorer and the WebSphere MQ Services Console all offer some form of remote administration. With the create queue manager wizard of the WebSphere MQ Explorer you can select to create a listener and a suitable server connection channel at the same time, to enable remote administration. If you wish to enable a pre existing queue manager please follow the guidelines below.

The WebSphere MQ Explorer can remotely administer WebSphere MQ on the following platforms.

Table 13-2 Remote management

Platform	Minimum Command Level
AIX and UNIX variants	221
Windows systems	201

The platform and command level headings of Table 13-2 refer to the platform and command level queue manager attributes. Both must be used to determine which system control commands are supported. You can see the platform and command level attributes in the WebSphere MQ Explorer window, shown in Figure 13-2 on page 264.

To remotely administer a queue manager from any of the WebSphere MQ administration interfaces, proceed as:

1. A command server running for any queue manager being administered. Command server is one of the services administrated through the WebSphere MQ Services Console. From UNIX or Windows command lines, you can start the command server issuing the following command:
2. A suitable TCP/IP listener for every remote queue manager. This may be the WebSphere MQ listener or the INETD daemon as appropriate.
3. A server connection channel, called SYSTEM.ADMIN.SVRCONN, on every remote queue manager. This channel is mandatory for every remote queue manager being administered.
4. Authentication and sufficient authorization. In a system that is uncritical in terms of security, you might wish to allow remote administration to any remote users by assigning an authorized user as MCAUSER of the channel. In MQI (runmqsc) commands, be sure to mask the user name with single quotes.

```
alter channel (SYSTEM.ADMIN.SVRCONN) CHLTYPE(SVRCONN)
MCAUSER('Administrator')
```

13.3 Creating the WebSphere MQ Integrator databases

The WebSphere MQ Integrator Configuration Manager and broker services use databases for administration. In our scenario we chose to create a distinct DB2 database for each component, message repository, configuration repository, and broker persistent store.

These databases can be local to the WebSphere MQ Integrator component or remote. We chose to put all of these databases on the integration server (broker) node. There are performance implications to consider when deciding where to locate the WebSphere MQ Integrator databases. In our test lab, performance was not an issue, but in a real network, performance is one of the most important issues.

Creating and binding databases

After creating the databases, you must bind each to the CLI package. On both the Windows and the AIX systems we used the following DB2 commands to create the databases and bind them.

Example 13-1 Create broker databases on Windows

```
create db mqsicmdb
connect to mqsicmdb
bind c:\sqllib\bnd\@db2cli.lst blocking all grant public
connect reset

create db mqsimrdb
connect to mqsimrdb
bind c:\sqllib\bnd\@db2cli.lst blocking all grant public
connect reset

create db mqsibkdb
connect to mqsibkdb
bind c:\sqllib\bnd\@db2cli.lst blocking all grant public
connect reset
```

Note: DB2 on AIX does not have a native DB2 Control Center, but you can easily add remote systems, including AIX DB2 systems, to a Windows DB2 Control Center. We did this and used the Windows DB2 Control Center to give access to MQSIBKDB to root and mqm. This was also very convenient during testing. We could check whether our message flows were working by inspecting the contents of the broker application databases.

Granting authorities to the databases

The following privileges need to be granted for the user ID that the WebSphere MQ Integrator service will run under to each database:

- ▶ connect
- ▶ createtab
- ▶ bindadd
- ▶ create_not_fenced

In our test lab, we used the same user ID (MQSIUSER) to create the databases as we will use to run the WebSphere MQ Integrator service, so this is done automatically.

If the WebSphere MQ Integrator service will be running on a different user ID from the one used to create the database, you can use the DB2 command in Example 13-2 to grant the proper access to the WebSphere MQ Integrator service ID.

Example 13-2 Grant database access

```
connect to database
grant connect, createtab, bindadd, create_not_fenced on database to user MQSIUSER
connect reset
```

Registering the databases to ODBC

WebSphere MQ Integrator accesses these databases using ODBC, so they must be registered as ODBC resources.

In Windows, this can be done using the DB2 Client Configuration Assistant. When you open the tool, select the database and click **Properties**. Select **Register this database for ODBC** and **As a system datasource**.

In AIX, you need to modify the .odbc.ini file. To do this, find the .odbc.ini file for the user, mqsi in this case. For us this was in /var/mqsi/odbc/.odbc.ini.

You will need to add two entries for each database. For example, the following two entries were added for the WMQIBK2 database:

```
[ODBC Data Sources]
MQSIBKDB=IBM DB2 ODBC Driver

[MQSIBKDB]
Driver=/home/db2inst1/sqllib/lib/db2.o
Description=MQSIBKDB DB2 ODBC Database
Database=MQSIBKDB
```

13.4 Creating the WebSphere MQ Integrator Configuration Manager

There is one Configuration Manager per broker domain and it must run on Windows NT or Windows 2000. It maintains configuration information in the configuration repository, manages the initialization and deployment of brokers and message processing operations in response to actions initiated through the Control Center, and checks the authority of defined user IDs to initiate those actions.

We have already defined the WebSphere MQ Integrator Configuration Manager databases and the WebSphere MQ queue manager so the Configuration Manager can be created.

First, make sure that the new Configuration Manager will be able to access its databases. If the databases are on a remote system, they need to be defined to the system that will host the WebSphere MQ Integrator Configuration Manager. In our case, the database and the Configuration Manager reside on the same system so nothing needs to be done. If this is not the case, the DB2 Client Configuration Assistant or DB2 catalog commands can be used to make this connection.

Now that we have all the underlying components installed and configured, we can create an WebSphere MQ Integrator Configuration Manager. The Configuration Manager is created using the `mqsicreateconfigmgr` command. You can enter this from a DOS prompt, or you can use the WebSphere MQ Integrator Command Assistant to build and execute the command. We elected to have the Command Assistant generate the Configuration Manager:

1. Select **Start -> Programs -> WebSphere MQ Integrator Version 2.1 -> Command Assistant -> Create Configuration Manager**.

WebSphere MQ Integrator - Create Configuration Manager

File View

mqsicreateconfigmgr

* Service User ID (-i)	mqsiuser
* Service Password (-a)	*****
* Queue Manager Name (-q)	SOLARSYS.QM.CM
User Name Server QMgr Name (-s)	
Security Domain (-d)	
NT Domain Aware Level (-l)	0
Workpath (-w)	

*-required parameter

mqsicreateconfigmgr -i mqsiuser -a ***** -q SOLARSYS.QM.CM -l 0

<<Back Next>> Page 1 of 3 Cancel Help

Figure 13-4 Create Configuration Manager window (1)

The darker boxes indicate the required parameters. As you enter the parameters, you can see the command that will execute being built at the bottom of the window.

For our scenario, we will use the queue manager created in the previous step, SOLARSYS.QM.CM.

The Configuration Manager runs as a Windows service. The user ID specified here needs to be a member of the mqbrkrs group.

Click **Next** to continue.

2. The next window configures the database names and the user ID/password used to access them.

Figure 13-5 Create Configuration Manager window (2)

Click **Next** to continue.

3. The final window shows the **mqsicreateconfigmgr** command that will execute. Click **Finish** to run the command and create the Configuration Manager.

Now we can start the Configuration Manager, from either a DOS prompt or from the Windows services window. The service will be set to start automatically, so this is the only time you will need to start it manually.

From a DOS prompt enter:

```
mqsisstart configmgr
```

Note: If there are errors during the creation or startup of the Configuration Manager, you will find detailed information about the error in the Windows Event Viewer under the application log.

13.4.1 Creating the brokers

The next step is to build one or more WebSphere MQ Integrator brokers to execute the message flows. Your applications communicate with the broker to take advantage of the services provided by these message flows.

You can install, create, and start any number of brokers within a broker domain. In our scenario, we are installing and configuring a single broker on the integration server node.

Before getting started, make sure you have defined the broker database, run the bind, and registered it to ODBC. Also make sure you have defined the queue manager for the broker.

Creating brokers on Windows

Creating the WebSphere MQ Integrator broker is similar to creating the Configuration Manager. The command used is the **mqsicreatebroker** command and can be entered from a DOS prompt, or can be created with the WebSphere MQ Integrator Command Assistant. We chose to use the Command Assistant.

1. Select **Start -> Programs -> IBM MQSeries® Integrator 2.0 -> Command Assistant -> Create Broker**.

The screenshot shows the 'WebSphere MQ Integrator - Create Broker' window. It has a menu bar with 'File' and 'View'. Below the menu bar is a tab labeled 'mqsicreatebroker'. The main area contains a table of fields for creating a broker. The first four fields are highlighted in dark blue, indicating they are required: '* Broker Name' (value: BROKER1), '* Service User ID (-i)' (value: mqsiuser), '* Service Password (-a)' (value: *****), and '* Queue Manager Name (-q)' (value: SOLARSYS.QM.BROKER1). The remaining three fields are light gray: 'User Name Server QMgr Name (-s)', 'Workpath (-w)', and 'LIL Path (-l)'. Below the table is a legend: a dark blue square followed by '*-required parameter'. At the bottom of the window, a command line shows the command: 'mqsicreatebroker BROKER1 -i mqsiuser -a ***** -q SOLARSYS.QM.BROKER1'. The bottom bar contains buttons for '<<Back', 'Next>>', 'Page 1 of 3', 'Cancel', and 'Help'.

Field	Value
* Broker Name	BROKER1
* Service User ID (-i)	mqsiuser
* Service Password (-a)	*****
* Queue Manager Name (-q)	SOLARSYS.QM.BROKER1
User Name Server QMgr Name (-s)	
Workpath (-w)	
LIL Path (-l)	

*-required parameter

mqsicreatebroker BROKER1 -i mqsiuser -a ***** -q SOLARSYS.QM.BROKER1

<<Back Next>> Page 1 of 3 Cancel Help

Figure 13-6 Creating a broker - window (1)

The darker boxes in Figure 13-6 indicate the required fields. Each broker has a unique name assigned. We will call this one BROKER1. The user ID and password to be used to run the service are required, as is the WebSphere MQ queue manager name to be used. We will use SOLARSYS.QM.BROKER1.

Click **Next** to continue.

2. Enter the name of the broker database created earlier.

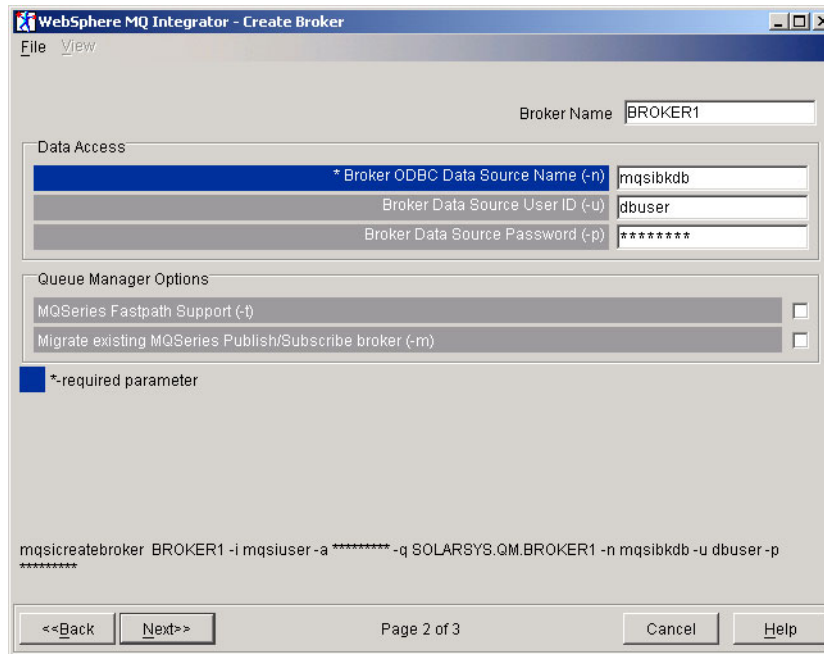


Figure 13-7 Creating a broker - window (2)

Click **Next** to continue.

3. The last window shows the **mqsicreatebroker** command that will execute. Click **Finish** to create the broker.

Common error: If you receive ODBC return code -1, make sure the bind was done for the broker database.

4. Start the broker service from the Windows Services window. The service will start automatically at reboot, so this is the last time you need to start it manually.

Creating brokers on AIX

In AIX, you do not have the option of using the WebSphere MQ Integrator Command Assistant. Also, there is some initial setup for WebSphere MQ Integrator that you will need to perform before using the broker.

1. Configure syslog for user messages by doing the following:

```
cd /var
mkdir log
touch /var/log/syslog.user
```

```
chown root:mqbrkrs /var/log/syslog.user
chmod 750 /var/log/syslog.user
```

Add the following line to /etc/syslog.conf:

```
user.debug /var/log/syslog.user
```

Note: Values can be error, information, debug, warning.

2. Create a .profile file for the mqm user. Take the sample from /usr/opt/mqsi/sample/profiles/profile.aix and copy it to /home/mqm.profile. Add an export LIBPATH statement directly under the line that sets the LIBPATH variable. Uncomment the line that executes the DB2 .profile.

Example 13-3 .profile

```
# The following will set up the NLS environment
NLSPATH=/usr/lib/nls/msg/%L/%N:/usr/lib/nls/msg/en_US/%N:$NLSPATH
export NLSPATH

# Set ODBCINI to pick up the ODBC ini file
ODBCINI=/var/mqsi/odbc/.odbc.ini
export ODBCINI

CLASSPATH=/usr/jdk_base/lib/classes.zip:$CLASSPATH
export CLASSPATH
LIBPATH=$LIBPATH:/usr/opt/mqsi/merant/lib
export LIBPATH

# If DB2 is used, the 'db2profile' script should also be run
. ~/db2inst1/sqllib/db2profile

#
MQSI_REGISTRY=/var/mqsi
export MQSI_REGISTRY
export LIBPATH
export PATH

set -o vi

PS1='$PWD> '
export PS1 #shows the directory you are in
```

3. Create the broker:

```
mqsicreatebroker BROKER2 -i mqm -a mqm -q SOLARSYS.QM.BROKER2 -n MQSIBK2
```
4. Install the Aggregator node. Follow the installation notes in the IA72 SupportPac.

5. Create user mqsiuser, and add this user to the mqbrkrs and mqm groups. mqsiuser is the user ID that we used to log on to the Configuration Manager machine to deploy message flows.

6. Start the broker:

```
mqsisstart BROKER2
```

Check for messages in the /var/log/syslog.user file. This is where we directed WebSphere MQ Integrator-specific error messages to go in AIX from a previous step.

13.4.2 Transaction behavior

A tag in the deployment properties of the message flow controls whether it is processed as a global transaction, coordinated by WebSphere MQ. Such a message flow is said to be fully globally coordinated. The default value is No.

Use coordinated transactions only where you need the message and any database updates performed by the message flow to be processed in a single unit of work, using a two-phase commit protocol. This means that both the message is read and the database updates are performed, or neither is done.

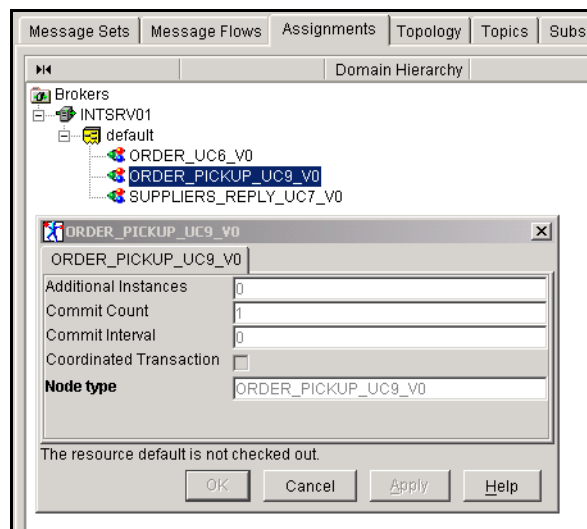


Figure 13-8 Toggle coordinated transaction

If you change this value, you must ensure that the broker's queue manager is configured to manage distributed transactions with your database. If you do not set up the queue manager correctly, a message is generated by the broker when a message is received by the message flow to indicate that although the

message flow is to be globally coordinated, the queue manager configuration does not support this.

See the *WebSphere MQ Integrator Administration Guide* for information about which databases are supported as participants in a global transaction, and the *WebSphere MQ System Administration Guide* for how to configure WebSphere MQ and the database managers.

There are several tasks that you must perform before a database manager can participate in global units of works coordinated by the queue manager:

Create an XA switch load file for the database manager. An XA switch load file is a dynamically loaded object that enables the queue manager and the database manager to communicate with each other. Define the database manager in the queue manager's configuration information. This information includes the name of the switch load file. WebSphere MQ comes with a sample makefile, used to build switch load files for the supported database managers. This makefile, together with all the associated files required to build the switch load files, is installed in the following directories:

- ▶ For WebSphere MQ for Windows, in the
 <WebSphereMQ_root>\tools\c\samples\xatm\ directory.
- ▶ For WebSphere MQ for UNIX systems, in the /opt/mqm/samp/xatm/ directory
 (/usr/mqm/samp/xatm on AIX).

Refer to your WebSphere MQ installation documentation for more information about the installation procedure.

Testing message flows during development

The WebSphere MQ IH03 SupportPac, the RFHUtil application, is a utility to monitor WebSphere MQ queues. It has every feature for managing the queues:

- ▶ Creating messages and putting them into a queue. It supports multiple types of messages, with all the parameters.
- ▶ Reading messages from queues.
- ▶ Browsing messages in a queue.

This utility can be used for testing purposes to put test messages into a queue. A huge advantage of the utility is that every possible field of a message is configurable with the tool. It is also a big help to debug a messaging application by using the browse function.

Figure 13-9 on page 278 shows the RFHUtil application.

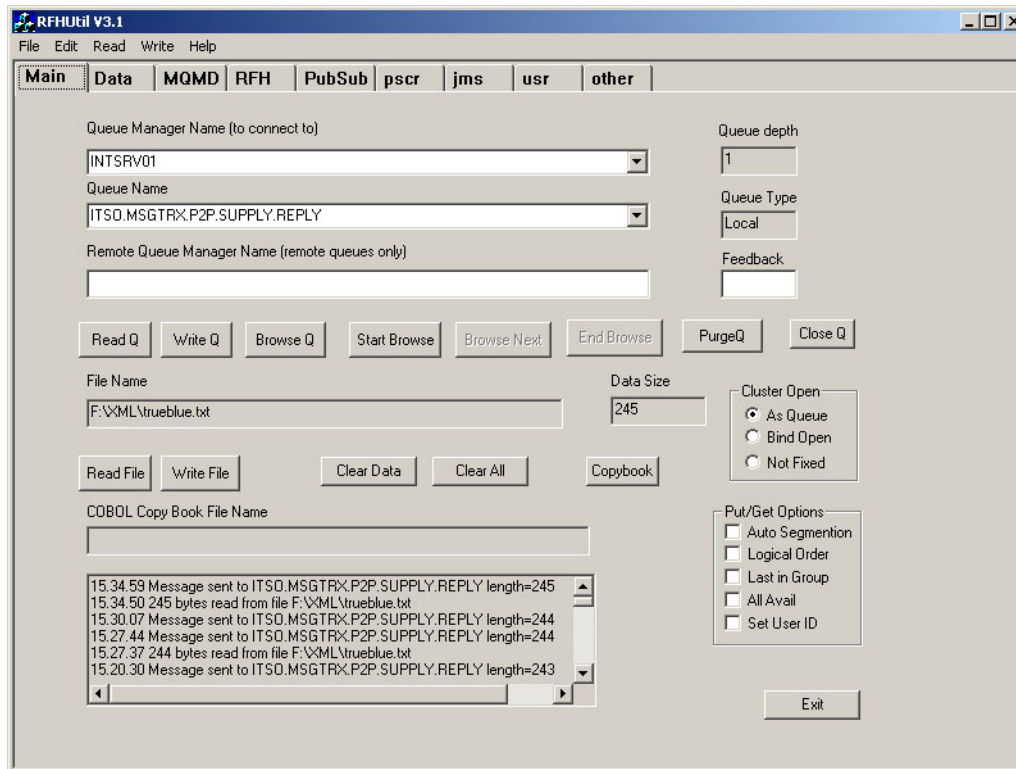


Figure 13-9 RFHUtil application

13.5 Testing, logging, debugging

Event Viewer

The standard logging mechanism for WebSphere MQ on the Windows platform is the operating system's logging facility. Using the Windows' Event Viewer, developers can determine the problems with the application when errors occur on the WebSphere MQ runtime level.

MQ Services

MQ Explorer is a standard WebSphere MQ application on the Windows platform to manage WebSphere MQ queue managers.

MQ Explorer

MQ Explorer is a standard WebSphere MQ application on the Windows platform for managing WebSphere MQ objects.

RFHUtil application, SupportPac IH03

RFHUtil is a utility to read from, write into, and browse WebSphere MQ queues. You can find more information about the utility at “Testing message flows during development” on page 277.



A

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG246875>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246875.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
SG246875.zip	The sample application and configuration instructions Zipped

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	5MB for the code
Operating System:	Windows 2000 Server with Service Pack 3
Processor:	1 GHz or higher
Memory:	1 GB or more

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Web material zip file into this folder. The contents of the .zip file are organized in directories, you will find further instructions about installing and configuring the application in Chapter 11, “Technical scenarios” on page 227.

Note: If you do not want to configure everything step-by-step, you can use the scripts, with some modifications from the runtime environment configuration manual.

Load the application into WebSphere Studio

The workspace for WebSphere Studio is packaged in the downloadable .zip file, unzip the file to a directory, for example: C:\SG246875.

You will find the *install* directory after unpacking the .zip file, open the *index.html* under this directory to get detailed information about the development and runtime environment configuration.

The development environment requires the following configuration steps in order to work:

1. Start WebSphere Studio with the following command:

```
wsappdev -data C:\SG246875\Resources\Workspace
```

The directory is where the workspace for the project can be found.

2. Select **Window -> Preferences -> Java -> Classpath Variables**, then create a new variable with the name MQ_JAVA and the path <WebSphereMQ_root>\java\lib.
3. Import the projects to the workspace. Select **File -> Import -> Existing Project into Workspace**. Click **Next**.
4. Click **Browse** next to the Project Contents field, then select the **CommonUtility** folder under the workspace directory (C:\SG246875\Resources\Workspace).
5. Click **Finish**.
6. Repeat steps 3 and 4 with the following projects:
 - VendorFrontEndUtility
 - VendorFrontEndWeb
 - VendorFrontEnd
 - VendorBackEndUtility
 - VendorBackEndEJB
 - VendorBackEnd
 - SupplierApplication
 - WMQIMessageFlows
7. After importing all existing projects, select **Project -> Rebuild All** from the menu. At the very end you should see only four warnings under the Tasks view. If there are more, make sure that all the projects are imported.
8. Regenerate the deployed code for the EJBs. On the J2EE perspective, J2EE Hierarchy view select **EJB Modules -> VendorBackEndEJB**. Right-click the item and select **Deploy and RMIC code...**
9. On the next window click **Select all**, which selects all the EJBs, then click **Finish** to generate the code.
10. Check the Tasks view to see if there is any error. There should be only four warnings. If you find one or more, make sure you fix them before you proceed.
11. The application assumes that your Web application can be found at the <http://localhost:9080/ivcorp> URL. If that is not the case you have to modify an XSL file accordingly and re-compile the translets belonging to this XSL.
 - a. Open the ProcessOrderReply.xsl file from the VendorFrontEndUtility -> xsl folder.
 - b. Change the <http://localhost:9080/ivcorp/catalog/catalog.xml> in the file according to your environment.

There are two places in the file where you can find this string! This string tells the style sheet where it can find the catalog.xml file to build the

catalog for the presentation. If you have the .xml file at another URL, for example under an HTTP server, you can also use that URL.

- c. Also change the URL in the file. Click **CommonUtility -> com.ibm.itso.msgtrx.Constants.java**, and look for the CATALOG_URI variable.
- d. Delete the frontendtranslets.jar file from the VendorFrontEnd folder.
- e. Select **File -> New -> Other** from the menu, then select **XML -> Compile XSL**. Click **Next**.
- f. Provide the following information for the compiler:
Input folder: /VendorFrontEndUtility/xsl
Destination folder: /VendorFrontEnd
Java package: com.ibm.itso.msgtrx.frontend.translets
Jar file name: frontendtranslets.jar
Click **Finish**.

12. Follow the steps from the configuration guide distributed in HTML format together with the sample application.

13.5.1 Supplier application configuration

The following is a sample .xml configuration file for the supplier application.

Example: A-1 supplier-conf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<supplier-conf>
  <runmode>point2point</runmode>
  <supplier-entity name="Red supplier">
    <supply>red</supply>
    <queuemanager>SUPPLIER01</queuemanager>
    <queue>ITSO.MSGTRX.P2P.SUPPLY.RED</queue>
    <reply-queue>ITSO.MSGTRX.P2P.SUPPLY.REPLY</reply-queue>
  </supplier-entity>
  <supplier-entity name="Green supplier">
    <supply>green</supply>
    <queuemanager>SUPPLIER01</queuemanager>
    <queue>ITSO.MSGTRX.P2P.SUPPLY.GREEN</queue>
    <reply-queue>ITSO.MSGTRX.P2P.SUPPLY.REPLY</reply-queue>
  </supplier-entity>
  <supplier-entity name="Blue supplier">
```

```
<supply>blue</supply>
<queuemanager>SUPPLIER01</queuemanager>
<queue>ITS0.MSGTRX.P2P.SUPPLY.BLUE</queue>
<reply-queue>ITS0.MSGTRX.P2P.SUPPLY.REPLY</reply-queue>
</supplier-entity>
</supplier-conf>
```

The configuration elements:

- ▶ `<runmode>` sets the mode to point2point. At this stage only point2point is the supported running mode.
- ▶ `<supplier-entity>` has the details of each supplier. The application starts multiple threads, as many entities are defined.
- ▶ `<supply>` refers to the supplied product.
- ▶ `<queuemanager>` is the name of the queue manager, the listener is listening on.
- ▶ `<queue>` is the name of the queue, the listener is listening on.
- ▶ `<reply-queue>` is the name of the queue where the reply is sent.

The supplier application is a stand-alone Java application. You can start it from the command line using the `runsupplier.bat` script. Before you run it, open it and make sure that the `JAVA_HOME` and `MQ_HOME` variables are set correctly.

The application sends log messages to the standard output. You should be able to see the incoming messages from the integration node and the outgoing messages from the supplier replying to the orders.

13.5.2 Running

Running the sample is quite simple.

1. Once the servers, the broker, and the supplier application are up and running, open a browser and access the application on the front-end server at the following URL: `http://<server_name>/ivcorp`.
2. Select the order link and place an order with the system. Select a color, set the ordered amount, and submit the form.
3. As a result, a page should come up with the order ID. It is a long string with numbers. Copy the ID to the clipboard.

If there was an error, then start checking the system where the error occurred. You can find information about testing, logging, and debugging in the next section.

4. Go back to the main page, select the order pickup link, provide the order ID (paste it from the clipboard), then submit the form.
5. The resulting page should come up and show the original color of the order and the resulting color that can be fulfilled.

Abbreviations and acronyms

API	Application Programming Interface	J2EE	Java 2 Enterprise Edition
BI	Business Intelligence	JAAS	Java Authentication and Authorization Service
BMT	Bean-managed Transaction Demarcation	JCA	J2EE Connector Architecture
BPM	Business Process Management	JCE	Java Cryptography Extension
CA	Certificate Authority	JCL	Job Control Event
CICS	Customer Information Control System	JMS	Java Messaging Service
CMT	Container-managed Transaction Demarcation	JNDI	Java Naming and Directory Interface
CSS	Cascading Stylesheet	JSP	JavaServer Page
DCE	Distributed Computing Environment	JSR	Java Specification Request
DHTML	Dynamic HTML	JTA	Java Transaction API
DMZ	DeMilitarized Zone	JTS	Java Transaction Service
DNS	Domain Name System	JVM	Java Virtual Machine
DOM	Document Object Model	LDAP	Lightweight Directory Access Protocol
DSS	Decision support systems	LOB	Line Of Business
DTD	Document Type Definition	MDB	Message-driven bean
EDI	Electronic Data Interchange	MIS	Management Information System
EIS	Executive Information System	MOM	Message Oriented Middleware
EJB	Enterprise JavaBean	MQ	Message Queue
ERP	Enterprise Resource Planning	MVC	Model-View-Controller
HACMP	High Availability Cluster Multi-Processing	ODS	Operation Data Storage
HTML	HyperText Markup Language	OLTP	On-Line Transaction Processing
HTTP	HyperText Transfer Protocol	OMG	Object Management Group
IBM	International Business Machines Corporation	OO	Object Oriented
IIOP	Internet Inter-ORB Protocol	ORB	Object Request Broker
ITSO	International Technical Support Organization	PDA	Personal Digital Assistant
		PKI	Public Key Infrastructure
		RDBMS	relational database management systems

RMI	Remote Method Invocation
RPC	Remote Procedure Call
RYO	Roll-Your-Own
SGML	Standard Generalized Markup Language
SOAP	Simple Object Access Protocol
SQL	Structured Query Language
SSL	Secure Socket Layer
TCO	Total Cost of Ownership
UDDI	Universal Description, Discovery and Integration
UI	User Interface
VSAM	Virtual Storage Access Method
W3C	World Wide Web Consortium
WSIF	Web Services Invocation Framework
XHTML	EXtensible HyperText Markup Language
XML	eXtensible Markup Language
XSL	eXtensible Stylesheet Language
XSL-FO	XSL Formatting Objects
XSLT	XSL Transformation

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 291.

- ▶ *Access Integration Pattern Using IBM WebSphere Portal Server*, SG24-6267
- ▶ *MQSeries Publish/Subscribe Applications*, SG24-6282
- ▶ *WebSphere Studio Application Developer Programming Guide*, SG24-6585

Other resources

These publications are also relevant as further information sources:

- ▶ *Patterns for e-business: A Strategy for Reuse*, by Jonathan Adams, Srinivas Koushik, Guru Vasudeva, George Galambos, published by IBM Press, ISBN 1-931182-02-7
- ▶ *The Five Axes of Business Application Integration* by Charles Brett, published by Spectrum Reports Ltd., 2002, ISBN 0-954518-1-X
- ▶ *IBM Security Architecture: Securing the Open Client/Server Distributed Enterprise*, SC28-8135
- ▶ *WebSphere MQ Integrator Administration Guide*, SC34-5792
- ▶ *An Introduction to Messaging and Queuing*, GC33-0805
- ▶ *WebSphere MQ Queue Manager Clusters*, SC34-6061
- ▶ *WebSphere MQ for z/OS System Setup Guide*, SC34-6052
- ▶ *WebSphere MQ Publish/Subscribe User's Guide*, GC34-5269
- ▶ *WebSphere MQ Security*, SC34-6079

Referenced Web sites

These Web sites are also relevant as further information sources:

- ▶ Patterns for e-business Web site
<http://www.ibm.com/developerWorks/patterns>
- ▶ WebSphere Application Server
<http://www-3.ibm.com/software/webservers/appserv/>
- ▶ WebSphere MQ
<http://www-3.ibm.com/software/ts/mqseries/>
<http://www-3.ibm.com/software/ts/mqseries/messaging/>
- ▶ WebSphere MQ Integrator
<http://www-3.ibm.com/software/ts/mqseries/integrator/broker/>
- ▶ IBM DB2 8.1
<http://www-3.ibm.com/software/data/db2/udb/v8/>
- ▶ CrossWorlds
<http://www-3.ibm.com/software/integration/cw/>
- ▶ HOLOSOFX
<http://www-3.ibm.com/software/ts/mqseries/workflow/holosofx/>
- ▶ Sun's Java Web site
<http://java.sun.com>
- ▶ Apache's XML Web site
<http://xml.apache.org>
- ▶ MD01 SupportPac for WebSphere MQ
<http://www-3.ibm.com/software/ts/mqseries/txppacs/md01.html>
- ▶ JSR 156
<http://www.jcp.org/jsr/detail/156.jsp>
- ▶ BTP specification
<http://www.oasis-open.org/committees/business-transactions>
- ▶ Middleware Spectra's Web site
<http://www.middlewarespectra.com>
- ▶ Apache Jakarta's Struts Web site
<http://jakarta.apache.org/struts/index.html>

- ▶ Apache's Xalan Web site
<http://xml.apache.org/xalan-j>
- ▶ Command pattern article at the IBM developerWorks Web site
<http://www-106.ibm.com/developerworks/patterns/index.html>
- ▶ IA0H SupportPack for WebSphere MQ Web site
<http://www-3.ibm.com/software/ts/mqseries/txppacs/ia0h.html>
- ▶ Tivoli Access Manager's Web site
<http://www.tivoli.com/products/index/access-mgr-e-bus/>
- ▶ IBM's security Web site
<http://www-3.ibm.com/security/index.shtml>
- ▶ Tivoli's security solutions Web site
<http://www.tivoli.com/products/solutions/security/news.html>
- ▶ Sun's J2EE Web site
<http://java.sun.com/j2ee/docs.html>

How to get IBM Redbooks

You can order hardcopy Redbooks, as well as view, download, or search for Redbooks at the following Web site:

ibm.com/redbooks

You can also download additional materials (code samples or diskette/CD-ROM images) from that site.

IBM Redbooks collections

Redbooks are also available on CD-ROMs. Click the **CD-ROMs** button on the Redbooks Web site for information about all the CD-ROMs offered, as well as updates and formats.

Index

Numerics

1-PC 216
2-PC 216
80/20 situation 3

A

Accountability 186
Agent application pattern 23
APPLET tag 67
Application level security 206
Application patterns 5, 12, 20
Application server node 36
As-is Host application pattern 22
ASR *See* Automatic Speech Recognition
Asynchronous messaging 121
Authentication 193
Authenticity 186
Authorization 194
Automatic Speech Recognition 70

B

Back-end application tier 25
Bean Managed Persistence 80
bean-managed transaction 134
Best practices 5
 Java Message Service 143
 WebSphere MQ 143
Best-practices 17
Binding databases 267
Binding mode 261
Bindings mode 139
Bluetooth 71
BMP *See* Bean Managed Persistence
Broker 170
Broker domain 170
broker persistent store 267
Business patterns 5, 7
BytesMessage 120

C

Cascading Style Sheets 63
 Validator tools 64

Cell scope 236
cHTML 74
Client container 62
Client mode 140, 261
Cluster Workload Exits 222
Cluster Workload Management Algorithm 221
Clustering 142, 262
CMP *See* Container Managed Persistence
Collection object 120
Command Assistant 269, 272
Common Object Request Broker Architecture *See* CORBA
Component integration 171
Composite patterns 5, 10
Confidentiality 186
Configuration Manager 172–173, 267, 269–271
Connection manager 213
Connection Pooling 212
ConnectionFactory 137
Container Managed Persistence 80
container-managed transaction 133
Control Center 172–174, 269
control commands 263
Controller 73
CORBA 81, 92
Correlation ID 121
CSS *See* Cascading Style Sheets
Customer Information Control System
 Resource adapters 87

D

Data Transformation 91
Database
 ODBC 268
DB2 V8.1 53
Declarative Security 196
decomposition 41
Decomposition application pattern 22, 40
Decomposition pattern 27
Decomposition tier 29
Destination 137
DHTML 74
Directly Integrated Single Channel application pat-

- tern 21
- Directory and security services node 37
- Distributed queuing 219
- DOM API 162
- Domain firewall 38
- Domain firewall node 34
- Domain Name System node 34
- DTD editor 165
- Durable subscriptions 128
- Dynamic HTML
 - DHTML 61, 63
- Dynamic Web services 85

E

- EAI *See* Enterprise Application Integration
- ECMA-262 65
- ECMAScript 65
- EJB
 - Local interface 105
 - Remote interface 105
- EJB container 79
- EJB interfaces 105
- EJB modules 81
- EJB *See* Enterprise JavaBeans
- EMBED tag 67
- Enterprise application 81
- Enterprise Application Integration 82
- Enterprise JavaBeans 79
 - Bean Managed Persistence
 - Container Managed Persistence
 - Entity beans 79
 - Message-Driven Beans 79
 - Message-driven beans 124, 129
 - Session beans 79
- Enterprise Resource Planning 86
- Entity EJBs 79
- ERP *See* Enterprise Resource Planning
- Existing applications and data node 37
- Extensible Stylesheet Language Transformations 69, 76

F

- Facade Pattern 110
- Five points for security 187

G

- Guidelines 5, 17

H

- HTML 61, 63
 - Validator tools 63
- HTTP tunneling 67
- hub-and-spoke 117

I

- IBM Directory Server 190
- IMAP 81
- i-mode 70
- IMS JCA resource adapter 87
- INETD 266
- Integration Bus Topology 117
- Integration hub 116
- Integration patterns 5, 8
- integration server 41
- Integration server node 36, 272
- Integrity 186
- Interaction Controller 73

J

- J2EE 59, 62, 73, 86
- J2EE 1.3 88
- J2EE Connector Architecture 86
 - Advantages 87
 - Disadvantages 88
 - IMS resource adapter 87
- JAF *See* JavaBeans Activation Framework
- Java 2 Platform, Enterprise Edition *See* J2EE
- Java applets 65
 - Disadvantages 66
- Java Message Service 88
 - Advantages 90
 - Asynchronous 121
 - Best practices 143
 - Consumers 129, 143
 - Disadvantages 90
 - Error handling 144
 - Java bindings mode 139
 - Java client mode 140
 - JNDI 136
 - Message models 112
 - Message types 119
 - MQSeries support 90
 - Point-to-point messaging 114
 - Producers 128, 143
 - Publish /subscribe messaging 115
 - Synchronous 121

- Java Naming and Directory Interface 81, 136
- Java Runtime Environment 67
- Java Transaction API 81
- JavaBeans 74
- JavaBeans Activation Framework 81
- JavaMail 81
- JavaScript 62, 65
- JavaServer Pages 74
- JMS
 - Acknowledgements 127
 - Client identifier 126
 - Message 260
 - Queue 260
 - Queue manager 260
 - Transactions 127
- JMS Provider
 - WebSphere Application Server 236
- JMSEException 144
- JNDI *See* Java Naming and Directory Interface
- JRE *See* Java Runtime Environment
- JScript 64–65
- JTA *See* Java Transaction API

L

- LDAP 81, 190
- Link level security 205
- listener 266
- Load balancer node 35
- Local home interface 105

M

- Macromedia Flash 62
- MapMessage 120
- message 277
- Message Broker services 91
- Message Facade Pattern 112
- Message flow 92
 - Testing 277
- message flow category 174
- Message format translation 171
- Message models 112
- Message Oriented Middleware 91
- Message selector 143
- Message timeout 143
- MessageConsumer 136
- Message-Driven Bean 218
 - Deployment 253
 - Development 152

- Implementation 152
- Life cycle 154
- onMessage 153
- Message-Driven Beans 79
- Message-driven beans 124, 129
- MessageProducer 136
- Messaging 89
 - Access control 197
 - Confidentiality 198
 - Data integrity 198
 - Non-repudiation 199
- Microbrowser 71
- MIME 81
- Mobile clients 70
- Model 72
- Model-View-Controller 72, 96, 128, 148
 - Controller 96
 - Model 96
 - View 96
- MOM 91
- MQ Channels 262
- MQAI 263
- MQSC 263
- MQSeries
 - JMS support 90
- MQSeries Explorer 263–264, 266
- MQSeries Services 263
- mqsicreatebroker 272, 274–275
- mqsicreateconfigmgr 269, 271
- mqsistart 271
- Multithreaded programs 214
- MVC *See* Model-View-Controller

N

- Namespaces 145
- Naming 145
- Node scope 236
- Nodes
 - Application server 36
 - Directory and security services 37
 - Domain firewall 34
 - Domain Name System 34
 - Existing applications and data 35, 37
 - Integration server 36
 - Load balancer 35
 - Protocol firewall 34
 - Web application server 36
- Non-persistent messages 143

Nonpersistent messaging 216
Non-Repudiation 186

O

Object Management Group 92
Object Request Broker 81
ObjectMessage 120
ODBC 268, 272, 274
OMG *See* Object Management Group
One-phase commit 216
Operational Data Store (ODS) 30
OTMA 87

P

Palm-OS 70
Patterns for e-business 3
 Application patterns 5, 12
 Best practices 5, 17
 Business patterns 5, 7
 Composite patterns 5, 10
 Guidelines 5, 17
 Integration patterns 5, 8
 Product mappings 5, 16
 Runtime patterns 5, 13
 Web site 6
PCF 263
Persistent Message Storage 128
Persistent messages 143
Persistent messaging 215
Point-to-point messaging 114
POP3 81
presentation tier 24, 37
Product mappings 5, 16
Profiling 212
Programmatic Security 196
Protocol firewall 38
Protocol firewall node 34
Public Key Infrastructure 34, 77
Publish/subscribe 115, 125
publish/subscribe
 advanced functionality 171

Q

Queue Manager Clusters 262

R

RDB to XML 166

RealPlayer 62
recomposition 41
Redbooks Web site 291
 Contact us xiv
Reliable Messaging 127
Remote home interface 105
Remote Method Invocation 81
repository 267, 269
Request/reply 121
Resource adapters
 CICS 87
Resource scope 236
RMI *See* Remote Method Invocation
RMI/IIOP 81, 92
Router application pattern 22, 33, 37, 40
Router pattern 23
Router tier 24
router tier 37
Routing services 91
Runtime pattern
 Security 188
Runtime patterns 5, 13
Runtime product mapping 44
RYO (roll-your-own) programming 30

S

Sample application
 Generating IDs 181
 Keeping context 182
 Order Decomposition Flow 174
 Response Composition flow 175
 Storing messages 182
 Supplier response flow 175
sample scenarios
 Development environment 230
 Runtime environment 231
SAX API 162
SCRIPT tag 66
Securing connections 192
Security 37, 188
 Connections 192
 Rules 187
Self-Service business pattern 20
Send-and-forget 123
Server scope 236
Service oriented architecture 85
Servlets 73
Session EJBs 79

- Session Facade Pattern 111
- Shared Cluster 218
- Shared-Nothing Cluster 219
- SMTP 81
- SOA *See* Service oriented architecture
- SQL wizard or SQL query builder 166
- Stand-alone Single Channel application pattern 21
- Static Web services 85
- stdin 263
- StreamMessage 120
- Struts 97, 148
 - Action 151
 - FormBean 150
 - Resource bundle 152
- Struts-config.xml 150–151
- SupportPac MS03 172
- Swing 65
- Synchronous messaging 121
- System hardening 193
- SYSTEM.ADMIN.SVRCONN 266

T

- TextMessage 120
- Text-to-Speech 70
- Thin clients 61
- Threading and concurrency 218
- Tiers
 - Back-end 25
 - Decomposition 29
 - Presentation 24
 - Router 24
- Tivoli
 - WebSEAL 190
- Tivoli Access Manager 190
- Topic-based security 207
- Transaction
 - Attributes 133
- Transaction Mode 182
- Transactional service 91
- TTS *See* Text-to-Speech
- Two-phase commit 216

U

- User node 34

V

- Validator tools

- CSS 64
- HTML 63
- VBScript 64
- View 73
- Voice-enabled applications 70
- VoiceXML 74

W

- WAP *See* Wireless Application Protocol
- Web application server 71
- Web application server node 36
- Web browser 62
- Web client 61
- Web container 73
- Web diagram 148
- Web modules 81
- Web server redirector node 35
- Web services
 - Advantages 85
 - Disadvantages 86
 - Dynamic 85
 - Static 85
- WebSEAL 190
- WebSphere Application Server
 - JMS provider 236
 - V5 50
- WebSphere JMS
 - Queue 242
 - Queue Connection Factory 237
 - Queue Destination 240
- WebSphere MQ 143, 145
 - Best practices 143
 - Caching 217
 - Cluster 218
 - Clustering 142–143
 - Exits 222
 - High Availability 218
 - Management 222
- WebSphere MQ classes for Java 138
- WebSphere MQ classes for JMS 138
- WebSphere MQ Integrator
 - Broker 170
 - Cloning messages 177
 - Configuration manager 170
 - Control center 170
 - Create queue managers 172
 - Decomposition of messages 177
 - Decomposition using ESQL 178

- Define queues 172
- Filter nodes 179
- Input node 177
- Routing 179
- Sample application 174
- Storing messages 182
- Sub-flows 184
- Transaction behavior 276
- V2.1 53
- WebSphere MQ JMS 243
 - Queue 247, 253
 - Queue Connection Factory 243
- WebSphere MQ V5.3 52
- WebSphere resource scope 236
- Windows CE 70
- Wireless Application Protocol 62, 70–71
 - Microbrowser 71
- Wireless Markup Language 62, 71, 74
- WML *See* Wireless Markup Language
- WMLScript 71

X

- XA resource coordinator 140
- Xalan parser 162
- XCF 87
- XForms 69
- XHTML
 - Extended HTML 68
- XML 61, 71, 74, 160
 - DTD editor 165
 - Encryption 77
 - RDB to XML 166
 - schema editor 165
 - SQL wizard or SQL query builder 166
 - transformation 162
 - XML and SQL query wizard 166
 - XML schema editor 165
 - XML to XML mapping 166
 - XPath expression wizard 165
 - XSL debugging and transformation tool 165
 - XSL editor 165
- XPath expression wizard 165
- XSL
 - debugging and transformation tool 165
 - editor 165
- XSLT 160
- XSLT *See* Extensible Stylesheet Language Transformations

- XSLTC 162
 - Compiler 163



Self-Service Applications using IBM WebSphere V5.0: and IBM MQSeries Integrator

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Redbooks

Self-Service Applications using IBM WebSphere V5.0 and IBM MQSeries Integrator

**Self-Service
applications using
Router and
Decomposition
patterns**

**WebSphere
Application Server
V5, MQ V5.3, MQ
Integrator V2.1**

**Patterns for
e-business solution
design**

This IBM Redbook introduces the Router and Decomposition application patterns for Self-Service e-business applications. The book discusses the messaging and transactional capabilities of an application. This redbook is a valuable source for IT architects, IT specialists, application designers, application developers, system administrators, and consultants.

Part 1, "Patterns for e-business", introduces the Patterns for e-business concept, focusing particularly on the Self-Service business pattern and the Router and Decomposition application patterns.

Part 2, "Guidelines", provides guidelines for messaging and transactional applications, including application design and development and some of the non-functional requirements for such applications, including security and system management and performance.

In the Appendix, you will find details on how to set up and configure both the development and runtime environments for the sample application discussed in this book.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks